Autonomous Real-time Object Detection and Identification

Gruffydd Morris

Supervisors: Professor Plamen Angelov
Dr Graham Lovegrove
Dr Graeme Knight

School of Computing and Communications
Lancaster University

Thesis for the degree of
Doctor of Philosophy

The County College October 2017

I would like to dedicate this thesis to my loving parents, and my forever friend in heaven, Jason Morgan.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Gruffydd Morris
October 2017

Live as if your were to die tomorrow. Learn as if you were to live forever.

Mahatma Ghandi

I believe that failure comes from giving up. If you never choose to fail, if you never choose to give up, then you're just in the process of making it happen.

Jeb Corliss

Acknowledgements

The author would like to acknowledge Professor Plamen Angelov for his dedication and guidance throughout this and previous works. In a similar breath, in is incumbent on me to recognise the strength and guidance of my parents Mrs Vivienne Morris and Dr Stephen Morris. Furthermore, the production of this work would not have been possible without the funding and support from EPSRC, Thales (UK) Ltd, and the Smith Institute, specifically Dr Graham Lovegrove, Dr Graham Knight and Dr Tim Boxer.

Abstract

Sensor devices are regularly used on unmanned aerial vehicles (UAVs) as reconnaissance and intelligence gathering systems and as support for front line troops on operations. This platform provides a wealth of sensor data and has limited computational power available for processing. The objective of this work is to detect and identify objects in real-time, with a low power footprint so that it can operate on a UAV. An appraisal of current computer vision methods is presented, with reference to their performance and applicability to the objectives. Experimentation with real-time methods of background subtraction and motion estimation was carried out and limitations of each method described. A new, assumption free, data driven method for object detection and identification was developed. The core ideas of the development were based on models that propose that the human vision system analyses edges of objects to detect and separate them and perceives motion separately, a function which has been modelled here by optical flow. The initial development in the temporal domain combined object and motion detection in the analysis process. This approach was found to have limitations. The second iteration used a detection component in the spatial domain that extracts texture patches based on edge contours, their profile, and internal texture structure. Motion perception was performed separately on the texture patches using optical flow. The motion and spatial location of texture patches was used to define physical objects. A clustering method is used on the rich feature set extracted by the detection method to characterise the objects. The results show that the method carries out detection and identification of both moving and static objects, in real-time, irrespective of camera motion.

Table of contents

List of figures

List of tables

Glossary

Term	Description
Candidate objects	Candidate objects are texture patches that represent objects of interest.
CEDAS	Online clustering process defined in [57]
Computer Vision	The field is interdisciplinary that deals with how computers or artificial devices can gain a high level of understanding from images or video streams
Edge clustering	The same process as edge linking, the terms may be used interchangeably.
Edge linking	The process of joining contiguous edge pixels together to form texture patches
F-measure	Measures the average squared colour error of the segments, penalizing over-segmentation by weighting proportional to the square root of the number of segments. It requires no user-defined parameters and is independent of the contents and type of image.
Texture patch	Used to describe the area of textured pixels linked together by the edge linking of WISE. They form candidate objects
Unmanned Aerial Vehicle (UAV)	An unmanned aircraft that is typically used by the military and border protection services to gain advanced reconnaissance. They usually have on-board cameras, and limited computational processing power.
WISE	Within-Image Spatial Edge Flow algorithm

Chapter 1

Introduction

This chapter introduces the scope of the research undertaken in this thesis. Firstly it considers the wider field of data analysis and data gathering through the use of optical devices, and then considers application specific problems. How computer vision as a whole is employed in a variety of ways to tackle these problems is introduced followed by scoping the particular problem area of this research and the questions associated with it. The scope of the work is condensed into the specific research conducted throughout the remainder of the thesis. Chapter 2 specifically deals with the research papers relevant to the field of research, and how existing research can contribute to resolving the questions proposed here.

1.1 Scope

Data analysis is a wide ranging field that can be applied to any number of data gathering and output problems. It can be summarised as a series of data inputs, data recording, data processing and some kind of information output, figure 1.1. The sub domains of data analysis are dependent on the type of data input, the type of processing to be performed and the expected information output. The research conducted in this thesis is centred around analysing video data obtained from a scene, determining the objects and their type that are present in the scene.

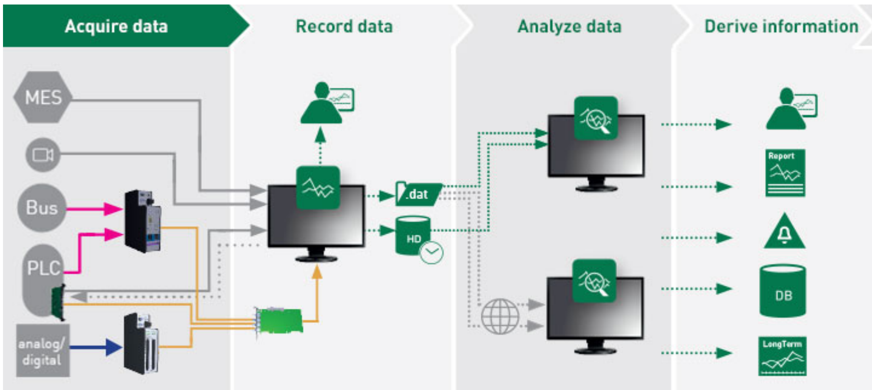


Fig. 1.1 The process of data analysis

The general emphasis of the investigation task is to be initially as broad as possible, incorporating as many computer vision concepts and their application potential to the problem. Many of the topics are overlapped with each other (e.g. classification and behavioural analysis) and using multiple concepts can reinforce the end objective. Many of the complex computer vision systems draw from all criterion, and many of the categories to form coherent objective solutions. Examples such are facial recognition [127] and motion estimation [145]. There are also system level applications using computer vision algorithms to support a wider system, for example the work by Luo et al [82] is a prosthetic eye system that uses elements of the computer vision field for its operation. It is important to consider each system level algorithm for its constituent components because the systems can compensate for any individual algorithm short coming. For example with the case of using background subtraction as part of the motion estimation; whilst the individual algorithms perform the designed job well when combined into a system of components the design performance hinders overall performance. Possible reasons for this scenario are that the algorithms are being used outside their designed operation or the algorithms are not ideal, but are being used as a "best-fit" solution when the ideal solution is not available. In either case, there is an argument for redesign of either the algorithm, or the system in which it is trying to be used. Consequently, it is important for us to understand the goals that drive the algorithms. There are many goals in computer vision, primarily motivated by whatever system it is being integrated with. However one overarching goal of the field as research is the ability for computers to perceive their environment as good as, if not better than, human vision. The next section considers the motivation for the research proposed in this thesis.

1.2 Motivation

In a world where sensors for data acquisition are used on an ever expanding scale, there is a requirement to efficiently process and interpret the data into meaningful information. There are millions of cameras; in the UK alone there are an estimated 5.9 million surveillance cameras in operation [142]. A much greater number of optical cameras are included on mobile phones, and are now estimated to outnumber the human population on the earth [20]. Additionally, cameras are gathering data from vehicles (such as aircraft, border patrol teams, satellites, unmanned aerial and ground vehicles, and amateur video recordings). The total volume of data gathered by these devices is astronomical, and far outweighs the time available to humans to review all of the gathered data. This all contributes to the mass volumes of video data being recorded and stored, some with useful information or important observations that, at present, require human observers to extract them. Considering just the CCTV context of the UK, over 141 million hours of video is recorded every day. That means everyone in the population would have to watch 2.2 hours of surveillance video each day to get through the entire recorded data. The number of hours each person would have to watch to cover all the gathered CCTV data is large and impractical. There are, in reality, many less operators to view all this information and many will need to review several cameras at once to try to identify problems, issues and incidents [63]. This is without taking into account the human factors such as attention span and sleep requirements of an operator [133]. The automation of surveillance camera systems would go a long way to ensuring that security requirements are met, such that useful information or important observations are not missed; as well as to reduce the load and pressure on operators. To this end, there is the goal of enabling computers to interpret visual data from these optical devices and process the data in a meaningful way to produce useful information as its outputs. The field of research into this technology is collectively known as computer vision.

Cameras are regularly used as sensors on unmanned aerial vehicles (UAVs) as reconnaissance and intelligence gathering systems [49] [103] and used for support of front line troops on operations [31]. The cameras on these vehicles can be of the order of 1 – 2 gigapixels with frame rates of the order of 25 -100 frames per second, meaning the data is gathered at terapixels per second, that is 3 -5 terabytes of information per second [9]. As this reconnaissance data is gathered, operators on the ground have to sift through each frame looking for important objects or points of interest to support the operations [31]. Figure 1.2 shows an example of a frame from a UAV, and the small object of interest that each operator is expected to see.

The proposed application area of the research is on Unmanned Aerial Vehicles (UAVs). This platform provides a wealth of sensor information and has limited computational power

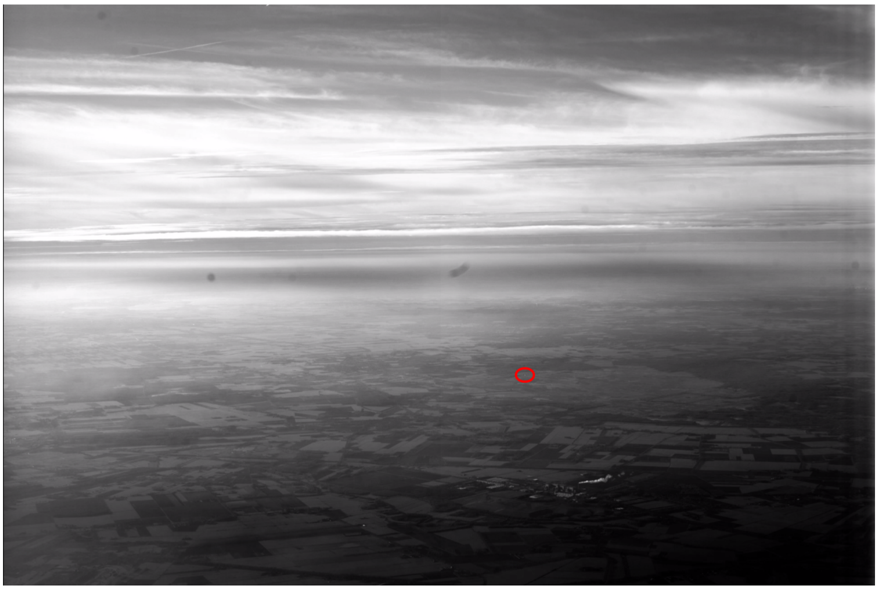


Fig. 1.2 An example frame from which operators need to detect this small target

that promotes algorithm efficiency. The potential application is however not limited to the UAV environment, and would be suitable to any sensor application that would require detection and tracking of features of interest. The outline of the research is to:

Returning to the UAV application, using the cameras on board the UAV, :

- Develop systems to automatically detect and track dynamic objects or features of interest in a real time live video stream environment. The development would be highly computationally and memory efficient, lending itself to being used on platforms with limited computing power such as UAVs.
- Combine these highly computational efficient online, real-time information extraction algorithms with capable self-learning algorithms that can detect and track objects in a live sensor environment with dynamically changing scenery.

1.3 Computer Vision Goals and Human Replication

The aim of the decision making component of computer vision analysis is to achieve similar, if not better recognition of objects autonomously by computers than by humans. The major obstacle to this goal is that computers are "dumb" terminals. A stream of bits looks exactly the same to the computer as another stream of bits; if a computer is just given a stream of RGB pixels, the output on the screen will simply be a visual display of the scene the sensor is pointed at, plus some error component. Errors are introduced by flaws in the camera detection and signal transfer to the output display. Software in the modern era is sufficiently advanced to enable computers to take a set of inputs, process them and provide a set of intelligent results that gives the impression of autonomous behaviour. Research into computer vision has allowed computers to take or receive a visual input as stimuli (data), conduct some processing (compute) and provide intelligent results on its output (information). The visual stimuli for the purposes of this work are data from colour cameras; and is referred to as simply a camera from herein. The generated image from this camera is made up of red, green and blue picture elements (RGB pixels).

In humans, vision is the main sensory input used to assess an environment. The other senses are used to enhance the detail of the environment, but vision is a supremely efficient method of assessing one's surroundings. There are many articles of research into how the human vision system may work, with several differing opinions. Given the efficiency of human vision systems, and being the central component to assessing surroundings it is widely accepted that an efficient computer based vision system will go a long way to achieving autonomous environment understanding by the wider artificial intelligence field. One of the most important parts of human vision, and thus computer vision, is object separation and discrimination. The process of computer vision may include augmenting the existing video data such that humans can interpret the data faster [112]. For example, colour profile analysis augments the information available within the image to show the distribution and variance of colour in an image. There are also techniques that manipulate the colour palette of the image; this type of technique can suppress colours of a certain type and augment others based on specific highlighting goals. For the most part, these algorithms are used to augment images such that humans can interpret them quicker, and can identify target objects quicker. Computer vision can also be built into a wider autonomous system where the computer makes decisions based on the processed video stream. The difference with current computer vision algorithms, compared to humans, for autonomous detection and identification is that the algorithms do not inherently detect and identify everything in a scene. Each algorithm has specific objectives in terms of what it needs to detect and identify. For example, in the case of Edge Detection, the objective is to detect contrast contours in the frame. The algorithms do not do anything with textures or object formation, they focus on detecting the contours and later algorithms are used for identifying objects and textures. Alternatively, image segmentation algorithms are focussed on detecting objects and textures. The ultimate goal is to achieve all types of detection and identification in one system, which would bring the computer vision implementation closer to that of humans. The next chapter presents some of the existing vision systems that are components that make up small parts of a vision system, and other techniques bring together many components to achieve an all round detection and identification system.

1.4 Real-time algorithms

In this research there is a consistent reference to real-time processing or real-time analysis. Real-time systems provide a constraint on the computer system in which the data must be analysed and the results outputted. This is also known as computational deadlines [108]. Being correct and real-time does not mean just outputting a correct calculation, it is also dependent on being within the time constraint [130]. The time constraint can be hard (safety critical systems for example), whereby missing any time deadlines constitutes a system failure, firm (network packets for example) where by missing some time deadlines are tolerable but the service or operational speed of the system may be degraded. The work packet is useless if the time deadline is missed (e.g. retransmission of network packet if arrival deadline is missed). Finally there are soft real-time systems, whereby the performance is degraded if the time deadline is missed, and the usefulness of the work packet is degraded if the time deadline is missed - e.g. system fault and condition data. It is important to identify that brute force processing does not necessarily yield a real-time system [136]. A computer may have high data and processing throughput (such as a Cray supercomputer [123]), however the results may still not be provided for several minutes (complex protein folding for example [147]). Conversely a low power system, with limited processing capacity may be considered as real-time, despite its limited capacity, if the result of the system is returned in sufficient time to immediately affect its environment e.g. temperature regulation of a green house - the windows and heaters are controlled in real-time to regulate the internal temperature of the green house. Referring back to the brute force type processing, the computing power industry is consistently following Moore's Law, and in some cases exceeding it [93]. In the graphic shown in figure 1.3, the cost per GFLOP [104] is decreasing at rapid rate, and is expected to continue in the immediate future. There is therefore scope for using parallelism on existing computer vision algorithms (such as optical flow, or image segmentation) using the brute force of this computational power to make them operate in real-time, [110]. There are some drawbacks to this approach.

- There is only so far the processing power can be optimised, and eventually the advancement of processing capability will slow (earliest estimates are 2020 or 2022 [83] [66] [132].

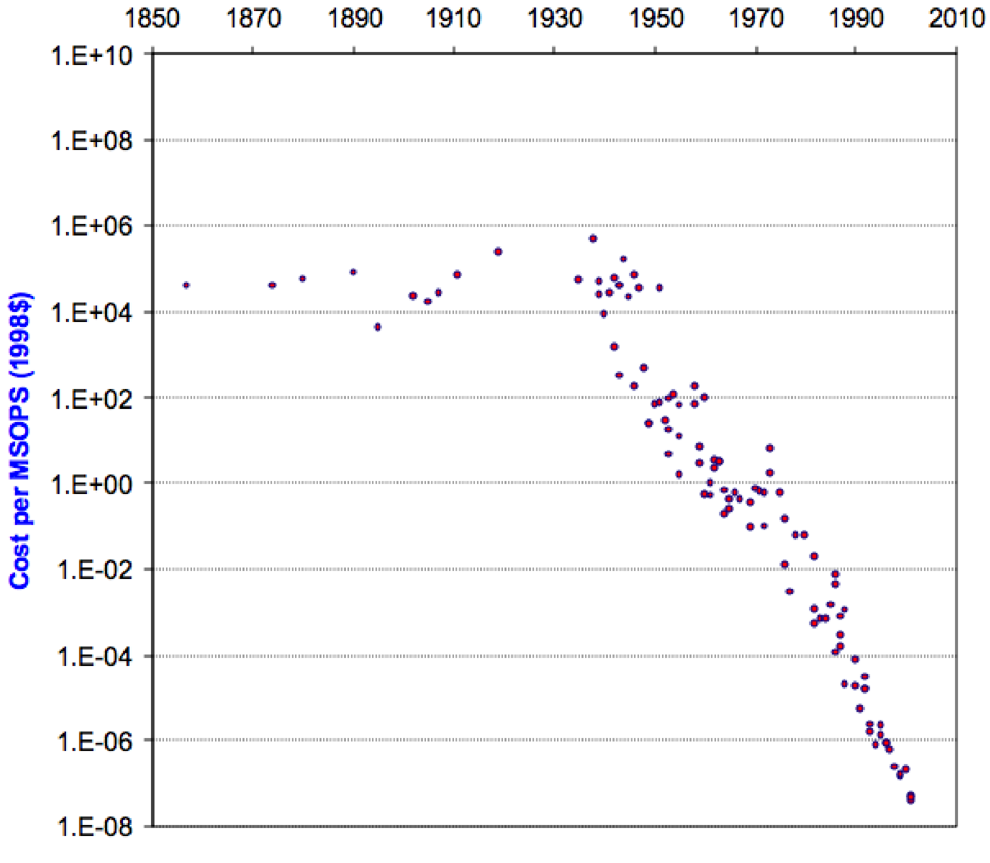


Fig. 1.3 The progress of computing measured in cost per million standardized operations per second (MSOPS) deflated by the consumer price index [104]

- If the image resolution is sufficiently large, even with the huge processing capabilities, real-time capability will eventually be reduced or unattainable.
- Not all application areas will be able to utilise the high powered processing due to other system or environmental constraints (e.g. the power capacity of a UAV is limited, thus the processing power available will also be limited).
- Despite the cost per GFLOP reducing, using a brute force approach does not address the requirement of cost outlay for high powered computing systems that would be required to make some algorithms real-time capable
- The complexity of some algorithms are such that not all aspects are necessarily suitable for parallelism. Thus the gains are not necessarily directly proportional to the increase in processing power.

It is important to note that real-time is often confused with an on-line system, or at least the terminology is used independently. Whilst they are similar, and both relate to the time of the processing, they are also distinctly different. Online or offline processing refers to when the system begins processing the data. An online system is continuously processing data as it arrives into the system, and does not wait for a collection or data set to be gathered before processing the available data. Conversely an offline system waits for the entire dataset to be gathered, and then subsequently processes the data. An offline system can be real-time, provided that it yields the results of the processing in a timely manner after the data has begun being processed / analysed. Equally an on-line system does not have to be real-time, whilst the data processing is continuous and begins as soon as a data sample is received, the result may not have to be returned with some time constraint. An example of a system that is on-line yet not real time is the SETI program, whereby the data samples are processed as they are received from the Archiebo telescope, yet the results of the analysis of the particular data set are not outputted for several hours [99].

Given the analysis of brute force approaches, over the coming chapters, the research focusses on the development of true real-time computer vision algorithms that are capable of running on low power computing processors, yet can be scaled on high power computing devices should there be a requirement to do so. For the requirements established earlier in the chapter, the system developed must both be real-time and online.

1.5 Autonomy or Intelligence

Humans could be considered either a very lazy species, or one that likes to optimise tasks such that other tasks can be accomplished simultaneously. There are clear examples of both in the technological world that we live in [44]. To that end, many of the electronic systems that we see in daily life can be considered, to a degree, autonomous. That is, the machines are given a task to do, and they will conduct the processing required until the task is complete and provide and output, without any intermediate intervention. If you consider the use of a washing machine, the user puts in clothes and the washing powder, selects a cycle, and presses go. This could be considered an initialisation of the autonomous system. During the wash cycle, provided no error occurs, the machine will autonomously (i.e by itself) wash the clothes and discard the dirty water. However, the automation is limited such that it cannot compensate for unknown or unforeseen scenarios [138] [11]. To address these scenarios, input from the user is required. Intelligent systems address this by having a flexible interpretation of the input and have ability to handle unexpected or unknown scenarios, autonomously without necessarily having human intervention [42]. Some of these can be supervised systems, such that there is an intelligent agent yet operators feed in additional information or parameters to support the decision making process [124]. Other, fully unsupervised techniques, do not require the input of parameters or intervention by users to learn about new and unknown scenarios [115]. The intelligence is often referred to as machine learning, whereby the system is interpreting its environment and creating new rules or constraints for autonomous operation based on different and changing inputs.

Referring back to computer vision and the work presented in this book, autonomy is needed to satisfy the UAV conditions and constraints. A level of intelligence is also required such that interpretation of new or unknown images is possible by the system. Given the diversity of the world environment, the expected level of intelligence is a semi-supervised approach similar to that of Zhu [159] such that object separation is conducted yet classification of the objects is conducted with the assistance of operator input.

Chapter 2

Computer Vision and Existing Research

Computer Vision has grown exponentially over the last 30 years such that it is a large research field, with many approaches that are derived to solve a wide range of vision problems. Latterly, the human objective to employ autonomous agents such as driver-less cars and unmanned surveillance vehicles has fuelled more research and funding into the field of computer vision. This chapter looks through the wealth of computer vision techniques available and what benefits and drawbacks some of the techniques have.

2.1 The Field of Computer Vision

The field itself can broadly be broken down into a number of sub categories, each of which have their own objectives in terms of data analysis and output. That is:

- Image Enhancement - image denoising, brighness and Gamma corrections, histogram analysis
- Transformations - Homography, Affine transforms, Warping, Data space manipulation
- Filtering, Fourier transforms and Image Compression - Image analysis, optimisation and size reduction (compression)
- Colour Vision - Colour mapping, colour management, colour profile analysis
- Feature extraction - Edge Detection, Corner Detection, Key-point Detection
- Pose Estimation - visual geometry, orientation and angle estimation, projections and modelling
- Registration - cross correlation, image segmentation, optical flow, particle filters

- Visual Recognition - feature transforms (SURF, SIFT), object recognition, posture and gesture recognition, facial and finger print recognition

The sub-categories fit three main criteria of image and video processing:

- **Image manipulation.** In this criteria, the processing is focussed on filtering, denoising and optimising the image itself without any notion of "what" is in the image. The processing may augment or highlight certain objects, and de-emphasis others, but this is mainly a process where the augmentation and de-emphasis are tuned based on the objects the user wants to see more or less of. The categories that fit into this criteria are Image Enhancement, Filtering, and Colour Vision although there may be some crossover from Registration and Transformations

- **Detection.** This criteria primarily focussed on detecting features, key-points and candidate objects in the image. The detection phase is an essential part of image understanding such that computer systems can understand and interpret a scene. In some cases detection can be pixel feature extraction [50] [75] and detection of contours and edges [33] [23]. It can also be in the form of detecting important points in the image such as keypoint detection [78] [14] [2] and optical flow detection [81] [55] [157] [149] or detecting the pixels potential associated together (candidate objects) such as background subtraction [137] [37] [4]. The categories that fit this criteria are Feature Extraction, Registration, and Visual Recognition

- **Image Understanding.** Technically, this criterion could not exist without the existence of one or both of the previous criteria. It is to do with interpreting and analysing the image such that situational and environmental understanding of the scene or image can be achieved. The understanding can be achieved following some image manipulation; for example if the result of the manipulation yields two distinct image colours, a level of understanding (provided appropriate rules are present) of the image can be achieved. Similarly if there are a series of candidate objects present from the detection phase, an understanding of these objects can be achieved through further processing. Categories in this criteria are pose recognition, visual recognition and transformations. Whilst the latter two are also applicable to the previous criterion, aspects of them are directly applicable here (such as facial recognition; the keypoint detection extracts the appropriate detections, and then this phase applies the matching analysis to a database or reference image)

The scope of the project as defined in the brief is such that a number of solutions, with components in computer vision, could be assessed. These are:

1. Novelty detection in a moving image plane
2. Object identification
3. Behaviour analysis – anomaly detection, trajectory analysis
4. Tracking of one or more detected objects in the image frame
5. Collaborative (swarm) of UAVs working together to achieve a common goal
6. Exploration of parallelisation in software agents

Each aspect contains different types of research, and over the next paragraphs the details of each area are explored.

2.1.1 Novelty Detection

Novelty detection is the first phase of computer vision image processing. The principle is to detect a foreground novelty from the clutter of the background. One of the most commonly used methods is background subtraction using KDE (Kernel Density Estimation). This relies on generating a statistical model of the background of an image that is representative and discriminates from new foreground novelties in the image [37]. The complexity required increases markedly when the background itself is not constant (as it would be with a moving camera). Here, two distinct problems exist and define the direction of the research:

1) The offline computational requirement of KDE is not suitable for UAV applications; the requirement is to have an online, real-time processing of the image data.
2) With the background no longer a constant, subtraction of the background using the traditional KDE will result in high noise, potentially leading to false detections.
The computational requirement can be reduced through using recursive algorithms to estimate the density of a pixel based on the similarity to the pixels at the same position in previous image frames. A recursive algorithm can discard the frame once it has been processed, and the density information for each pixel can be accumulated over time. The memory requirements are, therefore, much smaller, and consequently the volume of data to process is significantly reduced. Research into the background modelling of a moving image may take some input from algorithms such as SIFT [78] and SURF [14] and likewise (BRISK [74] and FREAK [2] also have some interesting aspects, and are more accessible from an IPR point of view). These algorithms are able to discriminate between background and an initialised object to enable the tracking of the object through a moving image frame.

These approaches, however, assume the objects are to be initialised manually. Research into merging or combining both recursive background subtraction and SIFT / SURF approaches is a possible progression which could yield a highly discriminatory novelty detection capability in a dynamic video stream which is both robust and computationally efficient [78] [14]. One particular development area of interest is to detect novelties in terms of a new patch / object even if it is not moving. For example, comparing with a previous days images and identifying that this new patch / object was not there previously (examples of such could be mobile SAM sites, military camp, hostages etc.).

2.1.2 Object Identification

Whilst detection of novelties in an image frame is an important first stage in image analysis, there is nothing at this stage to identify whether this is an object of interest or not – one could say that the information itself has not yet been extracted, just “potential” information areas. To identify foreground objects, some sort of clustering and/or classifier or labeller is needed to clearly distinguish objects. A common approach to identifying key areas is image segmentation using clustering and/or classification of the feature space. Image segmentation is a method to partition the feature space by labelling pixels that share similar visual features or properties, and connecting the pixels with the same labels in some meaningful way [102]. There are approaches such as region growing, watershed, clustering and fuzzy set techniques, which are either for a specific domain or image field, and require significant offline processing to work. In the work of Othman et al [107] it is proposed that an Evolving Fuzzy Inference System is used to classify objects within a static MRI. This approach is based on the eClass semi-supervised classifier [4]. An alternative is to consider supervised learning, with an expert user inputting feedback to indicate “correct” results during the training phase [40]. The classifier can evolve to incorporate the fed-back information, which in turn improves the classifier performance. Work has also been done using evolving clustering and classification to remove the supervised element of the object identification, and also to be “online” – analysis in real time [4] but this was not applied to video analytics. The difficulty with these approaches are that:

1) They consider ideal, static camera environments with little or no noise. One of the investigative areas will be to look at techniques that are robust at identifying and discriminating individual objects when the background and platform is dynamic (the motion causes increased noise, novelty occlusion, interference from the proximity of other potential novelty detections, false detections).
2) The majority of solutions either require supervision or are offline learning classifiers.

3) In the case of the evolving clustering / classification, whilst objects are identified, because of the online nature there is no determination as to what importance the identified objects have. There is also no indication of whether the behaviour of the identified object is "correct" (see behaviour analysis later). Whilst the evolving, unsupervised model is desirable in an unknown environment from the point of view of identifying previously unseen objects, an element of domain "correctness" is required for the UAV application. As a result, we plan to investigate a semi-supervised / unsupervised model which would suit the application area better. This means that identification of the objects in the video stream is proposed to be conducted in an unsupervised manner, but with the proviso that the operator / analyst can review the identified objects and update the classifier with "correctness" measures in an ad hoc manner (i.e. not required to update the model for every data sample, but review it when it is convenient, reducing the demand on the operator compared with a fully supervised classifier, whilst increasing domain "correctness" of the model).

2.1.3 Behaviour Analysis

It follows that (as alluded to earlier) analysis and classification of the behaviour of novelties or identified objects in the video stream is also desirable (behaviours can be, but are not limited to, kinematic – motion in the video stream, or perhaps visual – dynamic brightness / hue / saturation / illumination changes). This is beneficial so that when two objects of very similar initial visual properties appear in the video stream, they can be classified separately according to their behaviour. Classifying the objects according to appearance in a video stream is the first part; classifying similar objects by discriminating behaviour over a series of images is an extension of this. The plan is to extend the detection and identification techniques developed early on to explore the potential of identifying and classifying of behaviours. By studying behaviours it will be possible to identify normal and abnormal behaviours of an object in a video stream (behaviour "correctness"). Equally, it is desirable to identify objects with certain behavioural patterns so that future predictions on the objects trajectory or visual variance can be inferred – further aiding the capability to detect, identify and discriminate specific objects in the video frame.

2.1.4 Tracking

The investigations into the discussed areas leads the work into another area of interest and has already been considered for stationary cameras [6] – tracking. Currently, the solutions that exist do not efficiently track more than one object in a moving image, or only track more than one object in a video stream that is stationary. Many of the current approaches that are used are cumbersome or processor intensive tasks that are not well suited to the UAV application which is in a dynamically changing environment and is computationally limited. There is scope to advance and develop this area of research, working with methods such as SIFT (Scale invariant feature transform) [78] and SURF (Speeded up robust features) [14] which currently require manual initialisation of the objects of interest (BRISK [74] and FREAK [2] are also applicable). It holds that if the object can be autonomously identified through successive image frames, and the feature morphing or change of the object detected over these images, logically it will be possible to accurately track this object across the video stream.

2.1.5 Collaboration and Parallelisation

Further work is proposed in the area of both collaborative (swarm) operation of UAVs working together to achieve a common goal; augmentation of individual capabilities through information sharing, and the exploration of parallelisation in software agents; not just for the UAV application. Collaboration is a capability that has been attempted before using mobile robots for localization [43] (2006 patent). Sharing information and experience between UAV or sensor platforms could lead to augmented detection, identification and tracking capabilities beyond what is possible with a single sensor platform. In addition, specific to the UAV application, it could, potentially, allow the operator to be able to direct and control more than one UAV at the same time; each having a task within a global goal / objective. An extension of this would be to automatically identify tasks by the UAVs for a particular goal or objective with no operator input. In addition, approaches that utilise the concepts of image stitching [87] [72] could be developed to work across the collaborative platform suggested here. In the field of camera surveillance, such as road traffic cameras or CCTV monitoring cameras, the collaborative nature could lead to cross-camera coordination to track a vehicle or subject of interest across multiple cameras without the need for the operator to intervene. This would be an invaluable capability, considering that currently when a subject of interest moves out of the field of view the operator must manually identify which direction the subject of interest is moving in order to continue surveillance.

2.2 Relevant Research

Computer vision is a wide-ranging field covering a multitude of image analysis scenarios. It is necessary to explore the different research already in existence that can contribute to the goals set out in this project. The initial scoping suggested exploring areas such as tracking and parallelisation, along with behaviour analysis. Some of this has been explored in terms of current research, however given the scope of the project the main focus of the research is into detection and identification of objects. In some cases, there are several components that make up the solution (e.g. Motion Estimation), and the relevant research covered here looks at both the solution level and the individual algorithms. The material also considers the importance and relevance of real-time and autonomous algorithms.

2.2.1 Novelty Detection

In order to achieve a goal in computer vision and analysis, one must understand what is being analysed and why. If you take a moment to look around the room, your eye, brain, and associated neural connections quickly identify various objects around the room within a fraction of a second. The identification process uses multiple features, understanding object behaviour, and trajectory; and at a higher level, its threat level. Humans then interpret the output image on the display and identify objects in the field of view of the camera by linking appropriate pixels that form objects. A computer is somewhat different; there is no immediately apparent link between each RGB pixel on the screen and thus recognition of the objects in the field of view is not possible. Novelty detection is a method of low level detection such that a link between pixels can be detected autonomously by a computer. The process of linking pixels also allows for higher level analysis by non-computer vision algorithms. Novelty detection can broadly be divided into two domains, static images or moving images. In the static images, the analysis is conducted across the image space and spatial domain. The location of pixels are fixed and reference points in the scene remain constant. In moving images the analysis is conducted over spatial, temporal or both domains. The main activity is to identify differences between two images temporally separate. A further complication can arise in this scenario of the camera also being in motion. This provides additional challenges due to there being no fixed spatial reference points. The following are leading techniques in static image analysis:

- Edge Detection
- Corner Detection
- Keypoint Detection
- Image Segmentation

The leading techniques in moving image analysis:

- Background Subtraction

- Optical Flow

When the camera is also moving, the following are leading techniques:

- Dense Scene Optical Flow
- Image Stitching
- Motion Estimation

2.2.2 Edge Detection

Edges form one of the several features that compose an image, and the edge detection methodology focuses on analysing a scene or frame estimating the edges of objects. Edges, in terms of a visual scene, are significant contrast changes in one direction or another, and can typically form the boundary between objects. Interestingly, edge detection also appears in signal processing (usually 1-D edge detection), and so much of the maths used to derive edges in signals can be transferred in some capacity to the 2-D image space (such as Gaussian convolution, Laplacian transforms and Gabor filters). In general, edges can be classified as two different types, ramp or roof type edges (see figure 2.1).

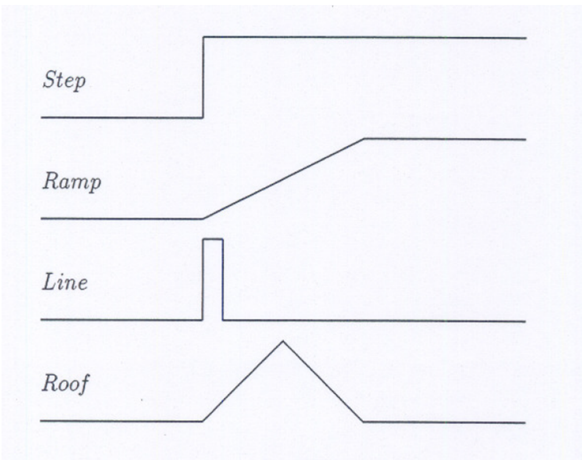


Fig. 2.1 One-dimensional edge profiles. [58]

It is unlikely, or certainly rare in real world signals, to get a crisp step or line edge due to contrast boundaries not being as sharp as these. This is mostly down to the capture technology which interpolates and adds low frequency components to the boundaries yielding the ramp or roof style edge. Both step and line type edges can be generated in artificial test images. Measuring of correctly detected edges can be subjective if just a visual reference of the image is taken. A better more quantifiable method is edge counting. This is where each edge in a scene is counted, and the resultant output of the edge detector is counted. These can yield true positives (actual edges in the scene that were detected), false positives (detected edges that do not appear in the scene) and false negatives (edges that are in the scene but were not detected). In a real world scenario, it is difficult to describe a true edge vs a false edge due to the complexity of textures and image angles, and therefore it is common practice to describe the performance of an edge detector against a known artificial image. The gradient magnitude of an edge in its simplest form is the differential of the intensity against a particular axis, so in the x-axis this would be the formula:

$$G(f(x)) = \frac{(dI)}{dx} \tag{2.1}$$

For continuous, non-digital images it is usual to define the x and y directions in terms of maximum gradient (thus the x-axis is the angle along the maximum gradient). The interest for this project is in digital imagery however, and thus the x and y axis remain as the digital axis depicted by pixels. One of the earliest examples of utilising gradients to detect edges in an image is the Roberts Cross operator, which uses the above principle in 2-Dimensional space to extract gradients [121]. Roberts proposed the equation:

$$G(f(i,j)) = |f(i,j) - f(i+1,j+1)| + |f(i+1,j) - f(i,j+1)| \tag{2.2}$$

which results in intensity changes in a diagonal direction. The equation can be shown as two kernels [58] figure 2.2

$$G_x = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array} \qquad G_y = \begin{array}{|c|c|}\hline 0 & -1 \\ \hline 1 & 0 \\ \hline \end{array}$$

Fig. 2.2 Roberts Operator. [58]

The computed gradients are provided at the interpolated point $\left[i+\frac{1}{2}, j+\frac{1}{2}\right]$ The Roberts operator is simple and efficient but lacks noise tolerance, and its simplicity with respect to modern day computers does not offset its lack of noise tolerance. A method by Erwin Sobel, [135] was introduced which avoids the necessity for an interpolation point by using a 3x3 operator. The Sobel operator is computed with partial derivatives:

$$s_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) s_y = (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4) \tag{2.3}$$

and the gradient magnitude calculated by:

$$G = \sqrt{s_x^2 + s_y^2} \tag{2.4}$$

Similar to the Roberts operator the Sobel operator is used as a convolution mask with images:

$$S_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad S_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Fig. 2.3 Sobel Operator. [58]

This operator uses a constant with the partial derivatives such that the pixels directly adjacent to the center mask pixel have more of an emphasis.

In contrast to the Sobel operator, Prewitt [114] developed an operator that also uses a 3x3 kernel, but does not place any emphasis on neighbouring pixels. An excerpt from [58] shows the comparison of edge gradient extraction over the operators discussed which can be seen in figure 2.4

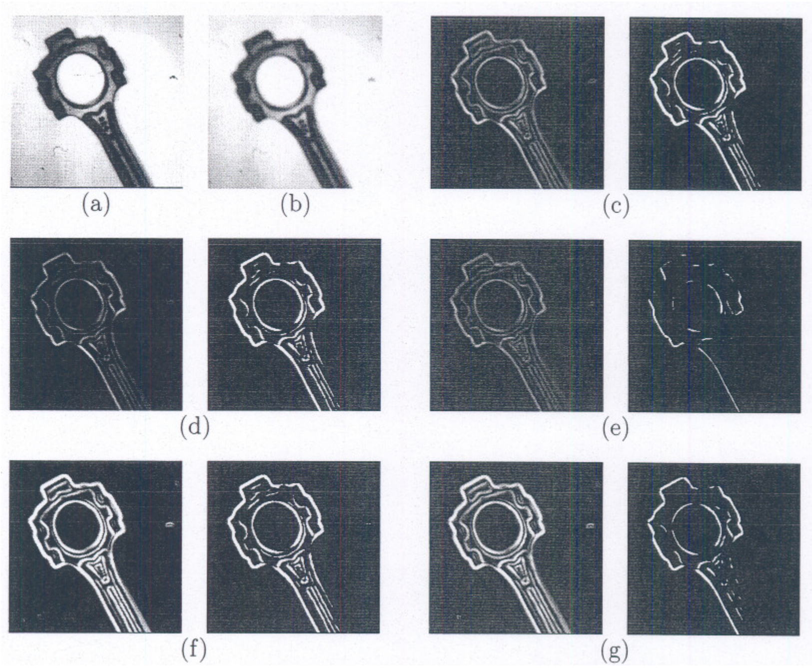


Fig. 2.4 A comparison of Edge Detectors. a) Original image b) Filtered image, c) Simple gradient using 1 x x2 and 2 x 1 masks, d) Gradient using 2 x 2 masks, e) Robert cross operator, f) Sobel operator, g) Prewitt operator [58]

Further work in Edge Detection has been done by using the second derivative of the gradient. The advantage of using the second derivatives is that at the zero crossing point, this indicates a local maxima in the gradients. The Laplacian is used in the two-dimensional version to obtain the second derivative of the gradients. The Laplacian of $f(x,y)$ is

$$\nabla^2 f = \frac{d^2 f}{dx^2} + \frac{d^2 f}{dy^2} \tag{2.5}$$

The following partial differential equations can be approximated:

$$\frac{d^2 f}{dx^2} = f[i,j+1] - 2f[i,j] + f[i,j-1] \tag{2.6}$$

$$\frac{d^2 f}{dy^2} = f[i+1,j] - 2f[i,j] + f[i-1,j] \tag{2.7}$$

This yields a mask that can be used to approximate the Laplacian, or second order derivative of the gradient 2.5.

$$\nabla^2 \approx \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$

Fig. 2.5 Laplacian Operator, derived as the second order differential [58]

One of the limitations in using the Laplacian second order differential is that it is highly sensitive to noise, and any noise artifacts apparent in the first order derivatives are going to provide a zero crossing detection in the second derivative. In the paper by [86], they propose a solution to the noise problem of zero-crossing second derivatives by adding a Gaussian filtering stage and smoothing, and following this with a Laplacian to obtain the zero-crossing points. The filtering removes the noise, but also widens potential edges and as such the zero-crossing local maximas are important to extract. The zero-crossing Laplacian output is then convolved with the image to yield the edges, which should be relatively noise free. The Gaussian filter and subsequent Laplacian zero-crossing is shown here:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1-\frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.8}$$

The limitation of using the Gaussian filter is primarily down to the smoothing constant which is applied to σ. Widening the filter reduces the noise further but also smooths the edge gradients which can lose resolution.

In the work by Canny [23], the problem of error smoothing and edge definition loss is addressed through the use of non-maxima suppression. The image is convolved with a Gaussian, as with Marr and Hildreth [86], and results in a smoothed image. The gradient of the smoothed image is then approximated using first difference approximations, usually using the Sobel or Prewitt operator.

One of the limitations of using these kinds of edge detectors is that little is suggested about the internal structure of any objects. The early methods such as Roberts operator [121] analysed edges but were susceptible to noise. The later edge detectors are less susceptible to noise, and define clear edges. All the methods throughout the convolution are either losing information (the sharp edges lose the gradient information) or are susceptible to noise. As mentioned earlier, the Edge Detection methodology can be likened to signal processing. Gabor filters used in conjunction with images, proposed by Mehrotra et al [88] provide an optimal balance between frequency resolution and time / spatial resolution. By convolving the filters with an image, at multiple angles across the image it extracts feature descriptors of edges in each direction. The Gabor filter is a linear filter, and the frequency and orientation representations successfully model the visual cortex of mammalian brains (thus linked to the thought of similarities in human perception) [85] [30].

2.2.3 Corner Detection

Edge detection can be used as a component of corner detection. A corner is thus defined where two edges intersect, or a point where there are two or more different edge directions in a local region. Corner detection has also been called detection of interest points, however this can lead to confusing terminology. Corner detection, along with key point detection is often used in conjunction with image understanding activities such as motion detection, video tracking, image segmentation and object recognition. An early example of corner detection are Moravec corners [94]. In this work, Moravec describes the existence of corners as neighbouring overlapping regions with low similarity. The similarity is calculated through the sum of square distances measure. The logic of this derivation is that pixels with overlapping regions of similar intensity will most probably be part of texture or some uniform area, pixels with overlapping regions that are different, but with parallel regions being similar are likely to indicate the pixel is on an edge, where as where overlapping regions intensities are most different indicate the presence of one or more edges, suggesting a corner in the local region. The issue, identified by Moravec in the work, is that it is not isotropic; an edge is must be present in the direction of the neighbours (horizontal, vertical, or diagonal), otherwise incorrect interest points will be selected [95]. Harris and Stephens [51] improve on the Moravec corner detection, by removing the dependence on isotropic patches. Instead, they propose taking the differential of the corner score with respect to the direction of the intensity gradient. The Harris operator or matrix used to calculate the corner gradient similarity is formed from a weighted sum distances equation:

$$G(x,y) = \sigma_u \sigma_v w(u,v)(I(u+x,v+y) - I(u,v))^2 \tag{2.9}$$

where I is the two-dimensional image, (u,v) is an image patch area, and (x,y) is the patch shifting distance. The equation, through using the Taylor expansion method can be approximated to:

$$G(x,y) \approx \sum_u \sum_v w(u,v)(I_x(u,v)x + I_y(u,v)y)^2 \tag{2.10}$$

$$G(x,y) \approx (xy)M \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.11}$$

Where M is the structure tensor [19] where the angle brackets denot averaging over (u,v):

$$M = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \tag{2.12}$$

As with the Moravec corners, a large variation in all directions (x,y) characterises a corner. Given that M is the structure tensor [67], the eigenvalues of the matrix M can represent interest points based on their value. If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$ there is no point of interest at the location; if $\lambda_1 \approx 0$ and λ_2 is large this generally indicates the presence of an edge (gradient in the perpendicular direction, small or no gradient in parallel direction); and if λ_1 and λ_2 are large this indicates the presence of a corner (as with Moravec, large differences in each direction). Calculating the Eigenvalues can be computationally expensive (as identified by [51]). To improve on computational efficiency they propose calculating the determinant and trace of M, with an empirically derived tuning parameter (n) applied to the trace:

$$M_c = \lambda_1 \lambda_2 - n(\lambda_1 + \lambda_2)^2 \tag{2.13}$$

Jianbo and Tomasi [60] observe intensity variations are bounded by the maximum allowable pixel value in the window such that λ cannot be arbitrarily large, thus they propose only accepting corner indications where the condition $min(\lambda_1, \lambda_2) > \rho$ holds, where ρ is a predetermined constant.

2.2.4 Key point detection

The concept of key point detection is widely used in computer vision for a variety of applications, and is a mathematical extension of the concept of edge detection and corner detection; unique feature points in an image space. A common use of keypoint detection is fingerprint recognition [90] [131], although some early methods use Harris [51] corners or Smith's method [134]. As with corner detection, the objective is to identify points in the image that are robust and consistent descriptors of invariant points in an image. Scale Invariant Feature Transform (SIFT) [78] is a scale and rotation invariant key point detector. The method is applied over several stages, the first of which is the detection of scale-space extrema through the use of a difference of Gaussian function. The objective here is to identify the areas of an image that are invariant to scale, that can be repeatably assigned from different views of the same object. The continuous scale space function was introduced by Witkin [155]. The scale space of an image is found by the convolution of a variable scale Gaussian function with an input image. To detect stable key point locations, a difference of Gaussian function is computed between two nearby scales separated by a constant. Several octaves of this scale space are used, and between scale octaves, the Gaussian image is down sampled by two and the DoG process is repeated. Sampling in the spatial and scale space domain are applied after local extrema detection in order to identify the most robust scale space key points. The accuracy of SIFT is improved by increasing the number of scale samples the key points are detected over, at the sacrifice of processing efficiency. The work conducted in [74] is an advancement on the existing SIFT methodology described above. The objective is still the same, to detect key points and assign descriptors (features) that are invariant of scale or rotation. The authors identify that one of the limitations with the SIFT approach is the extensive dimensional vectors produced for interest points. The processing of high dimensionality is proposed as the weakness. Other methods as [62] apply PCA to the high dimensionality that yield faster computations, although they suffer from being less distinctive features than the original SIFT approach. The detector in SURF uses a Hessian matrix, and the determinant of the Hessian to derive location and scale for the detector. This optimises the computational complexity and thus the processing speed. For efficiency, the Gaussian filters are approximated as this has been found not to degrade performance; they are approximated with box filters. Similarly with the descriptors, the complexity is reduced significantly/ The Haar wavelet responses are reduced to the vertical and horizontal sample points. The responses are weighted with a Gaussian centred at the interest point to increase robustness to geometric deformity. It was shown in Bay et al [14] that the so called fast Hessian detectors reduced processing compared to Difference of Gaussian by a factor of 4, and a factor of 6 for Hessian-Laplace detectors. The Hessian threshold can be adapted to increase robustness, at the cost of computational performance.

2.2.5 Image Segmentation

Image segmentation [143] is used to divide and classify detections within the visual scenes without applying the assumption of motion; in fact, there is no motion preservation in image segmentation techniques. Image segmentation is applied to a single image and can be achieved sequentially over a sequence of frames to achieve detection in a video stream. There are a number of different methods employed to achieve image segmentation, each to achieve a particular goal. One of the simplest versions of image segmentation is thresholding. This effectively turns the image into a binary image based on some feature clip level (colour, brightness, hue for example). Extended methods of this thresholding appear in Otsu's method and some simple clustering techniques (k-means for example). Otsu's method relies on establishing the intra-variance of each class, and selecting a threshold such that this is minimised. The method requires a search of the entire image space for the threshold that minimises this variance. The algorithm can be computed effectively by using a recursive update of the probabilities and means, however the search of the algorithm still provides a cumbersome method that yields only a binary separation. Similarly, clustering can be used to segment an image based on the image features such as colour, intensity, pixel location and any derived features such as density or saturation. With the k-means algorithm a preconception of how many clusters into which to divide the image is required, as this is the starting parameter. Other, more complex techniques, do not require initialisation to define the clusters and can autonomously divide the data space into any number of clusters. An important drawback of clustering an image space directly is that short of simple or artificial imagery, the clustering is not guaranteed to yield complete objects given luminance and positional variances (such as an inclined light shining onto one corner). Other image segmentation methods use classification to derive detections. The classifiers are trained on a variety of versions of the objects, and then tested on previously unseen footage. This methodology is offline processing however it has been found to be an effective way to detect specific objects [109] [33]. Adaptive Texture and Boundary Encoding, [117], models a region boundary using a Gaussian distribution which is encoded by an adaptive chain code. The limitation is the assumption of a normally distributed boundary for a texture, and whilst the method works well in simple imagery, a complex image with several varying textures can lead to missed texture boundaries and over fitting. Malik et al divides the image [84] into regions of brightness and textures. This is achieved by using the brightness and texture cues as a measure of similarity for neighbouring pixels which are subsequently linked if they show similarity. To describe textures they use a component coined "textons" which is a measure of the texture property through observing filter responses. The technique can produce good segmentation of greyscale images which is one of the limitations. Also by relying on filter responses for the definition of a texture, in cases where the texture is similar to a neighbouring object (but is a different object), the method can miss these and group them together as a single texture. Efficient Graph-Based Image Segmentation is a method introduced in the work of Felzenwalb and Huttenlocher [39], they measure the evidence for a boundary between two regions using graph-based representations of the image. The result is a method that can discriminate its textures dependent on the variability of said textures. Contour Detection and Hierarchical Image Segmentation, [10], is a commonly seen image segmentation approach which uses a contour detector that combines multiple local cues into a globalization framework using spectral clustering. The segmentation algorithm consists of generic machinery for transforming the output of any contour detector into a hierarchical region tree. This leads to the reduction in the problem of image segmentation to that of contour detection. A number of research papers look into improving each of these methods, by the accuracy of the detections or decreasing computational load to achieve the same results on hardware with a lower computational capability [68], [24], [71], [26], [150].

2.2.6 Background Subtraction

The techniques discussed in this section are all mono-modal (pixel-wise) background subtraction techniques [37] [137] [125] [156]. Pixel-wise techniques treat each pixel independently from all the others. This maybe a rash assumption (because there may be underlying pixel interdependency that pixel-wise techniques will not detect) but it does lend itself to some very fast techniques that can be optimized using multithreaded processing. All the approaches considered here perform well (produce tangible results) in static observation environments only. Dynamic observations are much more complex; background subtraction techniques do not fare well and tend to produce false detections. Extra techniques are used to compensate for dynamic observation platforms which are commonly grouped as motion estimation techniques. Other non-pixel-wise techniques can use texture / edge based detections which exploit local spatial information for extracting the structural information. Noriega, [105], divides the scene into overlapping square patches for detections (the overlapping is a non-mono-modal approach) whereas Heikkila, [52], describes a model of local texture characteristics and uses fixed circular regions of pixels for comparison. Another style of approach is sampling based which evaluates a wide local area around the pixels to perform complex analysis. A spatial sampling mechanism is employed by Cristiani, [29], using pixel-region mixing. Barnich, [12], uses spatial neighbourhood sampling to refine per-pixel estimates and is loosely based on a Parzen windows process. These approaches tend to be processor intensive and do not lend themselves to efficient multithreaded implementations due to the need to compare pixels across the frame. A popular pixelwise technique (Kernel Density Estimation), which is not a real time technique, is introduced for comparison purposes with the approaches discussed.

Kernel Density Estimation (KDE)

More recently, a probability density estimation technique has been proposed in Kernel Density Estimation (KDE) [37]. This technique is not real-time however it is an important consideration as it is a common offline method for background subtraction. It is also non-parametric once it has been initialized, which is especially important for autonomous algorithms; this technique does require external input from a user or device at initialization, limiting the initial autonomous capability and opening the model to subjectivity. The KDE technique estimates the probability density function of each pixel based on a number of consecutive frames (the number of frames, or ‘window’, is fixed throughout the operation of the algorithm). The probability density function (PDF) of each pixel is calculated for the defined window of frames using a Gaussian kernel, shown in eq (2.14).

$$P(x_t) = \frac{1}{N}\sum_{i=1}^{N}\prod_{j=1}^{d}\frac{1}{\sqrt{2\pi\sigma_j^2}}e^{-\frac{1}{2}\frac{(x_{t,j}-xi,j)^2}{\sigma_j^2}} \tag{2.14}$$

Where x_t is a d-dimensional colour feature, x_i is the mean of this colour feature over N frames and σ is the bandwidth or standard deviation in the jth dimension. Each pixel in the current frame is compared with the PDF; if the pixel is sufficiently different from the mean of the probability density function, it is considered to be foreground; otherwise the pixel is considered to be background. The threshold (sigma multiple) used to determine if a pixel is sufficiently different and therefore foreground is required to be pre-selected as part of the initialization. An important consideration for KDE is the selection of the kernel bandwidth (scale). If the bandwidth is too narrow false foreground detections become a problem because of the ragged density estimate for the pixel, too wide and the density estimate will be overly smooth leading to missed detections. In Elgammal et al [37] the bandwidth is autonomously defined for each pixel, and is adaptive throughout the operation. By measuring the deviations between two consecutive intensity values, in most cases, it can be assumed that the two pixels come from the same local-in-time distribution (as only very few pixel intensity pairs are expected to come from different distributions). If the local-in-time distribution is assumed to be Gaussian, the deviation distribution $(x\neg x_{n+1})$ is also Gaussian N. For a symmetric distribution the median of the absolute deviations is defined as eq (2.15).

$$Pr(N(\mu,\sigma^2) > m) = 0.25 \tag{2.15}$$

Thus the bandwidth of the distribution can be estimated in eq (2.16).

$$\sigma = \frac{m}{0.68\sqrt{2}} \tag{2.16}$$

Where m is the median over the frames in the colour space, and σ is the bandwidth or standard deviation. The approach can be extended to include “Probabilistic suppression of False Detections” [37] which considers pixels that are neighbouring the pixel currently being analysed. This increases the robustness to noise (e.g. leaf fluttering), but also increases the processing time required for each pixel. As this process requires analysing neighbouring pixels it limits the effectiveness of multi-threaded implementations. This review is specifically focusing on pixel-wise approaches and consideration of neighbouring pixels or local region approaches is beyond the scope of the investigation. The approach makes some assumptions about the real world. The distribution of colour (or other feature) for each pixel is modelled with a Gaussian and this assumption increases the susceptibility of the model to false detections and noise, because real world features are not necessarily distributed as a Gaussian distribution. Another assumption made by this method is that the background is sufficiently static to avoid being considered as foreground, however, rapid illumination changes or leaves blowing in the breeze can introduce noise or false detections. When considering real-time applications there are drawbacks to this technique. Most importantly, the model will not run in real-time because of the window of frames that is required to be read in order to generate the probability density for each pixel. If the window is moved in an overlapping manner on the receipt of new frames the approach can get closer to true real time simulation. The approach also has a high memory cost (because of the number of frames required to be remembered).

Gaussian Mixture Models (GMM)

Despite being proposed chronologically before KDE, adaptive background mixture models allow real time analysis of a video stream by using multiple Gaussian kernels [137] to represent the colour distribution of each pixel. Each pixel is assigned to one of the Gaussian probability density functions (the number of PDFs is defined at initialization) depending on how closely the pixel properties match the PDF. The number of functions used to describe a pixel determines how robust the technique is with busy or multi-modal scenes. Typically 3 to 5 Gaussian functions are used describe background and foreground pixels but generally this is problem specific (more would be defined for a motorway than a green field for example). As the number of functions used to represent each pixel is increased, the required processing also increases which can affect the real-time capability of the approach. This technique is useful when there is a multimodal background, with the multiple Gaussians able to represent several different modes of pixels. In a very busy scene the detection performance of the approach decreases due to the number of Gaussians used being insufficient to represent each mode of the pixels. This can be improved by increasing the number of Gaussian representations at the expense of processing and memory requirements. Using a recursive method, the Gaussian functions are updated in real-time removing the need to remember every point of the history and a window of frames; the Gaussian function that the pixel matches closest is updated with the current pixel value, and once updated, the pixel value is discarded eq (2.17)

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} * \eta(x_t, u_{i,t}, \Sigma_{i,t}) \tag{2.17}$$

Where x_t is the current data sample, K is the number of distributions, ω_t is an estimate of the weight (what portion of the data is accounted for by this Gaussian) of the $i^t h$ Gaussian at time t, $\mu_{i,t}$ is the mean value of the $i_t h$ Gaussian in the mixture at time t, $\Sigma_{i,t}$ is the co-variance matrix of the i^th Gaussian at time t, and η is the probability density function defined in eq (2.18).

$$\eta(X_t,\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x_t-\mu_t)^T\Sigma^{-1}(x_t-\mu_t)} \tag{2.18}$$

With the aim of saving computational memory and speed the covariance matrix is assumed to be of the form eq (2.19)

$$\Sigma_{i,t} = \sigma_k^2 I \tag{2.19}$$

Which assumes independence between the feature variables and that they have the same variances. These assumptions are not necessarily valid in the real world, but the approach avoids processing intensive matrix inversions at the expense of accuracy eq (2.20).

$$\omega_{i,t} = (1-\alpha)\omega_{i,t-1} + \alpha(M_{i,t}) \tag{2.20}$$

Where α is the learning constant and M is defined as 1 for the Gaussian that was matched and 0 for the remaining functions. The Gaussian Mixture Model (GMM) [137] is a parametric technique requiring both the learning constant and sigma threshold to be pre-defined at initialization. The sigma threshold for assigning a match to a Gaussian distribution is (according to [137]) normally set to 2.5. Parameters for unmatched distributions are not changed. The matching distribution is updated with the new observations in eq (2.18). When a match is not found for any of the distributions, the least likely distribution is discarded and a new distribution is introduced with the current pixel value as its mean. The technique was improved by [61] to enable shadow detection and the approach later optimized by [160] to increase robustness.

Recursive Density Estimation (RDE)

As a departure from the probabilistic methods, RDE introduces a new approach to background subtraction [125] [7] [5] [116]. There is no prior assumption about the underlying distribution of a pixel's feature value. The approach calculates how near (dense) a pixel value is to all the previous pixels that have been before it. The pixel history is stored as the mean and standard deviation of the pixels from all previous frames. The mean and standard deviation are updated recursively using the formula in eq (2.21). A Cauchy type kernel is used to calculate the density of the current pixel compared with the history [5]:

$$D = \frac{1}{1+||x_t-\mu_t||^2+X_t-||\mu_t||^2} \tag{2.21}$$

Where x_t is the current data sample, μ_t is the mean of all previous data samples; X_t is the scalar product of the previous data samples. Both, the mean and the scalar product can be updated recursively as shown in eq (2.22) and (2.23) [5].

$$\mu_t = \frac{t-1}{t}\mu_{t-1} + \frac{1}{t}x_t; \mu_1 = x_1 \tag{2.22}$$

$$X_t = \frac{t-1}{t}X_{t-1} + \frac{1}{t}||x_t||^2; X_1 = ||x_1||^2 \tag{2.23}$$

Where t is the number of frames read, including the current frame. If there is no change in the scene, the pixel density does not change, and therefore the pixel is considered as a background. When there is a change in the scene, the proximity of the value of the pixel in the current frame compared to all previous frames (mean and standard deviation) changes. If this change is significant enough (large enough difference in value) the pixel is considered as a foreground. The threshold for the difference is defined using the standard deviation (sigma) of all previous frames. Usually a threshold of 2 or 3 sigma is used; by increasing the sigma there is a reduction to the sensitivity to change in the scene thus reducing the number of false detections. Too high a sigma value and the system will start to miss detections. It is a realtime, recursive technique which is highly computationally efficient. As an aside observation, the accuracy of RDE (given the variable nature of real world environments) could be improved through using a semi-supervised approach where the sigma value is updated on an ad hoc basis.

2.2.7 Moving camera domain

The moving camera domain is a more recent area of research. The principle is to achieve the same detection capability at a similar level of robustness and performance when the camera is moving, to the static camera equivalent. This is harder to achieve because there are no static reference points in the sequence of frames to conduct background subtraction. Two of the fields for analysing moving camera scenarios are Optical Flow and Ego Motion. Optical flow analyses the flow of pixels through the scene; that is it models the brightness pattern changes between frames. Ego-motion on the other hand simplifies the scene over several frames by overlapping the similar areas between frames. This overlap creates an effective static frame to conduct novelty detection as if the domain was static.

2.2.8 Optical flow

Optical flow in a video stream is a particular type of analysis that assigns a vector of motion to a local region in the video stream [56] [81]. Optical flow works by tracing pixel illumination changes through a scene, and assigning a vector to the apparent motion between the points that are tracked over time. The ordered sequence of frames enables the calculation of image velocities, or object displacement over the sequence of frames. The basis of the works assume that the brightness of a point in the frame is constant:

$$\frac{dE}{dt} = 0 \tag{2.24}$$

With the chain rule for differentiation [55]:

$$\frac{\delta E}{\delta x}\frac{dx}{dt} + \frac{\delta E}{\delta y}\frac{dy}{dt} + \frac{\delta E}{\delta t} = 0 \tag{2.25}$$

By letting the differential of x and y with respect to t equal u and v respectively a linear equation is obtained:

$$E_x u + E_y v + E_t = 0 \tag{2.26}$$

This forms the basis of optical flow with the magnitude of the movement in the direction of brightness change is:

$$M = -\frac{E_t}{\sqrt{E_x^2 + E_y^2}} \tag{2.27}$$

Horn, [55], continues to explain additional measures such as smoothness by minimising the square of the magnitude of the gradient of the optical flow. Horn also explains a second smoothness measure can be obtained through the sum of the squares of the Laplacians in each of the x and y components. This technique is useful because it does not categorise pixels in the image as foreground or background, it assigns a vector of motion to it, therefore making no assumptions about which pixels are background or foreground. Thus, objects that may be of interest but are static within the video stream are not considered uninteresting, just that they have a different vector of motion to other objects. The vector of motion is applied to all pixels in the video stream, and in the general case, contiguous pixels with the same or similar vector of motion, that are in spatial proximity (high density), can be considered the same object. An extension of this technique uses stereoscopic video streams to provide 3D disparity of pixels, enabling the separation of occluded objects [149]. The primary disadvantage of this technique is that it can take 3 to 4 seconds to process a video frame.

Also, it requires a window of several frames to conduct the analysis to provide a coherent output, utilising more system resources and extending processing time. An advantage to this technique is that it does not stitch frames together so the noise and false detections created by the ego-motion approach (described below) are not present in optical flow techniques. Additionally, in Bigun and Granlund [19], there is a suggestion that the human visual system may use techniques similar to optical flow to assign motion patterns to objects. Optical flow is the detection and tracking of brightness pattern changes in the scene. Originally developed by [55] and [81] it has proven to be a powerful method of understanding object movements in a scene. One of the drawbacks of optical flow is the high processing demand, and thus lack of real-time analysis across an entire scene. An improvement to optical flow using a derivative called "TV-L1 dense optical flow" [157] significantly improves the parallelisation capability of optical flow and thus where brute force processing power is available, provides a very good and reliable solution to moving object detection and tracking. In some cases optical flow has been used in conjunction with stereoscopic cameras, and utilising the 3D disparity between camera viewpoints increases the accuracy of detections [149].

2.2.9 Motion Estimation - egomotion

Motion estimation does not suffer from the same limitations as optical flow. The motion estimation concept works by warping the current frame into the perspective of the previous frame and stitching the frames together; the overlapping areas of both images provide a static viewpoint. The initial phase is to detect key points in consecutive frames using an algorithm such as SURF, SIFT, or BRISK. The matching phase comes next, where each key point is matched in the sequence of frames. These matched key points are used to generate a homography matrix which is applied to the current frame, transforming the pixel locations into the coordinate system of the previous frame. This overlapped area can be analysed in a similar manner to the static frame with the objective of simplifying the problem into the original static camera domain. Conducting background subtraction on a series of frames without this correction leads to several novelty detections being part of the background due to the edges of the background objects “appearing” to move relative to the sensor platform (figure 1). The main aspects of this technique can be found in the work of Fischler and Bolles [41]. This technique has some advantages; primarily, that it is faster compared to other techniques; it is real-time in some test scenarios. It also allows background subtraction methods to be used to provide comparable results with that of static video streams. At the current level of maturity there are several fundamental flaws that interfere with the performance, robustness and reliability of this approach. There is a margin of error when warping one frame into the perspective of another. If the homography matrix is not exact, there will be a variation in the pixel geometry values when it is applied to a frame. This variation will cause pixels in both frames not to line up precisely. This can lead to completely artificial novelty detections being introduced at the boundary of the misaligned pixels. Further, the approach relies on the detection of key points and matching of these key points between two neighbouring frames. Should the key point matches drop below four (minimum required to calculate homography) the system will not be able to warp the two frames and, thus, a frame will have to be discarded. Thirdly, key point detection is carried out over the entire image; should there be key points matched on a moving object within the two frames the image will be warped not just on the background changes but also on the motion of the moving object; adding to the homography distortion and thus the pixel alignment variance of what the true perspective warp should be. Whilst the approach can be run in real-time in certain scenarios, the lengthy processing chain of this approach (feature extraction, key point matching, homography, image warping / stitching and background subtraction) does not lend itself to scaling very well. The key point detection and matching take most of the processing time. A significant increase in image size or density of key points leads to a dramatic reduction in processing speed. This can be a significant disadvantage when trying to conduct additional real-time behavioural analysis on detected objects because the majority of the processing resource is taken up creating the static scene to enable novelty detection.

2.3 Research Questions

The background research into computer vision exposes several gaps in the capability of existing algorithms. These gaps can be highlighted through the postulation of research questions:

- The processing time for detecting novelties and objects increases markedly as the resolution of an image increases. How can the detection of objects remain real-time as the resolution of the images increases without the use of brute force computing?
- Can the accuracy of novelty detection be improved without an increase in processing time?
- Each existing technique makes assumptions about what is a detected object and what is considered noise (usually due to detection errors). Can an algorithm be developed such that the assumptions on detection vs noise are removed until a higher level semantic reasoning stage?

- Image segmentation techniques divide a static frame into objects with no appreciation of motion. Background subtraction extract moving objects from a sequence of frames. How can the detection capabilities of image segmentation be combined with the speed and motion retention of background subtraction techniques to achieve a real-time static and moving object detection algorithm?

2.4 Hypotheses

Based on the research questions and the background research it is possible to formulate hypotheses on the outcomes.

1. By combining the benefits of image segmentation with background subtraction, a solution that is capable of detecting static and moving objects in real-time should be possible.

2. Removing assumptions on detections will mean that the algorithms will detect all object transitions in an image. It is therefore reasonable to predict that the algorithm will be able to operate irrespective of the camera motion.

3. Once the objects are detected in an image, with the number of features available, it should be possible to type each object based on their features (cluster each object). The type association may not correlate with human differentiation of objects due to the underlying features that are being clustered.

4. The algorithms should allow for a feedback mechanism such that a higher semantic reasoning section can adapt or tune a previous layer based on detection and identification objectives.

2.5 Research Objectives

This section describes the research life-cycle. There are two main research ethoses that can be considered going forward:

1. A wide ranging project that includes many features and core functionalities not developed to full maturity, exhibiting some areas for improvement.

2. A mature project with the elements that are included in the project developed thoroughly providing a robust system that works in many scenarios, but not exploiting many techniques.

A wide ranging solution is an attractive prospect given the broad range of methods the approach could yield, however developing several arms to the work concurrently could lead to poor results and lack of robustness in the final product. Developing features progressively to a mature state has the advantage of being more robust when a new functionality is added (should give more reliable results). The mature approach means there will be less features and functionality in the final solution at the conclusion of the research. A major factor in this choice is the potential for realising poor results by opting for the "wide ranging" approach due to the immaturity of each solution. Based on this, the research will follow the mature solution ethos and will add new functionality as each reaches a performance level that is sufficient. As sufficient maturity is reached each new functionality is added to an already solid, working foundation leading to a greater likelihood of good, reliable results in the final solution.

The objectives of the research are informed by a set of constraints from the context of Unmanned Aerial Vehicles (UAVs). Currently the gathered data is sent back to the Ground Station (GS) for analysis, in order to extract important information from the image data [28]. Data traffic of this magnitude is not only costly in terms of bandwidth, it is power intensive and, thus, range limiting for the UAV [128] [38]. From the returned data, some of the analysis is conducted by off-line computation, and some by operators and analysts in real-time. To aid operators, systems such as ARGUS [72] are used to bring many images together into a single large viewpoint image to help identify targets and objects of interest in real-time. As seen in figure 1.2, a large image provides a wide viewpoint of the scene, but it is extremely difficult to spot the small object of interest; the concentration and observation skill demands on the operator are significant. The way the UAV GS works currently, even with a multiple frame analysis technique, there is a large volume of wasted data; only a small percentage of the data returned by the UAV actually contains useful information or important observations.

The UAV constraints require the focus to be on efficient operation, and the environment analysis to be assumption free; the UAV will be operating in unknown environments and assumptions made about the scenario may lead to missed information. The scope of the work will progressively explore novelty detection, object identification, behaviour analysis and tracking, with a focus on the efficiency and assumption free methodology. The objectives, pre-requisite requirements and key performance indicators provide dependencies and performance targets that indicate a sufficient level of maturity for a given function, listed in the following subsections.

2.5.1 Novelty detection in moving camera environments

Detection of stationary (background) objects

In many of the approaches currently used for novelty detection in both static and dynamic sensor platforms, any stationary object that does not dynamically move within the image frame is classed as background by the algorithm. This can lead to missed points of interest, and highlights a key flaw in the background subtraction techniques. This effect is amplified in dynamic sensor platforms when an object is moving precisely at the same velocity as the sensor platform; the object appears to the algorithm as background and is not detected as foreground. Further, in a scenario such as a police chase from the perspective of an on-board car sensor, all the cars at the same relative speed will not be in dynamic movement (or at least very little) relative to the sensor platform; the objects (cars) of interest will often be missed by current approaches.

Objective
Reliably detect relatively stationary objects of interest whilst maintaining background discrimination
Pre-requisite Requirements (PrR)
Ability to distinguish between a background object and background scenery
Key Performance Indicators (KPI)

1. Discriminate a stationary object from the background scenery in a simple (plain) environment
2. Detect several stationary objects in a simple environment
3. Detect a stationary object in a scene with a complex background
4. Detect a camouflaged stationary object
5. Perform analysis within a 10ms window per 2 MP frame

Novelty detection in the video stream without image matching or stitching

An approach to detect novelties on a dynamic sensor platform is motion estimation or ego-motion of the scene and is described in section 2.2.7. Despite the motion estimation technique being one of the fastest currently available, the analysis is marginally real-time for images >1 mega pixel, and video frames larger than this are not real-time (2 mega pixel takes approximately 500ms per frame). This time is mostly taken up by detecting and matching key points between images to enable the warping and stitching of two consecutive frames.

Objective

Develop a novelty detection approach that avoids warping and stitching images together with an aim to increase robustness and speed of the approaches.

PrR

Association of consecutive frames without the need to co-locate pixels from both frames

KPI

1. Autonomously detect a single novelty in a simple dynamic scene without stitching images
2. Autonomously detect multiple novelties in a simple dynamic scene without stitching images
3. Apply a detection method to a more complex scene
4. Operate within the performance window of <100ms for a 2 MP frame

Specific region analysis

The aim of this is to conduct local analysis on a specific region within a frame. The purpose is to reduce the overall processing required when conducting video analysis; if an area of interest is already known, then conducting analysis solely on this region can reduce the total pixels required to process. This approach should also reduce unwanted noise in the analysis that is likely to be introduced when whole frame processing is conducted e.g. leaves rustling in a separate area of the frame. Each segment can be analysed separately for novelties, points of interest, or specific features.

Objective

Autonomous analysis of local regions of interest within a frame

PrR

Frame divided into regions of interest (segmentation)

KPI

1. In a frame with a single object of interest divided into local regions of interest; successfully detect the object of interest
2. In a frame with a multiple objects of interest divided into local regions of interest; successfully detect the objects of interest
3. Apply to a complex environment scene
4. Operate within the performance window of <10ms for a 2 MP frame

2.5.2 Object analysis and advanced tracking

When novelty detection is complete clustering techniques can be used to identify the individual objects in the screen. Each object will have certain characteristics or behaviour that can help contribute to the classification of the object.

Object identification accounting for occlusion

A significant difficulty in object identification is when two objects move behind each other and become occluded.

Objective
Distinguish objects in an occluded environment
PrR
Mature novelty detection algorithm as defined in section 2.5.1
KPI

1. Separate two objects that are occluded in a simple, sparse scene.
2. Apply to multiple objects in a simple, sparse scene.
3. Distinguish multiple occluded objects in a complex, busy scene
4. Maintain analysis performance of <100ms for a 2 MP frame.

Analysis of object velocity

One of the objectives of detecting objects and novelties in a scene is to derive their behaviour. Analysis of the velocity of the objects is an important feature to be able to determine behaviour.

Objective
Successfully identify the velocity of objects traversing the scene.
PrR
Mature novelty detection algorithm as defined in section 2.5.1
KPI

1. Determine the image / pixel velocity of an object.
2. Determine the relative velocities of two or more objects.

3. Determine the absolute real world velocity of an object.

4. Achieve analysis performance of <10ms for a 2 MP frame.

Assessment of object behaviour

The analysis of the behaviour is a variable concept as objects can have eccentric movements within a scene. The purpose of this is to extract features to enable the classification of objects based on their behaviour.

Objective
Extract features that define the behaviour of an object within a scene.
PrR
Mature velocity model defined in section 2.5.2
Robust classification method available
KPI

1. Analysis of the behaviour of a single object in a simple scene.

2. Analysis of the behaviour of multiple objects in a simple scene.

3. Classify the objects based on extracted behavioural features with a minimum of 80% classification accuracy. This value arrived at from what can be expected from the ground truth of human observation (see table 5.1)

4. Analysis and classification of object behaviour to be within the performance envelope of <20ms per 2 MP frame.

Auto object classification utilising rich feature set

The previous objectives extract rich features from detected novelties and objects. This feature set can to enable improved autonomous classification of objects that are visually similar, but have distinctly different behavioural patterns.

Objective
Autonomously classify objects within a scene in real time utilising the advanced feature sets
PrR
Rich feature sets available in a mature state
KPI

1. Separate classification of two similar objects that have different behaviours.
2. Classification of multiple objects within a scene based on behaviour with a minimum of 80% classification accuracy.
3. Achieve classification online, within a performance envelope of <10ms for a 2 MP frame.

Classification of objects based on dynamic change in shape or size

Some objects exhibit dynamic changes in size or shape (despite being the same object), either due to activity or change in camera perspective. Detecting this change proves to be challenging, even for the human visual system [120], the difficulty is recognising the object as being the same as a previously seen object despite some dimensional change. There is scope to investigate using the dimensional variance as a separate feature set for behaviour, and classify based on object shape / size variance.

Objective
Autonomously classify objects within a scene based on features derived from object change.
PrR
Mature novelty detection technique as defined in section 2.5.1
KPI

1. Re-classify a single object in a simple scene as the same object after changing its physical dimensions.
2. Re-classification of the object in a complex scene
3. Conduct the analysis <10ms for a 2 MP frame.

Chapter 3

Methodology and Initial Approach

3.1 Methodology

In order to address the research questions and hypotheses proposed in 2, the approach will initially explore existing research and the limitations on the capability. Because the work is focussed on video streams, the research that focuses on analysing video streams will be used. Despite reviewing the work in 2, this exploration will underpin why the limitations of these techniques exist; operation in moving camera scenarios or why only moving objects are detected. Each technique will be applied to a series of videos, and the results of the detections analysed. The analysis is both objective and subjective - the objective results are the number of objects detected. Subjectivity comes in when determining what a detection is; does it represent an object or is it a representation of noise. Measures of accuracy against processing speed will be made such that an appreciation of an algorithms real-time capability can be made. The plan of the research, once this assessment is made, is to draw parallels with the way human eyes work. This is because human eyes work in real-time and are excellent at detecting and discriminating between static and moving objects. By drawing parallels, the adaptation or development of a new novel approach to object detection should be possible. The new approach that is developed will be compared and contrasted with a wide ranging set of existing methods which operate in single frame analysis or video stream analysis. The reason for comparing the both single frame analysis and video stream analysis methods is the detection capability is generally better in a single image detection method where as the video stream analysis maintains a temporal (and therefore) motion component. The research aim, as stated in the hypotheses, is to achieve similar or better performance in terms of single frame detection whilst maintaining the characteristics of a temporal video stream in order to establish object motion. The results will be analysed against a somewhat subjective outcome. Each test image or video sequence will have a ground truth established by a human (how many objects can the human detect). The number of detected objects by the algorithm compared to the ground truth will provide a measure of the performance of the algorithm in terms of detection. In addition to this, the real-time performance of the algorithms will be measured by calculating frame rate. For a single image this is the time taken to process a single frame. The results from this testing will be collated together to provide an overall assessment of how well the developed approaches work compared with the existing approaches.

Given the constraints and objectives described in chapter 1, the simplest place to start working with a method is background subtraction because of its well established methodology of detecting object pixels in a static camera scenario. In chapter 2, KDE, GMM and RDE were introduced. The calculation of exponential functions is computationally expensive, more powerful machines are able to mask this expense, but in low power boards it can be a problem. Recursive Density Estimation does not use a Gaussian kernel, or any exponential in its calculations which therefore gives it a computational advantage and may help with meeting our computational efficiency constraints.

3.2 Experiments with Recursive Density Estimation

The concept, originally introduced by Angelov [5], is a data driven approach based on a Cauchy kernel to find the relative density of data points within some data space, in a recursive, and on-line manner. The output of the technique is the density of the current sample relative to all the samples that have come before. It uses a statistically not empirically derived threshold which is designed to exclude noise. The result outputs are the most eccentric pixels in the video stream. The threshold is an assumption that some of the pixel density data is noisy or invalid - typically anything outside 3-sigma [116]. One of the cornerstones of this research is to reduce the assumptions made by novelty detection algorithms so that decisions on discarding information can be made by the semantic reasoning component. The first experiment explores removing the threshold, with the purpose being to address the assumptions made by this algorithm.

3.3 RDE Greyscale

Removing the threshold reveals pixels that have a density (detected) but are artificially hidden when the density threshold is in place. A side effect of this is that any pixels of no interest or noisy, which have some movement within the scene, will also become visible as detections. The results of the greyscale output is compared with a number of scenes. The use of multiple scenes enables the comparison between the thresholded and non-thresholded RDE in environments of varying complexity.

The Video Sequences

Simple rotating rectangles

This video sequence is a simple, artificial sequence of two counter rotating rectangles inside each other. It has been selected because it is a simple sequence of multiple moving objects moving differently to one another, and the objects only move in one plane of motion (in this case rotational). The sequence is named "TwoRectangle.avi" on the accompanying data device.

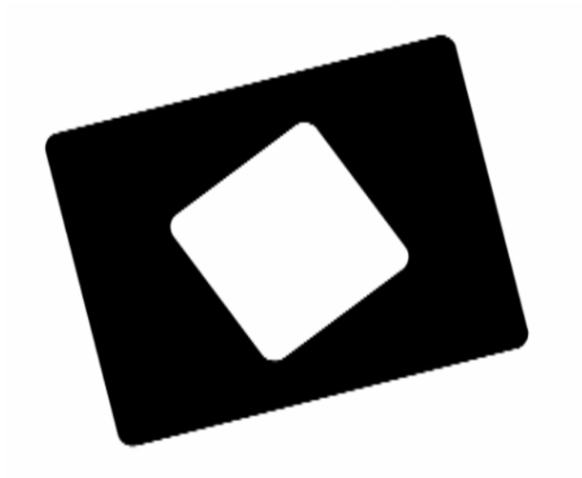


Fig. 3.1 Simple rotating rectangles

Vehicle accident scene

This video sequence is a real world scene of a car accident. It has been selected because there are a limited number of moving items in the scene, such that the complexity is maintained at a low level. Also, the aftermath of the impacts yield minor moving bits that may be considered as noise with the thresholded method. The sequence is named "TrafficCam.wmv" on the accompanying data device.

Fig. 3.2 Vehicle accident scene

Busy Walkway

This video sequence is a busy scene of people moving in multiple directions on a walkway. It has been selected as it increases the complexity of motion in the scene. There are also some smaller movements in the scene such as a person opening a car boot and cordon tape blowing in the wind. The smaller movements may be considered as noise with the thresholded method, and the complexity of the scene may effect the non-thresholded method. The sequence is named "768x520.avi" on the accompanying data device.

Fig. 3.3 Busy people scene

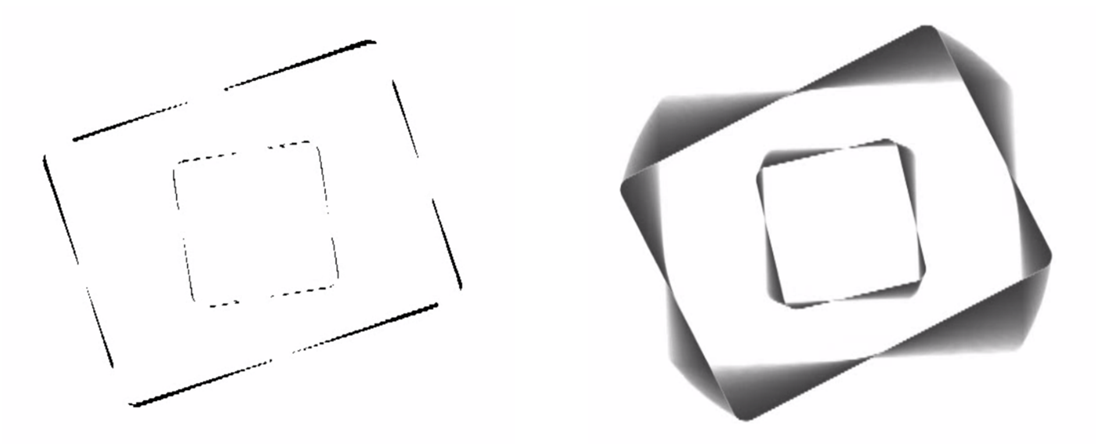
Z-axis Perspective Occlusion

This video sequence is a scene of a path in the background with people on it, and a car on a road passing in front of the path. It has been selected to test both methods in z-axis occlusion scenarios (objects passing in front of each other). Occlusion scenario testing is important because it is desirable to maintain novelty detections of objects after the occluding object has passed. The sequence is named "SOvid.mp4" on the accompanying data device.

Fig. 3.4 Z-axis perspective occlusion

3.3.1 Results of Experiments with RDE and Greyscale

The results shown here are from applying RDE, and the non-thresholded RDE greyscale to each of the video sequences from above.



(a) RDE (b) Greyscale

Fig. 3.5 A comparison of RDE and Greyscale applied to rotating rectangles

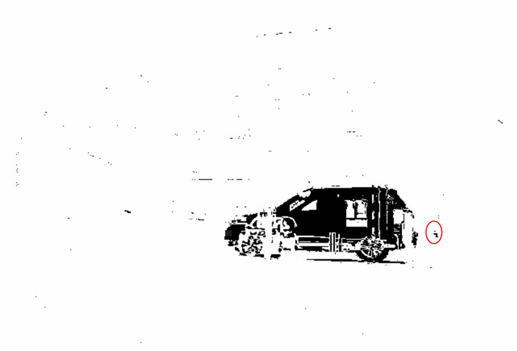
(a) RDE (b) Greyscale

Fig. 3.6 A comparison of RDE and Greyscale applied to a traffic accident

(a) RDE (b) Greyscale

Fig. 3.7 A comparison of RDE and Greyscale applied to a busy walkway

(a) RDE result before occlusion



(b) RDE result after occlusion

(c) Greyscale result before occlusion

(d) Greyscale result after occlusion

Fig. 3.8 A comparison of RDE and Greyscale results before and after an occlusion event. The red circle indicates the detection of the person with a white jersey walking down the path, see figure 3.4

3.3.2 Exploring the Results of RDE and Greyscale

In figure 3.5, the RDE detection of the two rotating rectangles misses some of the edges, where as the greyscale method defines all of the rectangle edges. The missed edges in the RDE experiment is because the visible detections are showing the density change (change in colour) of only that which is above the 3-sigma threshold. Any smaller density changes that occur will be below 3-sigma, and thus not shown as a detection. As an object of different colour to a pixel passes through the pixel location, the density of colour of the pixel changes. After the object has passed, the colour of the pixel returns to its original background colour. The colour density thus gradually returns to near its original value (the colour change from the passing object has less and less effect as more samples of the background arrive). If another object of differing colour passes the pixel, this will cause another colour density change. However, the density change will be less immediate and may not change sufficiently to break the 3-sigma threshold again because the density is already different to when it started. Despite another object passing the pixel, the detection has been lost in the averaging effect of the recursive algorithm. This effect is visible on the rectangles of figure 3.5. The edge parts that are missing, are pixels that are being passed by other pixels of the black rectangle (or white in the case of the centre one) as it rotates. The density does not change sufficiently to break the 3-sigma threshold and show as a detection. The contrails surrounding the greyscale experiment show the effect of the lingering density averages as pixels create an initial detection and then gradually average back to their original value. The dark to light grey pattern shows this; the darkest part is the leading edge of the rotational movement (the black edges of the rectangle are moving into white space), and the light grey indicates where the rectangle was when it first started moving.

The historical trails are clearer in the next set of images. In figure 3.6. The greyscale shows the historical path of the lorry that is hitting the car. The RDE image only shows the pixels of objects that are moving at that moment. The other pixels, are averaging back to their original density values, and are not breaking the 3-sigma threshold of density change. By removing the threshold, the decay of densities is visible (figure 3.6b). As a result, the lorry and car vehicles are clearer for longer and the passing car in the bottom left can be seen more clearly.

In the busy people scenario 3.7 the RDE result shows all or part of the moving people and the car boot opening in the top right. The moving people suffer from the same problem of density decay and thus are not clearly defined in some cases (their whole bodies do not break the 3-sigma threshold). There are pixels that are also considered noise in the RDE result. By removing the threshold, the noise turns out to be a part of a moving cordon tape in the background of the scene. The greyscale in a busy people scene, figure 3.7, can cause undesirable effects as well. The people are defined clearly, but the trails leave grey patches around the image, with no obvious pattern or trail to a particular object or person. This is because many of the trails from the people moving in the scene overlap where more than one person has crossed a point. Whilst the trails are useful in less congested scenarios (such as figure 3.6), in this scenario the trails introduce a significant amount of noise.

This noise is also apparent in the occlusion scenario, figure 3.8. Because the camera that took this video sequence was not perfectly still, the slight vibrations show detections of static objects such as the window frames because of the relative movement between frames. In the RDE result, the majority of these extra detections are removed by the the 3-sigma threshold. The greyscale also has a negative effect on the occlusion scenario (before and after images shown). In the RDE case the detection of the person walking down the path, whilst small, remains consistent before and after the car goes passed. In the greyscale case, the detection of the person before the occlusion event is slightly stronger than with RDE. However, after the vehicle goes passed, the greyscale trail left by the vehicle overlaps with the detection of the person, thus diminishing the definition of the person detection after the occlusion event.

3.3.3 Discussion of the RDE and Greyscale methods

In some cases the application of a threshold has assisted with removal of genuine noise, and has cleared up some detections; figures 3.8, 3.7. In contrast, there are places where the threshold does not detect enough of a moving object, or assumes noise where there is an object moving; figures 3.5, 3.2. In these scenarios greyscale was useful to define better detections and illustrate the historical path of object motion. The conclusion is that the removal of the threshold helps with the base detection of objects in a scene by not assuming that detections are noise, however the trails of historical motion interfere too much in busy or occlusion scenes. The trails interfere because it is a historical representation of all the density samples of a pixel since the algorithm started to run. In another background subtraction technique, KDE (see chapter 2), a window of samples is used and this is adapted based on the speed of the motion in the scene - too large a window in a fast moving scene can lead to detection overlaps. As the problem with greyscale is that the trails overlap in busy or occluded scenarios, a windowed approach to this will give the good effects of greyscale without the problem of overlapping object trails.

3.4 Windowed Density Estimation

The historical density is caused by the continuous recursive nature of the RDE algorithm, it retains an ever diminishing weight of previous samples. In some scenarios, the additional information of the history has been demonstrated to be of some use, and as such the aim is not to get rid of them completely. This windowed estimation approach applies an modification to the recursive update of the mean and scalar product equations. For reference, the recursive update equations from Chapter 2 have been repeated in (3.1) and (3.2) [5].

$$\mu_t = \frac{t-1}{t}\mu_{t-1} + \frac{1}{t}x_t \quad \mu_1 = x_1 \tag{3.1}$$

$$\Sigma_t = \frac{t-1}{t}\Sigma_{t-1} + \frac{1}{t}\,||x_t||^2 \quad \Sigma_1 = ||x_1||^2 \tag{3.2}$$

where t is the number of samples, μ is a vector mean, x is a pixel vector and Σ is the scalar-product.

To update the mean on a sliding window basis, the mean must be taken for the last n samples, where n is the window size. The historical mean is represented by μ_{t-1}, the sample from n frames prior is x_{t-n} and the current sample is x_t such that:

$$\mu_t = \mu_{t-1} + x_t - x_{t-n} \quad \mu_1 = x_1 \tag{3.3}$$

μ_{t-1} represents the mean of n prior samples, therefore scaling is required such that the new sample does not weight the mean towards the new sample:

$$\mu_t = \mu_{t-1} + \frac{x_t - x_{t-n}}{n} \quad \mu_1 = x_1 \tag{3.4}$$

Similarly applied to the scalar product recursive update equation:

$$X_t = X_{t-1} + ||x_t||^2 - ||x_{t-n}||^2 \tag{3.5}$$

and scaling with respect to n:

$$X_t = X_{t-1} + \frac{||x_t||^2 - ||x_{t-n}||^2}{n} \tag{3.6}$$

The experiments conducted with the sliding Windowed Density Estimation (WiDE) use a window size of two and a window size of five. The minimum selection possible is two, because there must be at least two frames to obtain a comparison. The detections from a two frame comparison is expected to yield the leading edge of the detections only with the trail indicating the position of the moving object in the previous frame. A window size of three or four is not expected to show enough detail of the trail to be compared with a window size of two. A larger window size than five is expected to be closer to the original greyscale representation, causing some trail overlap in the more complex scenes. Thus a window size of five is selected for the second experiment.

3.4.1 Results of Experiments with WIDE

The results shown here are from applying WiDE to each of the video sequences from above, with a window size of 2.

(a) Window Size 2 (b) Window Size 5

Fig. 3.9 Windowed density estimation applied to the two counter-rotating rectangles

(a) Window Size 2 (b) Window Size 5

Fig. 3.10 Windowed density estimation applied to the busy walkway.

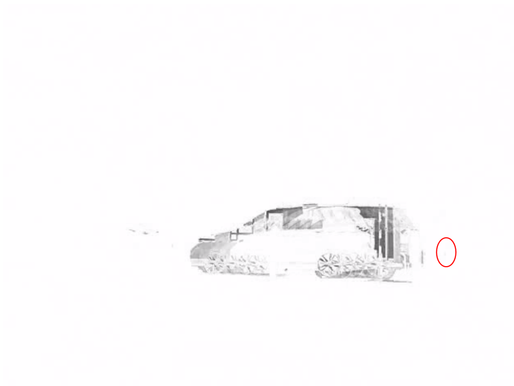
(a) Window Size 2 (b) Window Size 5

Fig. 3.11 Windowed density estimation applied to the road traffic accident scene.

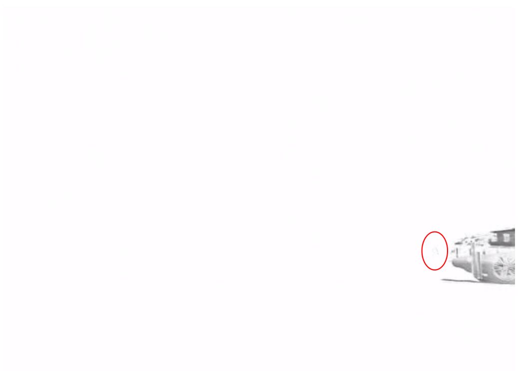
(a) Window Size: 2, before occlusion

(b) Window Size: 2, after occlusion

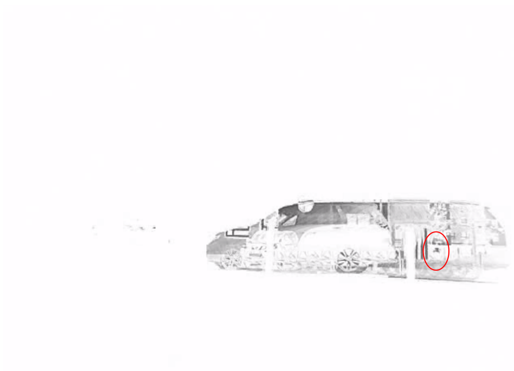
(c) Window Size: 5, before occlusion



(d) Window Size: 5, after occlusion



(e) Window Size: 10, before occlusion



(f) Window Size: 10, after occlusion

Fig. 3.12 Windowed density estimation applied to the occlusion scenario. The images show before and after the occlusion

3.4.2 Analysing the WIDE Experiments

The application of the WiDE technique to the rotating rectangle sequence, figure 3.9 shows that a window size of two suffers from the same missing edges. The central rotating rectangle is barely visible. With the window size set at five, the outline of the outer rectangle is complete, and has some historical trails. The internal rectangle still has gaps in the perimeter. At the fulcrum of the rotation, the pixels of the rectangle do not move, or move very slightly - sometimes not a whole pixel every frame. With a two frame analysis window, there is no motion detected near to the fulcrums because of this lack of motion over two frames and therefore a density change at these points is not detected. With a window size of five, there is sufficient motion over the five frames such that the pixels forming the outer rectangle move sufficiently to have a density change. The central rectangle gaps means there is not sufficient motion of this rectangle over a five frame window to show density changes near the fulcrums of rotation. The results from the window size of two are similar to the result of thresholded RDE for the outer rectangle. The thresholded method detected more of the central rectangle perimeter. The window size of five wide has improved detections from both the thresholded and window size of two, completing the external boundary of the rectangle with some history and mostly detecting the internal rectangle. The greyscale method performs better by completing the perimeter of both rectangles. The greyscale method has an additional trail apparent around the perimeter which does not impart any additional information because of the trail overlap (it is noise).

WIDE with a window size of two has a similar result to the RDE experiment in the accident scenario, figure 3.11. The immediate motion of the vehicles is detected, but any parts of the vehicles that have stopped moving are no longer detected. If there is no motion of a pixel within two frames the density of the pixel will remain at zero (unchanged) over two frames. At the window size of five, the outline of the vehicles becomes clearer and some motion history of the lorry and the car in the bottom left is shown with the trails. The greyscale method has a longer history of motion, and the definition of each vehicle is clearer without windowing the density output. Because this scenario is a quiet scene with not much overlap in motion, there is no issue with greyscale historical trails overlapping.

In the walkway scene with busy people, figure 3.10, the window size of two completely removes the greyscale trails seen in figure 3.7. The outline of each moving person is detected and the car boot opening in the top right is partially detected. The partial detection is because not all parts are moving over a two frame period. The cordon tape moving in the wind is not detected at all. There is a missed detection, but it is not a partial detection which appears as noise as with the thresholded RDE approach. The window size of five successfully detects the cordon movement without it appearing as noise, along with the people and car boot movements being detected. The additional information of motion history is detected as trails behind the people moving. The window size of five means the trails are short enough so they do not overlap with other people as with the greyscale method, figure 3.7b.

The windowed approach does not detect small moving objects as successfully in the occlusion scenario, figure 3.12. With a window size of two, the car is detected but the person walking down the path does not have sufficient movement between pixels to be detected over two frames. The window size of two does however attenuate the noise of the background caused by the camera shake. The window size of five frames also does not detect the motion of the person before the occlusion event of the car as there is insufficient motion over five frames for the detection (a combination of the object being small, and lack of motion over five frames). After the car has gone through, the persons movement is sufficient such that a faint detection is achieved. In this set of experiments the window size was increased again so that a detection can be made with the windowed approach before the arrival of the car. Through empirical experimentation, a window size of ten was found to be the minimum required to detect the person before the car arrives. With this size of window, after the car occlusion, the person detection is mixed in with the trails associated with the car movement. Thresholded RDE and greyscale both achieve better detection resolution of the person and car than any of the windowed methods.

3.4.3 Appraising the WIDE Method

The results from the WIDE experimentation are mixed. In the busy scenario of moving people, the detection performance of the greyscale method was achieved at a window size of 5, without the noise created from the continuous historical trails creating noise when they overlapped; historical movement can be extracted from the shorter trails. The poor performance of WIDE with the occlusion scenario can be explained with the geometry of the movements in the scene. The person is coming towards the camera at a slow rate, with little lateral movement. From the perspective of the camera, the only motion detected will be when the person moves closer to the camera in the z-axis. The motion detected is the person getting larger with respect to the camera perspective. Over a small window size such as two or five, the enlargement through perspective is not sufficient to appear as motion on pixels (the enlargement may be in subpixel scale), and therefore density detections are not made. The threshold and greyscale methods work because the pixel density is taken over a large number of frames, and the enlargement motion over a number of frames is sufficient to show a density change over the pixels. The geometry problem in the z-axis is a well known problem that occurs in computer vision [122]. WIDE discriminates noise, but at the cost of missing other desirable detections. Each of the four techniques that have been analysed over the last two sections have positives in some scenarios and drawbacks in others. The most consistent is the greyscale technique, but it introduced noise into the walkway scenario, as well as interfering with the person in the occlusion scenario. For a static camera environment, the use of each technique is scenario specific.

The experiments in sections 3.3 and 3.4 are applicable to the static camera environment, and there are many varying background subtraction techniques available for this application some of which are discussed in chapter 2. The occlusion scenario, figures 3.4, 3.8, 3.12, had noise introduced by the camera shake (the platform was moving, such that the background appears as a detection with the background subtraction techniques). The objective to use detection algorithms on UAV will also encounter the movement problem, as the aircraft is not still and will be traversing across terrain. This leads to a more complex scenario to consider; detecting objects in a moving camera. Background subtraction will detect the background as moving well as objects in the foreground. The next set of experiments explores the technique of motion estimation, which is a technique of warping consecutive frames into the same perspective and stitching them together so that background subtraction can be performed on the static overlapping areas of the frames.

3.5 Motion Estimation Accuracy

As described in Chapter 2 motion estimation warps two or more consecutive frames into the same perspective, and stitches them together to create a static overlapping region. Autonomous Real-Time Object Detection (ARTOD) is a method that extends the motion estimation to include background subtraction [126], applied to the static region. This approach uses the Recursive Density Estimation (RDE) background subtraction method [5]. Motion estimation introduces artificial noise at the stitching boundary; at the 3-sigma threshold, RDE excludes most of the artificial noise from its detections. The drawback of having a threshold (as seen in 3.3) is that pixels that make up an object of interest can be suppressed, reducing the clarity of the object detections. RDE increases the overall frame processing time by a margin of between 20 – 50 ms per frame, depending on the processing cores and image dimensions used. The motion estimation components use the majority of the processing time, typically each component takes betweeen 50 - 100ms per frame. The objective of the experiments in this section is to explore the accuracy and computational performance of each component, and to identify any trade-offs. The artificial noise introduced by motion estimation is due to the discrete nature of pixels and the double-precision result usually associated with geometric calculations [1]. The sub-pixel localisation of warping of the frames causes mismatches in alignment, introducing the noise. Further artificial errors are introduced by the alignment function not being precise enough - even without the sub-pixel problem, pixel localisation can be inaccurate. This noise can be minimised by optimising the alignment function, which is done by optimising key point localisation and matching. Adding in extra optimisation methods increases the processing resource requirements; increasing processing time. The processing speed can be optimised by minimising the complexity or the number of key point localisation and matching processes. To explore the optimisation characteristics of motion estimation, experimentation was conducted on the following components of motion estimation:

- Key point Detection
- Key point Matching
- Key point Filtering
- Homography (affine transform)

Both the RANSAC method for selecting keypoints for the homography matrix, and the homography generation are based on sound mathematical principles [36] [41]. The components also contribute the least in terms of processing time consumed. At this point, it was decided not to experiment with modifying these components, as the above list of components have a greater impact on both processing time and matrix accuracy.

Key Point Detection

The accuracy of the homography matrix (the matrix used to warp a frame into the perspective of a reference frame), is determined by the accuracy and validity of the key points that are detected and the matching algorithm used to associate key points between two frames. In the work by Sadeghi-Tehran and Angelov [126], the SIFT algorithm from [78] is used as the keypoint detector. This experimentation will explore the use of different octave values with SIFT ([126] does not specify the octaves used for the experimentation). Additionally, different keypoint recognition algorithms will be explored. The four keypoint methods experimented with here are SIFT [78], SURF [14], BRISK [74], and ORB [2]. These key point methods are used because the development is chronologically progressive, and the code to run these algorithms is readily available in OpenCV (the Application Platform Interface

(API) that is used by this project). The OpenCV implementation of each should provide a consistent code base so the code implementation doesn't artificially affect the running time.

Key Point Matching

The speed of the matching process is slower if the video stream represent an environment that produces a large number of key points. Utilising a brute force matching approach leads to a fast matching result but may require greater filtering post-matching. The brute force matcher takes a sample from the first frame and it is matched with all other samples in second set using some distance calculation (typically Euclidean), the closest match (shortest distance measure) is returned. A number of different feature comparartors could be used to conduct keypoint matching [73, 77, 89, 2], however the Functional Link Artificial Neural Network (FLANN) is readily available in the software API being used and will form a consistent code set to the experiment. The objective here is to experiment with the effect of matching algorithms on speed and accuracy of the final stitching process, not appraise the matching algorithms themselves. The FLANN based method [91], uses a feature classifier to match the keypoints and is a single layer feed forward neural network.

Key Point Filtering

The output from the matching process can produce tens or hundreds of matches, some of which are outlier matches; they are not close in distance, but are matched because they are the closest match from the available keypoints. This experiment looks at the effect of keypoint match filtering, using two different types of match filter and how they effect the end result accuracy. The filtering process removes matches that are outliers based on some distance measure. One filter is a simple match filter that uses a distance threshold such that the distance measure of a match must be below this threshold. Anything outside the threshold is rejected and the match is discarded. This can be useful in scenarios where the motion of the scene between frames is known or is constrained such that a threshold does not exclude valid matches. A second filter, when the motion is unknown, calculates cross matches of keypoints. The matches from frame 1 to 2 are calculated, and then the matches from frame 2 to 1 are calculated. Only the keypoint matches that agree in both cases are retained as keypoints, with the remainder discarded. If there are only a few keypoint detections in a scene, this method can lead to over-filtering such that there are not enough keypoints remaining to construct the homography matrix.

Homography

The sub-pixel alignment problem introduces artificial noise. This experiment considers interpolation as a method to optimise the alignment to minimise the artificial noise created by the stitching process. Simple nearest value interpolation is used in the ARTOD proposal, which is insufficient to avoid misalignments of pixels which result in false detections around the edges of objects in video sequences moving in more than one plane. Utilising bi-cubic interpolation [64] (because the warped frame does not align to discrete pixel values) during the stitching of frames could improve the alignment in sequences with more than one plane of motion. This method has been selected because it is efficient on modern hardware that could improve accuracy whilst being unlikely to introduce a large performance penalty.

The Video Sequences

Helicopter chase

This is a video sequence where the camera is moving in one plane of motion, translational, following a motorbike and a car. This sequence has been selected because translational motion is less prone to noise on stitching and there is a limited complexity to the moving objects (two objects, in mostly a straight line).

Fig. 3.13 Helicopter chase scene with a motorbike and car

Street panning

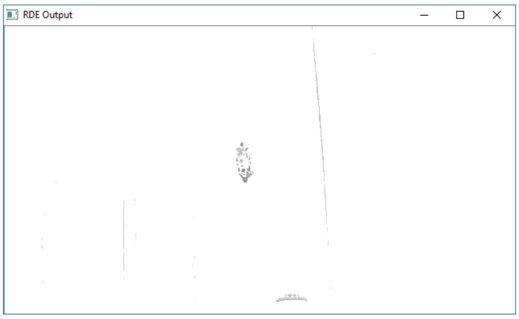
This is a video sequence of a fixed camera moving in a rotational axis about the y-axis. This sequence has been selected because stitching of rotational motion is more prone to artificial errors than translational movement (because a 3-D component must be taken into account). The beginning of the sequence has no moving objects, so misalignments and noise can be seen clearer. Later in the sequence there are three moving cars which tests the noise performance when motion is introduced.

Fig. 3.14 Panning motion of a street scene

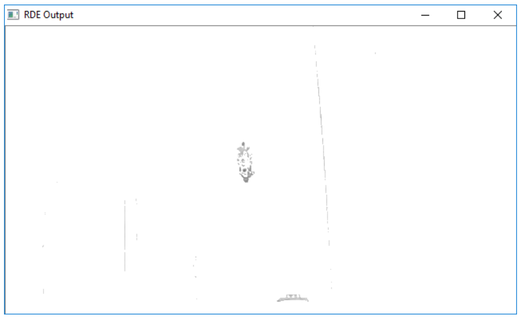
3.6 Results of Motion Estimation Experiments

In this section single frames of the video sequences are shown with the various stages of motion estimation applied. Each stage has different methods applied and the results displayed illustrate the differential between each method.

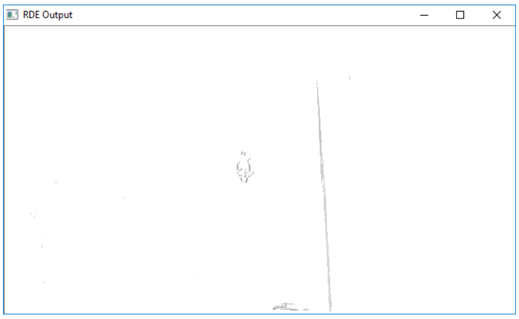
3.6.1 Keypoint detection



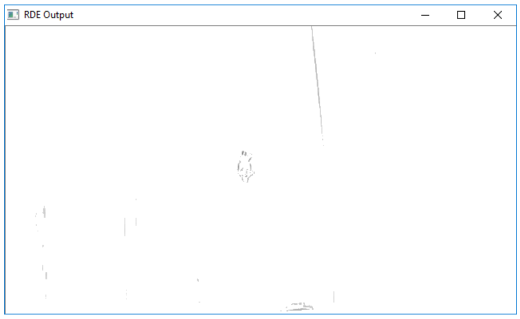
(a) SIFT, Helicopter



(b) SURF, Helicopter



(c) BRISK, Helicopter



(d) ORB, Helicopter

(e) SIFT, Panning

(f) SURF, Panning

(g) BRISK, Panning

(h) ORB, Panning

Fig. 3.15 Different key point detection algorithms used for motion estimation on the Helicopter and Panning videos

(a) SIFT, Z-axis

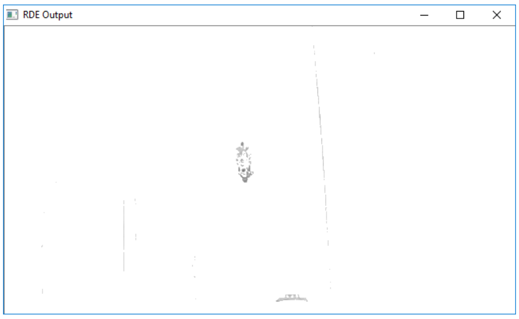
(b) SURF, Z-axis

(c) BRISK, Z-axis

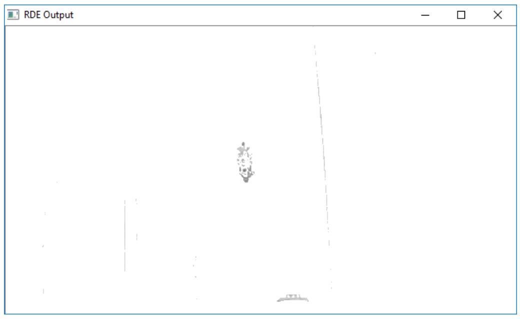
(d) ORB, Z-axis

Fig. 3.16 Different key point detection algorithms used for motion estimation on the Z-axis video

3.6.2 Key point matching



(a) Flann, SURF, Helicopter video



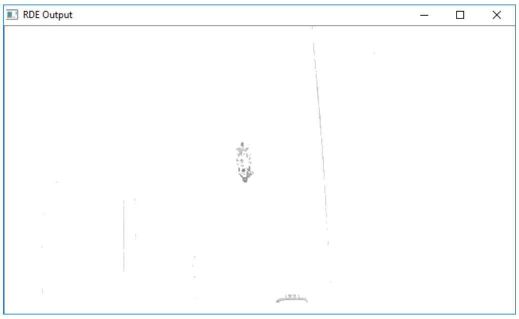
(b) Brute Force, SURF, Helicopter video

(c) Flann, SURF, Panning

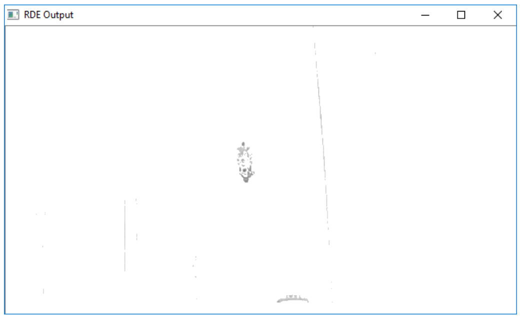
(d) Brute Force, SURF, Panning

Fig. 3.17 Key point matching algorithms used for motion estimation on the Helicopter and Panning videos

3.6.3 Key point filtering



(a) KNN Filter, BF, SURF, Helicopter



(b) Radius Filter, BF, SURF, Helicopter

(c) KNN Filter, BF, SURF, Panning

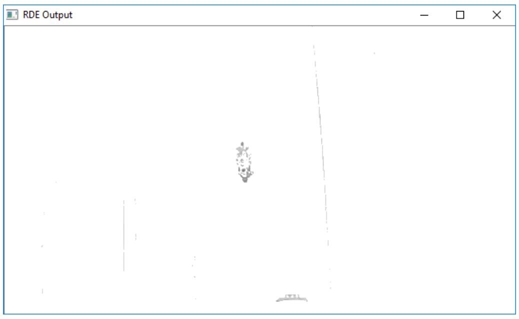
(d) Radius Filter, BF, SURF, Panning

(e) Outlier Filter, BF, SURF, Helicopter

(f) Outlier Filter, BF, SURF, Panning

Fig. 3.18 Filtering algorithms used for motion estimation on the Helicopter and Panning videos

3.6.4 Homography Interpolation



(a) Linear Interpolation, BF, No filter, SURF, Helicopter

(b) Cubic Interpolation, BF, No filter, SURF, Helicopter

(c) Linear Interpolation, BF, No filter, SURF, Panning

(d) Cubic Interpolation, BF, No filter, SURF, Panning

(e) Linear Interpolation, BF, No filter, SURF, Extended panning

(f) Cubic Interpolation, BF, No filter, SURF, Extended panning

Fig. 3.19 Interpolation algorithms used for motion estimation on the Helicopter and Panning videos

3.7 Analysing Motion Estimation Experiments

3.7.1 Keypoint detection

The results shown in figures 3.15 and 3.16 are of different key point detection algorithms applied across three separate scenes. The objective of this experimentation was to explore the effectiveness of motion estimation, and the variation of results from the algorithms. Each frame was selected based on the difficulties motion estimation had at eliminating noise. In each video sequence there are several frames where the image stitching is good enough so that no extraneous noise is found. However, in a comparison scenario, it was desirable to have frames where all permutations of the algorithm exhibit noise to an extent. The helicopter frame provides for some disparity between the methods. Both SIFT and BRISK exhibit the noise of the road markings on the right clearer than SURF and ORB. BRISK eliminates the road markings to the left completely, as does ORB. Whilst the SURF algorithm detects both verge lines, it has a fainter detection than all others for this line (a fainter detection means a smaller shift in alignment). ORB is the most noise free in this scenario. The panning scenario, which should not have any detections (no moving objects) is fairly consistent across all four techniques, each exhibiting small detections of background. SIFT and BRISK both detect some line noise in the bottom right of this frame. ORB has small detections in this region, and the SURF algorithm is the least noisy as there is no noise detected in the bottom right. The z-axis motion is the most noisy result, with each algorithm producing many detections. This is highlighting a weakness in the motion estimation approach; it is susceptible to noise and cannot detect moving objects when the camera motion is in scale space. Notice how the cars in the Z-axis frame do not register as even noise. In this scenario, both SURF and ORB are slightly better at discriminating background noise. In terms of performance, both BRISK and ORB perform the keypoint detection faster than SIFT or SURF, with SIFT being the slowest and ORB being the quickest. Despite SIFT being the slowest, SURF detects the most keypoints, followed by BRISK and then ORB. In terms of image size scaling, despite being the second fastest, BRISK scales the worst with the times increasing by order of magnitudes between video sequences. ORB maintains a log-linear scalability with image size.

3.7.2 Key point matching

Figure 3.17 shows the comparison of matching algorithms used in this experiment. For consistency, the SURF algorithm is used as a baseline key point detector. The algorithm is used because the output provides a large number of keypoints that lends itself to filtering the matches in the next process. If there are too few matches, a homography matrix cannot be generated. In pure matching, the FLANN algorithm provides the same number of matches as the brute force algorithm. It is consistent that with the same matches, the resultant output videos are the same. The computational performance of the brute force method is faster in both scenarios than FLANN, and scales better - only increasing by 11ms for a more complex frame compared to 17ms increase by FLANN.

3.7.3 Key point filtering

Figure 3.18 shows a comparison of filtering algorithms applied to the keypoint matches. The objective is to remove keypoint matches that are inaccurate matches and will skew the homography generation. SURF and the brute force matcher are used in these examples. The KNN and radius filters are both cross check filters, and the outlier filter is a simple distance measure filter. In the Helicopter video the radius filter has slightly better noise reduction than the KNN filter. The verge line on the right is less pronounced in KNN filtering compared with the radius filtering. In the panning scenario, there is little difference between both cross check filters. The outlier filter removes noise even further on the Helicopter video, with much reduced noise on the left compared with KNN or radius cross check filtering. The result suggests that the frames were warped together slightly differently because the verge line on the right is more pronounced towards the bottom of the frame compared with a higher up detection of the verge. The panning sequence is barely affected by the filtering. The computational performance of both cross check filters is comparable with each other with the radius filter being 4ms slower than KNN in the helicopter video. The filters are almost identical in terms of performance in the panning sequence. The outlier filter adds very little performance overhead to the matching process, and removes similar numbers of matches compared with KNN and radius matching.

3.7.4 Homography Interpolation

The interpolation results are shown in figure 3.19. In the simple translational movement of the helicopter video, the centroid interpolation errors are difficult to spot because of the single direction of scene movement, the cubic performance is fractionally better than the linear interpolation. In the video scene with the camera panning from a fixed spot, two tests were conducted. One with no moving objects and the second with an extreme panning motion. The linear interpolation appears to perform fractionally better in the helicopter scene than the cubic interpolation with less distortions, but in the panning scene the cubic interpolation performs the best. It is difficult to draw conclusions directly from these results and they do not show whether the interpolation method makes a significant impact or not. A possible reason for this is the homography matrix generated is not perfectly accurate in the first place.

3.7.5 Conclusions on the Motion Estimation Approach

With all the accuracy improvements in-place, the processing time of the approach increases by approximately 250ms for each frame. Over the experiments shown here, and others with different combinations of components, it is apparent that the motion estimation approach is reaching the limit of manipulation with the trade-off being between alignment accuracy and speed. A full table of the results for motion estimation experiments can be seen in Appendix A. The optimal solution is heavily dependent on the application of the technique. In simple scenarios, where there is only transitional movement, the filters and interpolation used to improve robustness not necessarily due to the limited affine transform required. In very stable scenarios the key point detection method changed to the faster but less accurate BRISK algorithm with little or no effect on the result accuracy providing a computational speed benefit. However, scenarios where there is significant movement beyond transitional (such as camera jitter, rotation, and panning), a reduced filtering set and faster but less accurate key point detection can lead to noisy results. In this case the improved accuracy methods should be used, at the cost of processing speed. However, the accuracy improvements seen over the experiments is quite small, despite a large offset in computational performance. In these experiments, the keypoint detection algorithm has been shown to be the largest factor in accuracy or speed, with the filtering, matching and interpolation contributing minor improvements.

3.8 Hierarchical Framework

The methods looked at so far use a hierarchical framework model such as motion estimation [126]. In computer vision, a traditional analysis approach is to use the hierarchical model; there is the initial detection of pixels of interest at the low-level, extending up to the high level semantic reasoning for behavioural or tracking analysis (which uses features derived from lower levels). Figure 3.20 shows an example hierarchical model that can be typically seen in computer vision systems [15] [47] [129] [59].

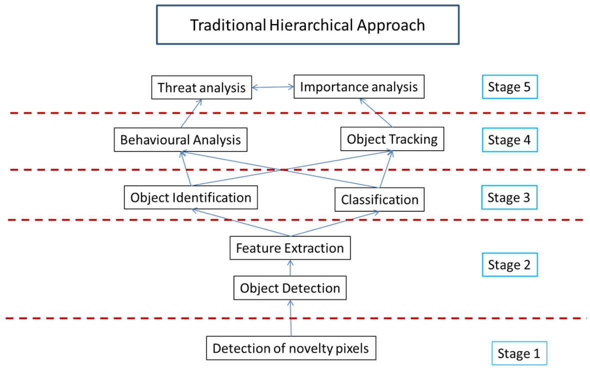


Fig. 3.20 Computer Vision hierarchical model

The hierarchical model is also common amongst other computer science applications. Examples of uses in computer science; the networking OSI model 3.21, operating system kernels 3.22, and computer game design 3.23.

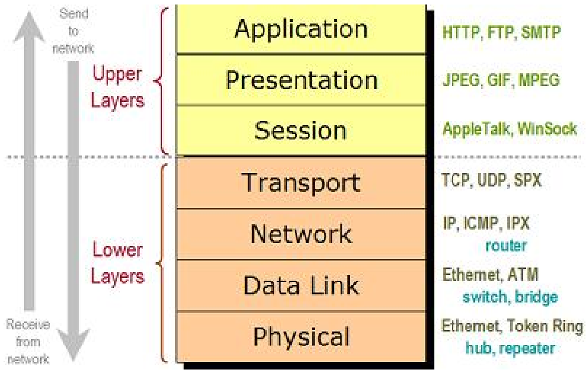


Fig. 3.21 OSI network model

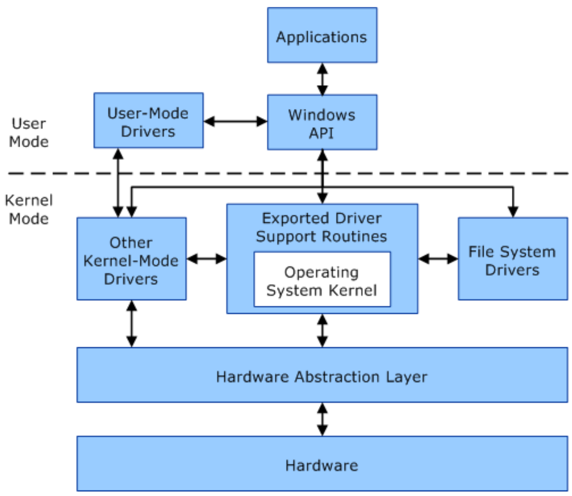


Fig. 3.22 Typical operating system kernel hierarchy

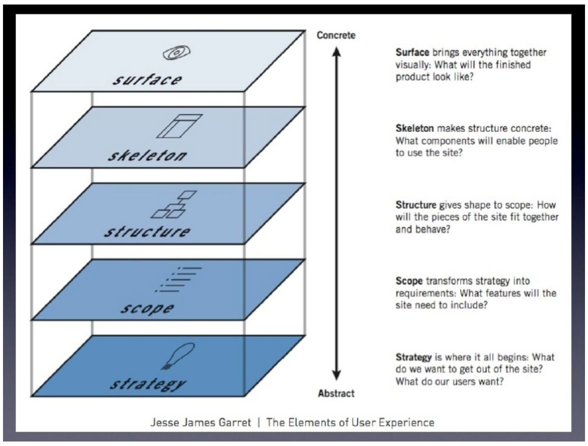


Fig. 3.23 Example hierarchy for game design

3.8.1 What are the limitations?

Despite being a successful model in these applications, the hierarchical model has limitations. Each component interacts with the components directly above or below them, they have no connection to any other components. The lack of interaction between other components means that components at different levels of the hierarchy have no influence over the data input into other components. If there is a noise component in the low-level detection phase, either the noise will be present in the input data to the next component, or there will have been a noise filtering process. The next component may also introduce noise of its own, and possibly accentuate existing noise in the data passed to it. At the top level of semantic reasoning components may have incorrect information based on the accumulation of errors and noise. A problem with using a hierarchical analysis models is that the scope gets smaller and is more constraining. Higher level analysis can only use the data provided by the previous levels, with no influence on the data that is gathered at levels below. If a component discovers information that may be useful to the components below, there is no method to pass this information to these components. The outcome of this is that the number of objects, behaviours or tracks can only be equal, or less than the number of data samples detected at the lowest level.

3.8.2 Developing the framework

The computer vision framework is as important as the algorithms used within it. It is the framework that decides what organises the types of operations, and how each operation or task might interact at each stage. A framework that is not susceptible to the limitations and problems of hierarchical models could lead to better overall performance of computer vision systems. Software development is a good example of a field in computer science that uses other types of frameworks; the V model where the top components of each side of the model have an interaction as well as with the layers below them [92], seen in figure 3.24; "Agile Development" [32] which is a cyclic approach to the problem as seen in figure 3.25.

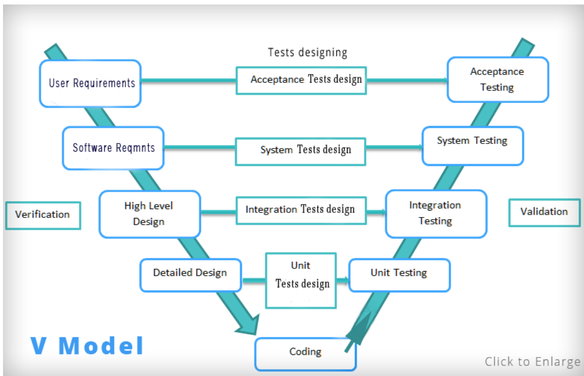


Fig. 3.24 V-Model for software development

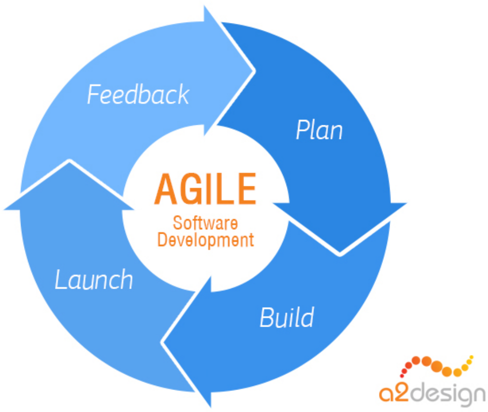


Fig. 3.25 AGILE model for software development

Using these models to influence the development of a new framework has led to the development of the cyclic framework for computer vision 3.26

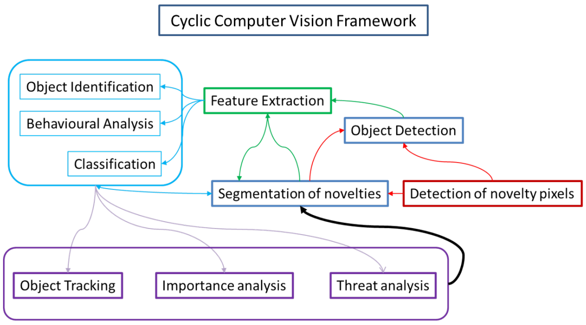


Fig. 3.26 A cyclic framework proposed for computer vision

This model allows "higher level" analysis to feedback information to "lower level" components to help optimise the information gathering. This could be providing localisation information on objects of interest meaning the detection phase focuses on only a particular region of the image. This has the potential to lead to improved performance, and lends itself to self-learning techniques and evolving analysis (such as autonomous parameter selection).

3.9 Understanding the Limitations

Before a method can be used to solve the problems set out in this project there is a requirement to understand the limitations of the current approaches. This section summarises limitations of the techniques that have been looked at (both in review and experimentally). Each of the techniques analyse each pixel several times to obtain an assessment of its novelty. The novelty is assigned based on pixels that change significantly in the scene (usually moving objects) in some feature space (e.g. gradient, colour, brightness). Having to analyse a pixel several times to obtain a measure of its novelty repeatedly uses processing resources, slowing the efficiency of the method. The performance of the algorithms is also proportional to the frame size as a result. In the case of optical flow, the gradient analysis must process each pixel's value a minimum of 8 times for a 3x3 analysis grid (usually it is larger) and therefore only achieves frame rates of around 2 or 3 seconds per frame. Motion estimation must analyse the scene for key points, filter the key points and match them between frames. The homography matrix is then applied to every pixel in the scene, and in the case of ARTOD [126] further analysis is conducted on each pixel to determine its novelty using RDE. This is analysing the frame a minimum of 5 times (depending on how many key points were found and in the case of SURF, the hessian value used). A recent improvement [8] uses optical flow on the key points to determine their importance and has reduced the number of times the frame needs to be analysed to determine novelty pixels. However, given the optical flow processing time (despite being on a reduced number of points), the frame rates for processing are still only around 3 frames per second. Both motion estimation and optical flow techniques make the assumption of a novelty being a moving or dynamically changing object. Important objects within a scene may well be stationary for an extended period of time or the object of interest may be a static object. To enable a system to operate fully in an unknown environment (such as that a UAV operates in), there needs to be very little assumption about the environment being analysed. The assumption that an object must be dynamic to be a novelty can open the system up to exploitation, and can leave out important detections. This effect also manifests in moving camera environments when an object is moving precisely at the same velocity as the camera; the object appears to the algorithm as background and is not detected as foreground. The methods that are capable of detecting static objects are the edge detection and image segmentation methods. These have the limitation of only being spatial analysis, and do not calculate any motion perception. Coupled with the limitations highlighted with the hierarchical framework, there is scope to develop a technique that encapsulates the limitations described here.

3.10 Next Steps

A new idea based on the pixel vectors in optical flow is to initially use greyscale RDE on a dynamic video stream, which will produce edge detections on all objects within the frame. As explained in section 3.3 the greyscale RDE leaves a trail behind based on the previous detections. If the gradient (optical flow) from light to dark of the trails is obtained it is possible to assign a vector of motion to the pixels, where the darkest line of pixels is the leading edge of the object in the frame. By assigning a vector of motion to each pixel in the frame it will be possible to determine the typical vector of motion seen in the video stream. All pixels conforming to this typical vector can either be removed (so only dynamic objects are seen) or allocated a particular shade or property to isolate these pixels. Any pixels that are eccentric to the typical vector can be considered part of a foreground or dynamic object and each variation of motion vectors can be assigned a separate colour or property to differentiate them. The next chapter explores the concept further.

At this stage a precis of the objectives of the work can be made:

- Develop a novelty detection approach that avoids warping and stitching images together with an aim to increase robustness and speed of the approaches.
- Association of consecutive frames without the need to co-locate pixels from both frames

- Reliably detect relatively stationary objects of interest whilst maintaining background discrimination
- Ability to distinguish between a background object and background scenery

There is scope to explore a new concept to achieve the objectives set out above. There are some already existing approaches that contribute to the development of a new computer vision algorithm.

Chapter 4

A New Way of Thinking - Edge Flow and WISE

4.1 Human Vision - Models of Biederman and Wertheim

Humans can easily identify objects e.g. simply looking out of the window, or glancing around the office there are several objects; some inert, some in motion, and others partially occluded or unclear. The focus of this project is not how the biological system achieves the interpretations, rather what are the factors that allow visual differentiation of objects. What features define the coats on a peg are not part of the peg? Biederman [18] posed a similar problem and proposed a framework to describe the object detection process. In this framework the first suggested activity is edge extraction by reaction to colour and luminance changes in surfaces and textures; defining the boundaries of the textures. Biederman goes on to describe further analysis of the detection based on the properties of the edges. The framework diagram for this analysis is shown in figure 4.16. This is a simplified model and Wertheim [153] with commentary from Büttner and Straube [22] goes further and proposes a model that describes how we deal with motion perception in conjunction with object detection, and subsequently knowledge of object motion relative to us. Wertheim proposes that the processes of motion detection and object detection are separate parallel processes. In the work, the action of motion detection and perception appears to be reference surface based and independent of what the object is detected to be. Specifically with motion perception [80] postulate models that describe the way in which the motion extraction occurs. The ideas of Lu and Sperling [80] are not too dissimilar to the concepts of optical flow (first order brightness differentiation). This understanding of human vision has been used to inform the construction of a new approach which uses the edge contrasts within the optical field of a camera to define particular texture patches. Optical flow has been applied in a parallel paradigm to provide motion perception and understanding.

4.2 Edge Flow - A new concept in novelty detection

Looking out of a window people can see a myriad of various objects in the scene ranging from trees, to fields, to buildings and cars. An analyst looking at a video stream with a mix of moving and static objects would find it difficult to focus solely on an important object, static or otherwise. The method described here is a new data driven method to novelty detection and object definition in dynamic video streams that detects boundaries of **all** object textures, static or moving, and extracts detail about the internal structure of each texture patch. The thinking behind the proposed approach is not to make assumptions about the content of the video stream, and to model the human vision system proposed by Biederman [18] in an effort to emulate the model in computer vision. The approach also maintains the philosophy of model separation proposed by Wertheim [153] by conducting the detection and motion perception separately. The method described here, dubbed Edge Flow, detects texture patches in a scene and then uses optical flow to give motion perception to each texture patch. The WIDE method is used to detect texture patch edges, Sobel filtering is used to extract gradient of the edges, contiguous edge linking is used to define objects and optical flow is used to define the motion of the texture patches.

Figure 4.1 illustrates the components of the proposed method:

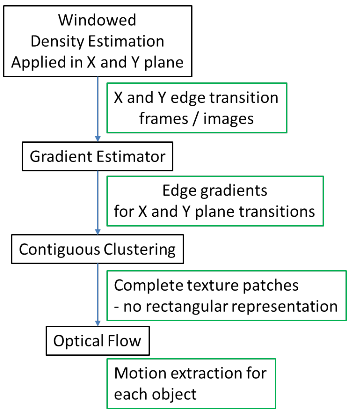


Fig. 4.1 Edge flow components, and in green, the output at each component stage

Fig. 4.2 Greyscale RDE applied to moving camera to define object texture edges

Fig. 4.3 WIDE applied to moving camera to define object texture edges

Windowed Density Estimation

The component utilises WIDE, described in chapter 3 however it is not used for the purposes of background subtraction. In WIDE, as with greyscale, density detections are seen across the entire scene, not just when the density reaches a threshold. The density of a pixel changes over time depending on the movement of colour textures over it. Each texture in the scene is referred to as a texture patch. The leading edge of the texture patches that are moving will yield a sharp density change, as with foreground moving objects in background subtraction. Similarly, the historical trails are seen in 3.4 are also present with reducing density change the more historical the texture patch movement is. The greyscale method, section 3.3, showed densities allowed to build up over an infinite number of frames. When the camera is moving as well as objects in the scene, this leads to the entire scene averaging out to a small range of densities that represent the colour patterns of the image (effectively making the output frames greyscale replicas of the input frames, figure 4.2). Using WIDE in this scenario is useful because historical contribution to the density is only over n frames (where n is the window size of WIDE). The effect of using WIDE with moving cameras is that each edge that is moving relative to the camera platform will be detected (static or otherwise) with a leading edge and a short historical trail. **All** the edges in the scene will be detected; the exception is texture patches that are in synchronous movement with the camera, there is no pixel density change relative to the camera.

Figure 4.3 shows the effect of applying WIDE to a moving camera scenario.

-1	0	+1
-2	0	+2
-1	0	+1

x filter

+1	+2	+1
0	0	0
-1	-2	-1

y filter

Fig. 4.4 X and Y Sobel filters

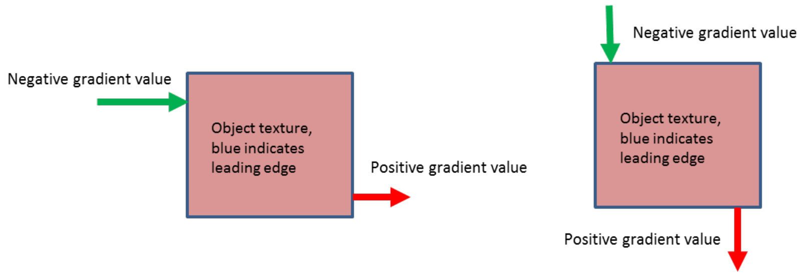


Fig. 4.5 Left, gradient values assigned for an x-plane Sobel filter. Right, gradient values assigned based on a y-plane Sobel filter

Gradient Estimator

A method to isolate the magnitudes of the gradients of each edge detected is required to characterise the edge profiles in order to provide internal texture information. To establish the gradients of each edge, a Sobel operator [135] is applied to the resultant edges in both the x and y planes. There are a number of mathematical operators and solutions available to calculate the gradient of the edges, including modern variants [141]. The Sobel operator has been selected because of its low computational complexity, and because it does not make assumptions or distribution predictions about the gradient profile. Figure 4.4 shows the Sobel operator used. The size of the Sobel operator (illustrated is 3x3) determines the local area over which the gradient is calculated. A small Sobel operator is more sensitive to large local changes in pixel value. A large Sobel operator (say, 7x7), on the other hand, smooths the gradient profile of large local changes. The size of operator is a parameter that allows the method to be tuned to application specific scenarios.

The Sobel filters are applied in both the x and y axis. If there is a positive gradient, this indicates a gradient going from low to high, and similarly a negative value indicates a gradient going from high to low. In the case of this application, because non-edges of object textures (background) are defined as white and edges of object detections are a grey value between the white and the black, positive gradients indicate transition from a leading edge of an object texture and negative gradient values indicate transition to a leading edge of an object texture.

Fig. 4.6 Result of a Sobel filter in the y-plane applied to RDE image. The gradient values have been coloured for a better visual effect – blue indicates large gradient changes, whilst green is a smaller gradient value. Yellow indicates areas of no edge gradients (and therefore no edges – the texture of an object).

The Sobel filters, in both cases, are applied from top-left to bottom right. Thus the first encountered edge of a texture patch will always be negative, and the final edge of the texture patch will be positive. The absolute value of the Sobel filter output indicates the magnitude of the gradient at the pixel. A large value (positive or negative) indicates a large gradient. The approach is novel through combining the first two components, and the processing speed surpasses any available algorithm for novelty detection in a moving camera scene. The processing of a 640x480 video frame is done at a real-time speed of 40 frames per second (25ms per frame) on an Intel i7 2.6 Ghz processor.

Contiguous Edge Linking

At this stage there are two outputs from the initial frame input – the x-plane and y-plane Sobel filter gradients for each of the edges in the scene. To extract the texture patches from the information derived, a linking method, dubbed Contiguous Edge Linking is used. Each contiguous pixel on an edge defined by a gradient is tested for neighbouring similarity. If a neighbouring edge pixel is within a specified tolerance, it is linked to the current pixel. The approach is made faster by avoiding processing every pixel within the frame; only the pixels that have a gradient assigned to them are considered (the body of a texture patch will not have a gradient – it is not an edge). The procedure for this edge linking method is as follows:

1. Working from the top left of each image (x-plane and y-plane Sobel images) find the first pixel with a non-zero gradient value.
2. Create a region of influence of 1 pixel either side of the candidate pixel. The region is the area the edge linking considers contiguous for new candidate pixels. The Sobel images are stacked so the area of influence also applies to the other Sobel image in a 3-D plane. This results in the combination of both images into one set of linked edges on the original frame.

3. Assess the surrounding pixels within the area of influence of the pixel (except for image edges) for pixels (or linked edges where this pixel overlaps an existing pixel area of influence) within the same gradient range. The gradient range is a pre-defined parameter on initialization (currently not autonomously defined).

   (a) If this pixel is within the gradient range of an edge pixel, and it is within its area of influence, add to the edge.

   (b) Otherwise, if not the first pixel being assessed, create a new edge, and if a neighbouring pixel is within the gradient range, and is contiguous (within the area of influence), add to the newly created edge.

   (c) If there are no neighbouring pixels within its gradient range, do not remove the edge, and leave as a singleton. It is either a very small texture patch (a mole hill in a field for example), or it will be absorbed by another edge as its area of influence expands.

4. Adjust the area of influence of the edges that were affected by 3. See Figure 6 for an illustration.

   (a) The minimum x and y influence are the lowest pixel coordinates that is a member of the edge, minus one in both directions.

   (b) The maximum x and y influence are the highest pixel coordinates that is a member of the edge, plus one in both directions.

   (c) Update the mean gradient value of the edge – this will be used for assessing the proximity of new pixels to the gradient value of each edge.

5. Flag any pixels that were assigned to a edge to avoid re-linking these pixels.

6. Find next pixel with a non-zero gradient and repeat steps 3 to 5 until all non-zero gradient pixels have been assigned – from both images. It is not important which Sobel image is processed first (as there will be crossovers from the images anyway).

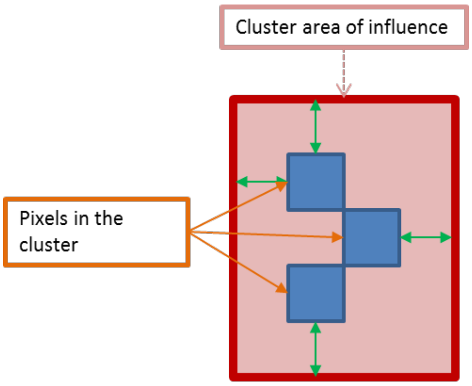


Fig. 4.7 Pixels linked in an edge, and the area of influence of the edge.

Optical Flow

Optical flow applies motion perception to the texture patches detected in the scene. Optical flow is chosen because the similarities of the method to the human motion perception model proposed by Lu and Sperling [80]. The output of optical flow will indicate the magnitude and direction of motion of texture patches in the scene. The optical flow algorithm is not applied to the entire image because this is computationally resource heavy and defeats the purpose of the Edge Flow approach. Five pixels are selected within each cluster, one from the centre and four from the extremities of the edges that make up the texture patch. The five points are chosen because this is the minimum required to represent the 8 degrees of freedom of movement in a three dimensional plane. Optical flow [81] is applied to these individual points which yields a flow vector for each of the individual points within a texture patch. By calculating the movement vector of each texture patch, we can determine the similarity of movement between texture patches. If the motion vectors are similar, and the texture patches overlap in x-y proximity in both frames (two frames required for optical flow), the texture patches could be considered to belong to the same physical object. The optical flow output also allows separation of texture patches that are in spatial proximity, if they have different vectors of motion. A car moving along the road and a crack in the road is a nice example; the car texture patch will have a different motion vector to the crack in the road despite being in spatial proximity when the car goes over the crack, so despite visually occluding the crack, they can be considered as separate objects and will not merge as the same object (unless the crack is completely occluded).

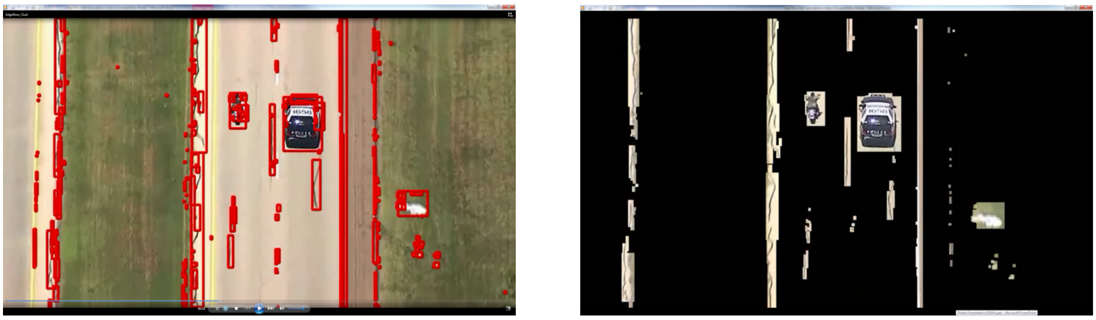


Fig. 4.8 (a) Contiguous clustering of gradients from Sobel stage (b) Result of optical flow applied to clusters

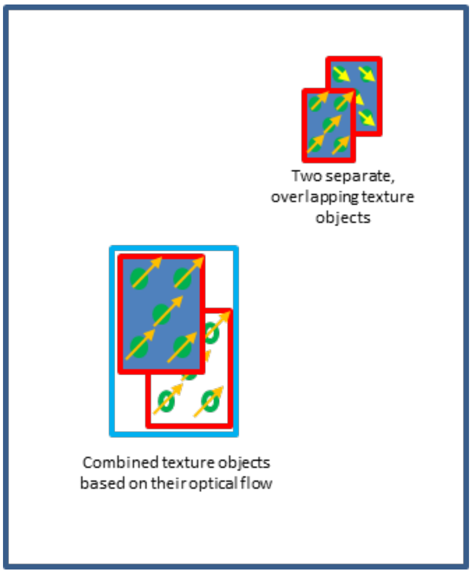


Fig. 4.9 Two separate scenarios for texture patches with optical flow calculated for each of the 5 pixels within them.

The optical flow method is used both for motion information on texture patches, and to define which contiguous texture patches form candidate physical objects. Optical flow, as mentioned in chapter 2, can be used independently to detect and identify objects and their movement. However without large computational resources, it does not operate in real time. This usage of optical flow is made tractable compared to the sole use of it in video processing by limiting the number of pixels used in the optical flow calculation (five pixels of a texture patch), usually resulting in one or two thousand points in total. This is a significant reduction to the number of pixels in an entire image. A 640x480 image contains over 300,000 pixels, and optical flow would be applied to all of these pixels when directly to the source image.

Table 4.1 Characteristics of some sample novelty detection algorithms

Algorithm:	KDE	RDE	Dual TVL1 OF	ME (ARTOD)	Edge Flow	WISE
Static Object Static Viewpoint	N	N	N	N	N	Y
Static Object Dynamic Viewpoint	N	N	N	N	Y	Y
Dynamic Object Static Viewpoint	Y	Y	Y	Y	Y	Y
Dynamic Object Dynamic Viewpoint	N	N	Y	Y	Y	Y
FPS 640 x 360	3.00	98.20	0.17	4.60	58.00	32.90
Multi Object Detection	Y	Y	Y	Y	Y	Y
Static / Dynamic object discrimination	N	N	N	N	Y	Y

4.3 Experimental Results for Edge Flow

Table 4.1 shows the characteristics of a selection of algorithms used in novelty detection in video streams. Both KDE [37] and RDE [5] are only suitable for static camera scenarios; these are included to show the capability differences between applications designed for static and dynamic camera scenarios. Optical flow [55][81] and Motion Estimation [145] are well known examples of algorithms that detect moving objects in a dynamic camera environment. One of the latest Optical Flow algorithms is the Duality Based Optical Flow TVL1 algorithm proposed in Zach et al [157] and will be used as a comparison with the proposed approach for Optical Flow. Motion estimation was proposed in [151] and has been improved on several occasions to enable reliable novelty detection [151][8][126]. The motion estimation comparison uses the ARTOD version [126]. KDE and RDE are not compared in the video sequences, because they are unsuitable for dynamic camera scenarios. A selection of videos were used to demonstrate performance across different scenario types and resolutions. The practical implementation of Edge Flow has parameters that can be changed to yield the best results for a particular scenario i.e. its sensitivity to minor objects can be adjusted. The parameters that can be adjusted are:

- A threshold range for a gradient to be considered an edge; a minor gradient change can be a texture oscillation and not a true edge. This parameter adjusts the sensitivity of object detection based on gradient magnitude.

- A threshold range defining the gradient differential an adjacent pixel needs before being considered a separate object. This parameter adjusts the sensitivity of the algorithm to occluded objects.

- A window size for the windowed background subtraction section (RDE)

Each parameter was pre-selected for consistency with the other algorithms. The parameters were selected based on optimum performance for the videos used [148, 96, 118].

- Threshold range for object detection sensitivity: +/- 20.

- Threshold range for occlusion sensitivity: +/- 20.

- WiDE frame window size: 3.

The testing of detections in video streams can be subjective; what constitutes an object in a video stream? The subjectivity is exacerbated with the proposed approach because it is designed to detect static objects as well as moving objects. With algorithms that detect moving objects false positives are detections that do not correspond to a moving object, and false negatives are missed detections of moving objects. In order to minimise the subjectivity and obtain repeatable, quantifiable results, constraints on what constitutes a detection are applied to the experiments.

- True positive novelty detections, which are larger than 10 pixels width and 10 pixels in height. These values are chosen because a size smaller than this with either motion estimation or optical flow appears as noise, not a clear object. A filter is applied to Edge Flow to show objects greater than this size. Edge flow has the capability to show more than this (minor objects such as disturbed earth from planted IEDs for example)

- False positives are defined as detections that do not represent an object; detections on areas with no distinct contrast with the background. Edge flow has the capability to handle occluded objects, and as a result shows detections within detections. These are not considered as false positives unless it is clear the nested detection does not represent an object or an internal structure of the object

- False negative detections are defined by objects that have not been detected that meet the criteria of a true positive detections (including static objects).

The full videos used in this work can be observed on You-tube [148, 96, 118].

4.3.1 Video 1 - Helicopter chase with car and motorbike

The first test video is the helicopter and motorbike video which has been used consistently throughout experiments in this work. It provides for a reliable motion pattern with known objects present.

Fig. 4.10 Motion estimation result from scene (left) Edge flow result from scene (right). Red boxes are included on edge flow to highlight detections more clearly. 640 x 360 pixels

(a) Edge Flow (b) Motion Estimation (c) Optical Flow

Fig. 4.11 A comparison of Edge Flow with Motion Estimation and Optical Flow on the Helicopter video

The scene used in this experiment is shown in figure 4.10.

Table 4.2 Detection performance for video 1

Algorithm	Total Detections	TP	FP	FN
Motion Estimation	2	2	0	13
DF TVL1 OF	3	2	1	13
Edge Flow	11	9	2	6
WISE	1069	1069	0	3

Figure 4.10 shows the motion estimation result which identifies two objects, the motorbike and car. Edge flow detects other objects as well; the static white objects on the right of the road, the road defects, road markings and the road verge. Table 4.2 shows the empirical results from each algorithm. The detection with Edge Flow are limited because of the filtering condition applied to the detections, below 10 pixels width or 10 pixels in height, and with the filtering removed it detects a much wider range of objects although the number of false positives increase. One of the limitations of the technique is the requirement to specify the parameters to achieve the desired level of detection detail.

4.3.2 Video 2 – Dashboard mounted

The frame from this video sequence is shown in figure 4.12.

Fig. 4.12 Second test video, dashboard mounted camera

(a) Edge Flow (b) Motion Estimation (c) Optical Flow

Fig. 4.13 A comparison of Edge Flow with Motion Estimation and Optical Flow on the dashboard video

Table 4.3 Detection performance for video 2

Algorithm	Total Detections	TP	FP	FN
Motion Estimation	29	3	26	37
DF TVL1 OF	11	6	5	34
Edge Flow	52	37	15	3
WISE	1932	1932	0	40

The significantly more complicated scene yields a higher detection rate for all algorithms. Table 4.3 shows the detection outcomes of each algorithm. One of the limitations of the clustering component is that it can include multiple objects in the same cluster, as in this case with the white van and the white road markings. Otherwise several static and moving objects have been distinguished separately.

4.3.3 Video 3 – Drone launch, multiple motion vectors

The video is a high density complex scene of static objects with the occasional small moving object (cars / vans), shown in Figure 4.14. The results of the detections are shown in Table 4.4. The algorithm is capable of discriminating between groups of houses, other landmarks and excluding the general background (the forest in this case). The poor performance by both the optical flow and motion estimation techniques is due to the image warping or the brightness pattern tracking not being able to keep up with the rate of change of the camera perspective, another important limitation of existing methods.

Fig. 4.14 Video 3 scene - drone flying with multiple axis of motion

(a) Edge Flow (b) Motion Estimation (c) Optical Flow

Fig. 4.15 A comparison of Edge Flow with Motion Estimation and Optical Flow on the UAV video

Table 4.4 Detection performance for video 3

Algorithm	Total Detections	TP	FP	FN
Motion Estimation	15	2	13	125
DF TVL1 OF	5	4	1	123
Edge Flow	141	119	22	8
WISE	6932	6932	0	71

Table 4.5 illustrates the processing performance comparisons between each algorithm. The results were obtained by processing each video sequence for 500 frames and recording the minimum, maximum and average frame rate.

Table 4.5 Performance analysis of each algorithm across each test video stream

Algorithm	FPS	Dual Flow TVL1 OF	ME (ARTOD)	Edge Flow	WISE
640 x 360	Min	0.16	2.84	19.59	24.97
	Max	0.17	8.48	82.77	43.85
	Avg	0.17	4.58	58.01	32.99
848 x 480	Min	0.14	1.00	7.31	11.03
	Max	0.16	5.38	39.44	21.69
	Avg	0.15	2.60	24.35	18.69
1920 x 1080	Min	0.04	0.22	0.66	0.80
	Max	0.04	1.09	6.25	5.10
	Avg	0.04	0.35	1.85	3.81

Using this new approach, the detection of texture patches can be carried out accurately and in real-time. In this work we demonstrate the capabilities of the algorithm on video scenarios, and show that object textures in the scene are reliably detected. We are able to show clearly the capability of the algorithm to be robust in occlusion scenarios; working in real-time, and defining clear objects where other techniques attribute such small detections to noise. The method set out in this work is novel in its approach to addressing / approaching the moving camera problem in detecting *all* objects in a scene. All existing techniques assume that foreground objects of interest must be moving or changing in some way and can only detect such objects. This method enables both moving and static (unchanging) objects to be detected. This is a significant step forward, paving the way for detections of small minor objects as well as the large moving parts of a scene. Also, the method does not make any prior assumptions about the scene, and is wholly data driven. The latter statement is critical; what other techniques dismiss as noise or unimportant, this technique extracts and highlights it as an object texture. This enables retention of information which would otherwise be lost at the detection stage, which can be filtered and analysed as required. Key objects or people can easily disappear into the background if the detection algorithm dismisses small or "noise-like" novelties early on. This can later be filtered out based on the object parameters the analyst is looking for (type, size, motion, texture etc. of the detected object).

The direction and relative speed can be associated with the edge gradients – a sharper gradient indicates a higher relative speed, with the sharper gradient being the leading edge of the object texture. This form of clustering is robust and combined with the first two components is resistant to occlusion. Should an texture patch be occluded partially by another texture patch, they will remain separate clusters unless the object is completely occluded. Further, once the occluding object has moved on, the cluster will return to being a separate texture patch.

Currently, the approach is parametric, requiring a magnitude range to be defined. This magnitude defines the similarity of candidate pixel gradients required to be linked together as an edge. In principle it is possible to autonomously define a gradient magnitude range but this will be left for the future. Through this method each similar and proximate edge are clustered together, resulting in a contiguous object being defined for each different texture (object with edges); an object is defined as an area of similar texture, not as an isolated object per se. For example, a car may be defined as 3 separate texture patches in edge flow – the bonnet which is of a particular texture, the roof which is a different texture, and the boot which is the same texture as the bonnet but separated by the roof. The main innovations of this approach are; A motion vector can be extracted from each texture patch within a scene in real-time. Objects which are moving in different directions but are spatially proximal can be clearly separated despite any occlusion in the scene. The motion vector is a representation of the relative velocity of an object compared to the camera platform; later, given the platform velocity, this can be used to determine the absolute velocity of all the objects within a scene. With the inclusion of optical flow in the method, the average processing time remains around 20 frames per second (50ms per frame) for a 640x480 video stream. As with the clustering technique the processing time changes slightly dependent on how many objects are detected. The following advantages are introduced by Edge Flow:

1. It works well with partially occluded texture patches and keeps them separate until completely occluded,

2. It rediscovers the texture patches post-occlusion,

3. Static and moving texture patches are clearly separable, and

4. The processing speed combined with the first two components of Edge Flow remains real time (between 25 – 40 fps depending on the number of texture patches discovered in a scene).

This is significantly faster than other methods, and still permits some head room for additional processing. An example of the occlusion discrimination capabilities can be seen in figure

4.11; if a car drives over a crack in the road (both of which have been clustered and identified) the clusters will remain entirely separate unless the car completely occludes the road crack. Once the crack appears the other side of the car it is immediately re-discovered and clustered as a separate object texture.

4.4 Discussion of the Edge Flow Algorithm

A new approach to video analysis has been demonstrated in this chapter. It has the following main components:

1. WiDE
2. Gradient Estimator using Sobel Filters
3. Contiguous Clustering

The computational performance of the proposed Edge Flow has been demonstrated to be in an order of magnitude faster than motion estimation and optical flow. The capability of the algorithm to detect static objects as well as those that are moving are also demonstrated here. A limitation of the approach is the parameters applied to Edge flow, without correct selection, in complex scenarios, is that it can yield a cluttered environment; unlike optical flow and motion estimation which are excellent at isolating the moving objects in a scene. Edge flow can detect both static and moving objects and discriminate between occluded objects. One limitation suffered by all algorithms is that objects which have motion in synchronisation with the camera platform make detections somewhat more difficult - there is no change between frames to allow detection. The limitations of scene clutter (caused by "over-detection" of objects) can be overcome through parameter selection according to the type of object to be detected. Further work will focus on optimising this method for specific applications, and introducing an improved contiguous clustering method.

4.5 Within Image Spatial Edge Flow (WISE)

Humans can easily identify objects for example simply looking out the window, or glancing around the office there are several objects; some inert, some in motion, and others partially occluded or unclear. As explained in section 4.1 Biederman [18] posed a similar problem and proposed a framework to describe the object detection process. In this framework the first suggested activity is edge extraction by reaction to changes in colour and textures; defining the boundaries of the textures. Biederman then goes on to describe further analysis of the detection based on the properties of the edges. The framework diagram for this analysis is shown in figure 4.16. Edge Flow applied this but used a temporal component for motion which aligns with the Biederman model but not that of Wertheim (because in his model, he proposed separating temporal and spatial information, applying them in separate processes). This new method explores the application of Edge Flow concepts in the spatial domain, and restoring temporal information (to allow for motion perception), after texture patches have been detected.

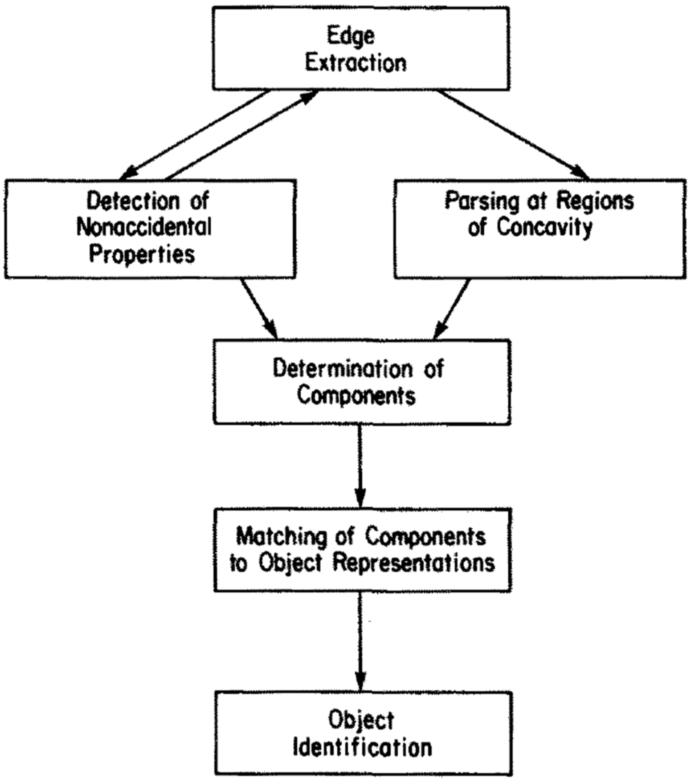


Fig. 4.16 Conceptual framework for human object detection, [18]

Working with this information, we can construct an artificial equivalent that focuses on extracting the edge contrasts within the optical field of a camera and determines particular textures. Separately we can apply motion perception and understanding in an independent parallel paradigm to the object detection and identification. Some methodologies take a different approach and consider detection as a computational modelling perspective.

4.5.1 Traditional object detection and image segmentation

There are many widely used approaches for object detection in camera produced images, with broad contextual differences. Background subtraction, [111], works where the camera is stationary and the aim is to detect the moving objects in the scene. The aim in motion estimation [145] and optical flow [55] is similar to the background subtraction but compensating for any camera motion [126]. In each of the detection schemes, assumptions are made about the content of a scene [48] or apply contextual restrictions on the scene based on holistic viewpoints [54] or probabilistic modelling [144]. Each technique targets the detection of moving objects. The constraints applied when making assumptions about a scene enables a measure of success for many of these techniques. However, in a busy surveillance scene the number and different types of objects can be broad and unpredictable. This can result in false detections and misrepresentations of objects using these approaches. Image segmentation [143] is used to divide and classify detections within the visual scenes without applying the assumption of motion; in fact, there is no motion preservation in image segmentation techniques. Image segmentation is applied to a single image and can be carried out sequentially over a sequence of frames to achieve detection in a video stream. A number of research papers look into improving each of these methods, by the accuracy of the detections or decreasing computational load to achieve the same results on hardware with a lower computational capability [68], [24], [71], [26], [150].

4.5.2 Edge Detectors

Edge detection [23] is used as an object boundary detector and can be used in isolation in both static and moving environments or to reinforce segmentation [140, 101]. The edge detector described by Canny [23] uses a Gaussian kernel convolved with an input signal to determine the location of an edge. In the case of a one-dimensional signal it finds the peaks, troughs and changes in the signal. There are many different types of edges, and the method can distinguish between ”square” edges, ”roof” edges and other profiles. In the two dimensional plane, i.e. an image, the kernel also has to be convolved in two dimensions. The resultant values are then assessed by using a Sobel operator to fully understand the gradient directions and magnitudes of the convolution. The complexity of the calculation is in the kernel convolution, with the Sobel operator being a relatively low cost action. One of the limitations of using these kinds of edge detectors is that little is suggested about the internal structure of any objects. The detector does not impart which edge is part of which object or whether it is part of any object at all. Therefore, to consider finding objects or texture patches in a scene, some additional edge or object boundary information is important.

The proposed new approach does not make assumptions about the video stream, and concentrates on emulating what the human vision system does to separate objects whilst maintaining the separation described by Wertheim [153]. The primary function of the method described here, which we have dubbed Edge Flow, is edge or boundary detection between textures, and imparting greater detail about the internal structure of each texture. Whilst the Canny solution detects edges, it does not extract further gradient information, merely whether it is an edge or not, and of what type (ramp, roof etc.). Edge Flow extracts the magnitude of the change in contrast and the rate of change (gradient) of this magnitude which provides a much more detailed information set about the texture patch. The ultimate goal of the Edge Flow method is to replicate the framework set out by Biederman [18] in an artificial environment.

4.6 Methodology

To solve the identified limitations of Edge Flow and to follow the Wertheim visual perception model, the framework is modified by changing from the time domain to the spatial domain. This method, which has been dubbed Within-Image Spatial Edge Flow (WISE), is more capable than the Edge Flow technique [98] because of the ability to detect static objects when the camera platform is also static. Throughout the work presented, there have been many references to existing techniques that assume foreground objects of interest must be moving or changing in some way and can only detect such objects. The improvements of the Edge Flow method do the opposite; at the detection phase all object edges are important until a semantic process deems otherwise. The analysis differs in that it uses a single frame as opposed to a sequence of frames as with the time domain analysis. That effectively means that every object in the scene is stationary. The motion of objects is restored at a later stage by applying optical flow; referring back to the logical steps of human vision, where the detection of objects and relative motion are thought to be treated separately [152]. As with the Edge Flow method, density estimation [7] is used in this method. The modification is a change to how the density estimation is applied to the image. The changes are outlined in the updated flow chart for WISE, see figure 4.17.

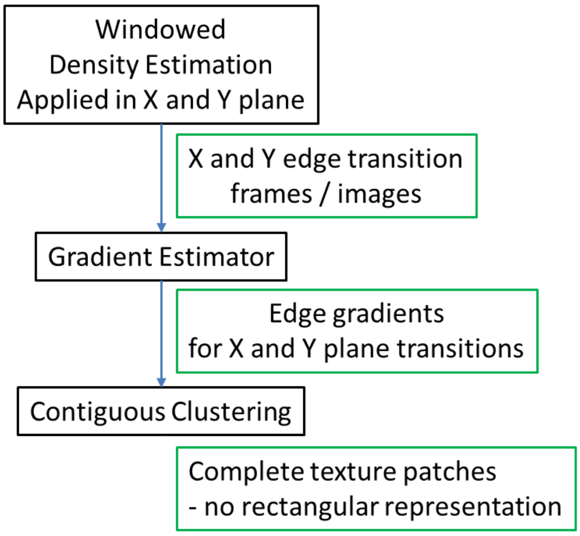


Fig. 4.17 WISE components, and in green, the output at each component stage

4.6.1 Windowed Density Estimation applied in the spatial domain

The application of the WiDE technique in the XY plane is detecting changes in pixel colour density in the X and Y spatial directions yielding two density estimation images. The WiDE equations remain the same as in the Edge Flow method (section 4.2, [98]). However, the window no longer represents how many frames the density is observed over, it is how many pixels the density is observed over.

4.6.2 Gradient Estimator

The gradient estimator is applied to the XY plane images in a similar manner to the Edge Flow method. The X plane density estimation is applied with the Y Sobel operator, and similarly the Y plane density estimation is applied with the X Sobel operator. This is because the X plane density estimation will only detect significant lateral changes from objects orientated between vertical and 45 degrees from vertical. As observed from the previous gradient estimation, it is the Y filter that best represents these transitions. The same principle applies with the Y plane, X Sobel operator application.

4.6.3 Contiguous Edge Linking

WISE uses an updated edge linking method which groups adjacent pixels that are within a set gradient range as before, however, this method does not make the assumption that each texture patch is rectangular. It maintains an irregular area of influence around the contiguous pixels to enable the merging of adjacent edges with the same parameters. The update to the method ensures that edges not part of a texture patch cannot be inadvertently added to an edge due to its rectangular nature; a problem that is present when using the Edge Flow edge linking. Each pixel now has its own single pixel area of influence which is not adjusted in all dimensions each time a new pixel is linked with an edge; the edge shape changes to incorporate the new pixel, without over-extending the area of influence of the existing pixels. Figure 4.18 illustrates how the edge area of influence updates. The edge linking method still has the two parameters associated with it and they operate in exactly the same manner as the Edge Flow contiguous edge linking.

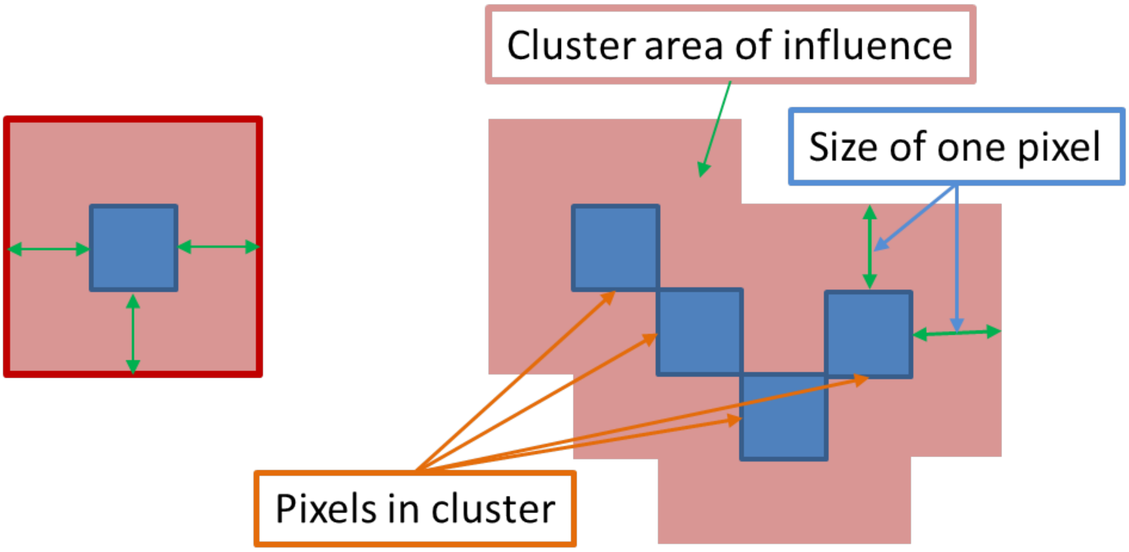


Fig. 4.18 An illustration on how the edge influence (and thus membership) propagates across the image

Computationally, to enable the irregular edge shape, a modification to the linking procedure is also required. The pixels adjacent to the current pixel are not immediately added to the edge; a flag is set instead indicated to which edge this pixel should be added. The pixel is only added to the edge when it is the current pixel being analysed - consequently its area of influence is also then assessed. The proposed methodology can be summarized in the following steps:

1. Working from the top left of each image (x-plane and y-plane Sobel images) find the first pixel with a gradient value outside the acceptable gradient exclusion range.

2. If the pixel does not have an edge flag set, create a new edge starting at this pixel; otherwise link the pixel to the edge indicated by the flag. In this case, update the edge mean gradient.

3. Assess the area of influence; 1 pixel either side of the pixel. Important: The area of influence extends into both Sobel images - this enables the merging of objects from the x and y shifted images into a single frame. If a neighbouring pixel is within the density adjacency range of the edge and has no flag set; set the flag of the pixel to be a member of this edge. If a flag is already set, add the edge number of the current pixel and the edge number of the neighbouring pixel to the merge list. Later, this list is used to establish which edges overlap each other for the merger purposes.

4. Repeat steps 2 to 4 until all pixels in both images have been linked.

5. Filter the edge merge list to only include single instances of a pairing.

6. Merge overlapping edges into a single edge by adding pixels from one edge into the other.

7. Calculate the centre point of each edge boundary, this is done by assuming a rectangular formation to simplify the calculation for each texture patch.

At present, the WISE edge linking method lacks full autonomy due to the requirement of setting a gradient similarity parameter to inform the linking process. However, with the deployment of the system in scenarios with different detection objectives it may be preferable to provide the analyst or operator with a manual ability to adjust the gradient parameters on system initialisation. The parameters are not fixed for the life of the system, and can be changed on-line should the objectives of a scenario change.

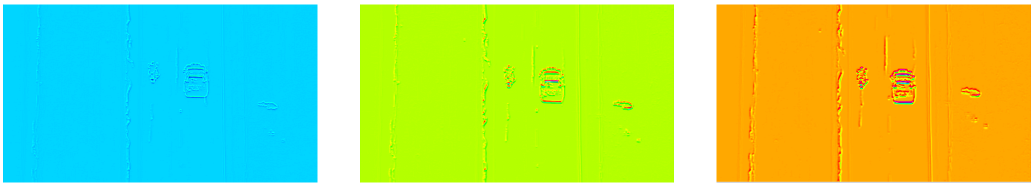
4.7 Results of Experiments with WISE

The WISE method was tested against the same experimental scenarios as Edge Flow. The experiments are presented in two sections. Section one tests the core components of the algorithm (edge detection and edge linking) with existing methods (edge detection and image segmentation), and section two focuses on the overall performance compared to the results seen with Edge Flow, Motion Estimation and Optical Flow.

4.7.1 Edge Detection Results

(a) (b) (c)

Fig. 4.19 This set of images shows the results of the WISE technique when applied with different window sizes. (a) is with window size 2, (b) is a window size of 3, and (c) is a window size of 4. The Sobel filter has a consistent size of 7x7 for each image

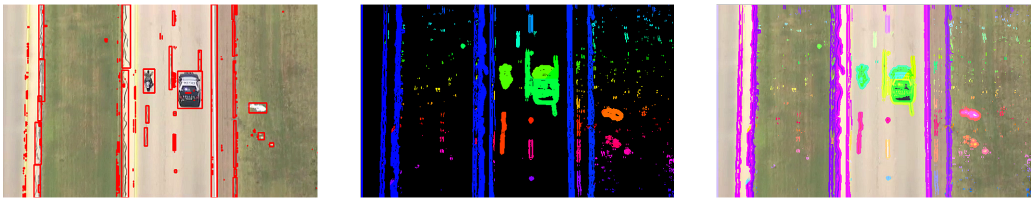


(a) (b) (c)

Fig. 4.20 This set of images shows the results from changing the size of Sobel filter from 3x3 (a), 5x5 (b) and 7x7 (c) with the WiDE window size set to 3.

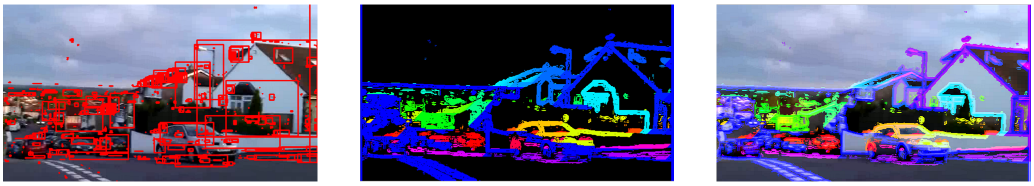
For the edge detection results, a false colour overlay has been used to illustrate the varying gradient magnitudes and directions of the density from the Sobel filter application. In each case, the actual false colour value varies due to the colouring algorithm selecting the distribution of colour based on the maximum and minimum gradient values across the input frame. Where there is little variance the colour scheme appears similar. The purpose of this is to highlight the importance of gradient variance and uniqueness to produce an interpretable dataset. The first set of results is WISE tested on the helicopter video scene. In each of these figures a 7x7 Sobel filter is used, and different levels of windowing across pixels to show the variance of quality as the window size changes. The combined image is created using additive distancing with the X and Y Sobel images such that the negative gradient information is not lost. When a window of 2 pixels is used, the image is poorly defined and some edge components are missed, figure 4.19(a). In figures 4.19(b) and 4.19(c) the window size of WiDE is extended to 3 and 4 respectively. We can see that as the window size is increased each of the detected edges are more defined and pronounced than that of figure 4.19(a). The window size of 4 is such that some of the edges are defined thicker than the edge boundary in the frame however the definition of the fainter lines such as the yellow road markings are more pronounced with a larger window size. The results suggest that in a scene where the requirement is to detect faint texture patches in a scene, a larger WiDE window size should be used. Conversely, in a scene with small, narrow texture patches, a smaller WiDE window size should be used to avoid the overly large line thickness interfering with the output definition. Adjusting the size of the Sobel filter has a different effect to the change in density estimation window size. The Sobel filter influences the local area of a pixel and its density disparity across the frame. With a larger Sobel filter, there will be a smaller distribution, of higher variance of pixel gradients in the frame, and with a smaller filter there will be a larger distribution of smaller local variance of gradient values. The effect of modifying the Sobel filter size is shown in figures 4.20(a), 4.20(b) and 4.20(c). A consistent window size of 3 pixels is used in each case and the filter sizes are 3x3, 5x5 and 7x7 respectively. The colour scheme, as before, represents the distribution of density values across the input frame. In figure 4.20(a), the gradient values have a sufficiently small distribution so that the colour variance across the scene is similar and does not yield clear differentiation of the output gradients. As the filter size is increased to 5x5, figure 4.20(b), the distribution of gradients and thus separation of the gradient edges is greater and clearer. Extending to 7x7 yields a sharp, crisp display of varying gradients in different areas across the scene, figure 4.20(c). If this is the trend, the question is why not use an infinitely sized Sobel filter to achieve the greatest gradient separation? The drawback with larger and larger Sobel filters is that there would be no consistent edges from which to cluster given the increased global variance that a larger Sobel filter yields. A frame sized Sobel filter, for example, would not only increase the processing requirements, but the locality of the change would also be lost; no longer would a pixel's gradient be a locally based variance. It is therefore important to achieve a balance between the locality of gradients and the distribution of gradients across the scene. In all cases of the spatial domain analysis, the ghosting problem in Edge Flow has been resolved. In these examples we can also see the groundwork of how the approach is able to be robust in occlusion scenarios. In the analysis of the road cracks imagery, note that the gradients or colour change accelerations between the motorbike, car and the road cracks are significantly different, figure 4.20(c). This factor provides an excellent feature differentiation when it comes to clustering, allowing both objects to be distinguished separately.

4.7.2 Edge Linking Results



(a) (b) (c)

Fig. 4.21 Here is a range of visualisations for the clustering results of WISE. The first (a) is a view of the clustering, using a bounding rectangle to include the pixels that represent and individual texture. The bounding box image is included to highlight the extent of each cluster and reinforce that the clusters are separate distinguishable textures. Both the (b), and (c) images show the actual pixels of a cluster represented with a colour overlay.



(a) (b) (c)

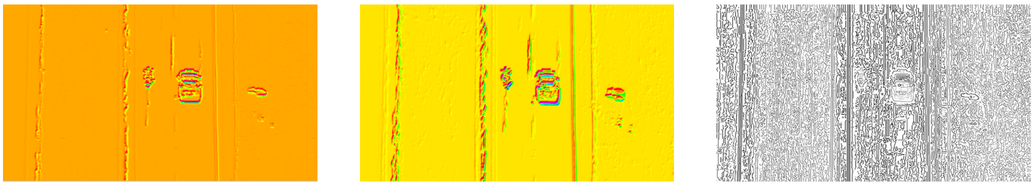
Fig. 4.22 Clustering results for WISE applied to the busy road intersection scene, introducing a greater 3D differential for the algorithm to handle. Figure (a) shows the rectangular cluster representation. Figures (b) and (c) show the clusters pixel-wise using a colour overlay. One is just the cluster pixels (b), and the other is the cluster pixels overlayed onto the original image (c)

The clustering results are displayed in two different styles to aid with visualisation. The red rectangular clustering shows the extreme boundaries of each dimension (x and y), and the colour overlay visualisation shows the actual boundaries of the clusters as they are arbitrary shapes. For the colour overlay, false colours are used. Due to the number of clusters formed, the colour values can be similar enough to look as if they are part of the same cluster. The two representations are necessary to illustrate the cluster separation (figures 4.21(a) and 4.22(a) and to show that the actual cluster dimensions are irregular not rectangular (figures 4.21(b), 4.21(c), 4.22(b), and 4.22(c). When clusters are narrow, 2 pixels or less wide, the bounding box method cannot draw the bounding box however the clusters are shown in the overlay images. This is particularly observable with the yellow road markers in figure 4.21. In figure 4.21(a) there is an example of the occlusion separation capability of the algorithm. The road crack is shown to be moving underneath the car, however there are two clusters formed despite this occlusion; the car cluster and the bounding box representing the road defect cluster.

To produce these results, the full WISE algorithm was operating at 60ms per frame on an Intel Sandy Bridge 2700k processor, using four cores for the density estimation component and a single core for the remainder of the components.

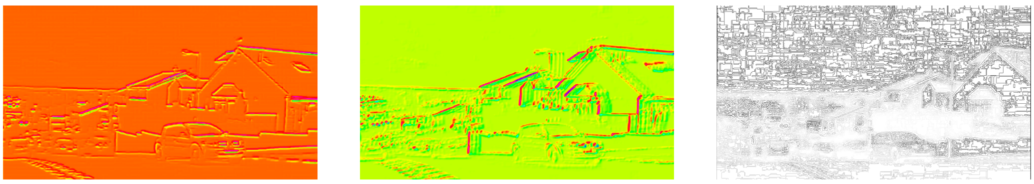
The WISE algorithm is compared with the Canny edge detector, Edge Flow, and two image segmentation techniques, Mean shift and Grab Cut, to explore the difference in performance and fidelity of the work. The reason for the selection of these techniques for comparison is that they do not make any assumptions about the scene, nor do they have any semantic derivation component.

4.7.3 Edge Detection Comparison



(a) (b) (c)

Fig. 4.23 Edge detection results for WISE (a), Edge Flow (b) and Canny Edge Detector (c) applied to the helicopter scene

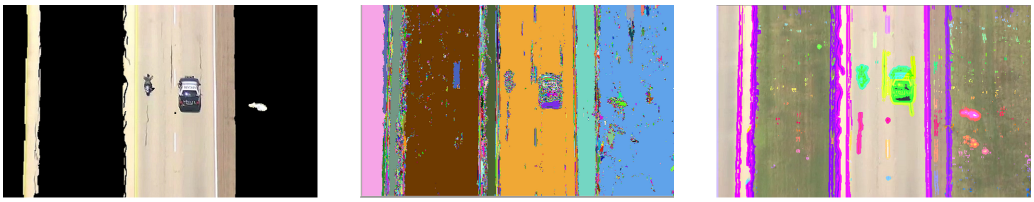


(a) (b) (c)

Fig. 4.24 Edge detection results for WISE (a), Edge Flow (b) and Canny Edge Detector (c) applied to the busy road intersection scene, introducing a greater 3D differential for the algorithms to handle

The Canny edge detector, figure 4.23(c), successfully detects the main edges in the helicopter scene and joins up the boundary of the textures. The car and motorbike are distinguishable along with the white side object and the road markings. A drawback however is that all other minor colour variances are detected as edges including differences in grass texture. This yields a cluttered set of detections and masks clear definition of road defects and earth disturbances on the verge. In contrast, both the WISE and Edge Flow technique create a clearer separation of the textures of the scene. Edge Flow introduces some ghosting of edges, and some edges are wider than the definitions in WISE. The Canny Edge Detector has already made the connection of boundary edges such as the car, motorbike and object on the side of the road whereas WISE and Edge Flow have not made that determination at the edge detection stage. In the more complex scene 4.24(c), the edges of the houses, the lamp post and foreground car are well defined and clearly separable. The difficulty with the Canny edge detector comes where the minor variations in cloud cover have all been considered as edges as well as some of the other objects combining together to form an unclear picture. The performance of WISE in the simple picture is much clearer and more interpretable than the Canny approach and separates static, moving and occluded edges The ghosting effect with Edge Flow is much more apparent in this scene leading to multiple lamp posts being visible in the results. Some of the detections from WISE appear as partial edges at this stage. The definition of partial edges is improved during the edge linking process.

4.7.4 Edge Linking (Texture Patches) Comparison



(a) (b) (c)

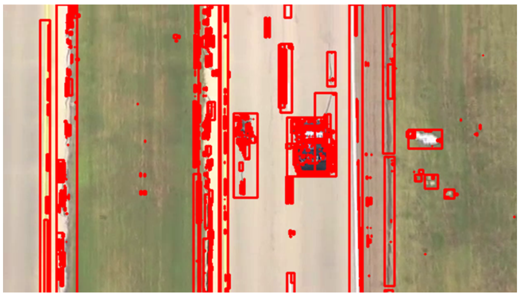
Fig. 4.25 These images show the comparison of Grab Cut (a), Mean Shift (b), and WISE (c) on the helicopter scene

(a) (b) (c)

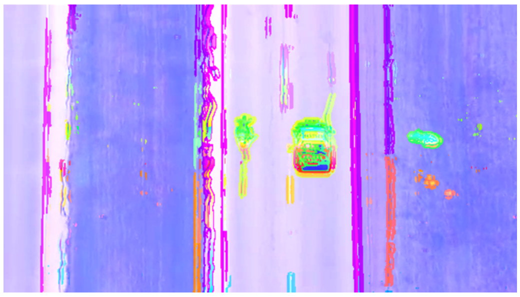
Fig. 4.26 These images show the comparison of Grab Cut (a), Mean Shift (b), and WISE (c) on the helicopter scene

When considering the image segmentation approaches, the grab cut algorithm does not perform well with an entire scene. Both 4.25(a) and 4.26(a) are whole scene selections with grab cut. In the case of the first frame, the road is identified as a separate segment and so is the white object on the side of the road. There is however no clear separation of the car and motorbike from the road. Object separation was only achievable when grab cut is applied to a selected small area of the scene. Similarly, in the second image very few objects are segmented by the algorithm. Only when the selection area is reduced does the grab cut algorithm extract the house and car as separate objects. The processing time of the Grab Cut technique was 2 seconds per whole frame, and reduced to 500ms when smaller areas of the frames were selected. Mean-shift has more success with both of the images 4.25(b) and 4.26(b). The simpler scene is well segmented with the verge, car, bike and white object all separated. The road markings and defects are also segmented well. The image is somewhat over-segmented, detectable by the many internal false colours to each texture patch, suggesting that the mean-shift algorithm is detecting the minor variances in surface texture or luminance and creating smaller internal segments. The more complex scene is segmented well with the houses, car and lamp post easily distinguishable. The light blue area represents an area where the dark side of the house and the hedge row have not been separable and are thus represented by one large segment. Despite excellent segmentation performance, the processing time is an order of magnitude slower than any of the other techniques (>5 seconds using an Intel Sandy Bridge i7 2700k processor). The texture patch segmentation of WISE in the first frame performed better than Grab Cut, and was similar in performance to Mean Shift segmentation. There was less over-segmenting of objects, which makes the scene appear less cluttered. Over segmentation can be avoided by WISE by adjusting the clustering parameters. The complex scene is different with foreground objects segmenting well into textures. The glass part of the car, bodywork, and wheels are segmented into different areas, along with road markings and the house being clearly separate texture patches. Further into the 3-dimensional depth of the scene the texture patches are less clear, however parked cars and buildings are consistently separable. The processing performance of the WISE algorithm was 80ms per frame using an Intel Sandy Bridge i7 2700k processor.

4.7.5 Overall performance results

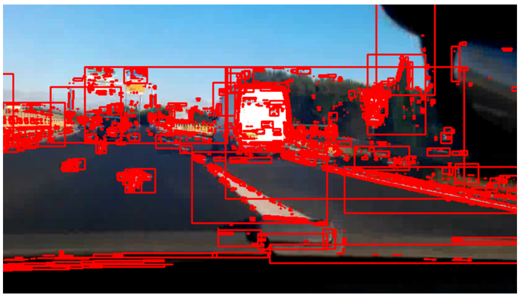


(a) WISE showing texture patch extremes

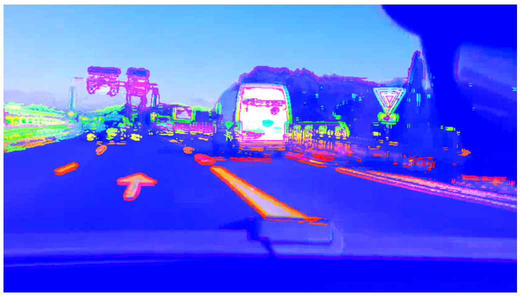


(b) WISE showing the pixels that make up the texture patch

Fig. 4.27 WISE applied to the helicopter scene.

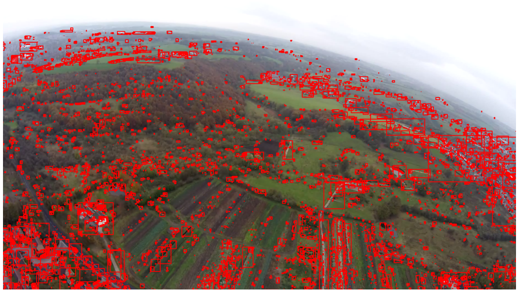


(a) WISE showing texture patch extremes

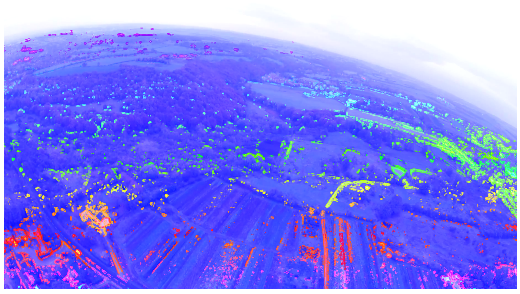


(b) WISE showing the pixels that make up the texture patch

Fig. 4.28 WISE applied to the dashboard scene.



(a) WISE showing texture patch extremes



(b) WISE showing the pixels that make up the texture patch

Fig. 4.29 WISE applied to the UAV scene.

4.7.6 Analysis of the WISE Method

The WISE algorithm divides a scene into many more texture patches than the original Edge Flow approach.The results are appended to the Edge Flow tables 4.2, 4.3 and 4.4. WISE operates at a slower frame rate compared to Edge Flow (see table 4.5) and this can be attributed to the number of texture patches WISE is detecting. The computational performance of the method is proportional to the number of texture patches that are detected and defined (edge linked) in the frame, not the resolution of the frame as with many of the other pixel based approaches (e.g. [97], [126], [150]). Typically in a higher resolution image, there will be more texture patches to detect but this is not always the case. This performance correlation means careful management of the number of texture patches detected is required by adjusting the edge linking parameters - making the approach less assumption-free than desired. As a result the parameters used in WISE can have a large effect on the performance of the algorithm. If the candidate edge gradient parameter is too small, the technique detects every minor texture change in the scene, and each shows as a separate texture patch. The WiDE component of WISE increases linearly with the increase in the number of pixels processed. Empirically (over thousands of frames, with three different resolutions) processing time was found to increase by 79% for every 100% increase in pixels. However WiDE only makes up a small proportion of the processing time required (circa 20%). Experiments with processing time relative to the number of objects in a frame show that on average the algorithm takes 29.8 μs per object with a standard deviation of 6 μs. The object processing time calculation was conducted over a several hundred frames with three video sequences (the videos used for comparison here). The table for the WISE time performance for each frame in this testing can be found in Appendix B. In the results, the first two frames and the last frame are not counted. This is because on initialisation the processing time for the first two frames is longer, and on completion the final frame also takes a longer processing time. Including these frames in the object processing time comparison would incorrectly bias the results. In the experiments, the candidate edge gradient parameter was set at 5.0. That is gradients have to be within +/- 5.0 of the pixel being processed to be linked into an edge. The selection of this parameter was empirically derived through observations in several video sequences. A higher value begins to link clearly separate objects, and a lower value divides the scene into too many small individual textures that do not form large objects, and a large overhead in terms of processing. The Helicopter video, figure 4.27 shows several more smaller detections, and individual objects (such as car or motorbike) are divided into smaller text patches. Two display methods are used for this set of results, the first which is the same as Edge Flow, is done to enable direct comparison with the Edge Flow results. The rectangles do not represent the actual edge boundary, just the min / max of x and y of each texture patch. The second display shows the actual extent of each texture patch, which is an irregular convex hull of the pixels in the texture patch. The colour scheme is a simple RGB rainbow that cycles through each texture patch that has formed (hence some clusters appearing as the same colour). The results throughout produce tighter results around textures that are different in the scene, and the texture patch linking over occluding objects that happened in Edge Flow does not happen. For example, in figure 4.27(a) the road crack under the motorbike is clustered separately to the motorbike, even though for a period of time the motorbike occludes the crack. In figure 4.28 the white van is no longer clustered with part of the white line as with Edge Flow, it is a separate texture patch. It is in fact several separate texture patches, making up the perimeter of the van and a separate texture patch for the van writing, the van latch handle and the light clusters. In the UAV scene, figure 4.29 the algorithm detects the main scene texture patches such as the paths, the lake, the buildings and town and cars on the road. Where there is a common texture, such as the forest, only major darkness changes are detected as texture patches, due to the parameter selection made for the experiment (insufficient density gradient to form an edge). There is one lake on the edge of the forest that is not detected. Compared with Edge Flow there is a greater number of detections, and more of them are true positives. However, because of the parameter selection, there are also a large number of false positives where a texture patch has a partial edge detected but the entire edge is suppressed by the parameter exclusion.

4.8 Discussing the WISE algorithm

The development of the WISE algorithm set out to address issues and drawbacks with current available detection algorithms whilst maintaining a real-time performance capability. In the case of object detection, many of the current approaches detect only moving objects in a video stream whether the camera is moving or not (section 2.2.6) and are not robust in occlusion scenarios, section 3.3. WISE has demonstrated the capability to detect both static and moving objects independent of relative motion and that it is robust in object occlusion scenarios. When compared with edge detection or image segmentation techniques the algorithm's visual fidelity is at least as good as existing techniques. The additional benefit of WISE is that it extracts information about the context of each texture or segmented object by describing the colour boundary properties of each edge. This information could later be used in conjunction with semantic reasoning modules to identify the objects in a scene. The WISE algorithm proposed here has advantages over the algorithms to which it has been compared because the algorithm:

- Detects both static and moving texture patches independent of the motion of the camera

- Is robust to occlusion
- Reduces in edge clutter in complex scenes
- Produces feature-rich edges that improves texture patch differentiation appropriately for later stage semantic derivation
- Is real-time

The limitation of WISE is in the parameter selection required for texture patch detection and definition. A different parameter selection value produces different results. This parameter selection at the moment is application specific and is an assumption made by this method. Chapter 1 discusses the feedback methodology proposed with the cyclic framework, and the intention is, in future work, to develop this such that the parameters of WISE are automatically selected based on the detection constraints or requirements of semantic reasoning components. At this stage, part of the model proposed by Wertheim [153] has been emulated with WISE but no motion perception has been done. In order to complete the model in [153] motion perception is required.

4.9 Motion Perception with WISE

For the most part, optical flow is an intractable real-time component for object detection due to the number of pixels required to analyse. This problem can be reduced if optical flow is only applied to a subset of pixels of an image. In order to get motion perception, the proposal is to use four extreme points and the centre point of each texture patch as pixels for optical flow. The reduction in calculation points allows modern optical flow to be applied in real-time with little overhead to the processing chain. Optical flow was initially proposed by Horn and Schunck [55] and Lucas and Kanade [81]. There are many variants of optical flow, many of which are used on stereo cameras [149] [146]. For this task, an efficient monocular optical flow method is required. [157] is an up to date real time optical flow method called TV-L1 optical flow which represents an improvement in accuracy and processing speed and is later improved by Wedel [150] which optimises the algorithm further and this is the optical flow used for this experiment. The component model of WISE is also updated to reflect the motion perception component, figure 4.30

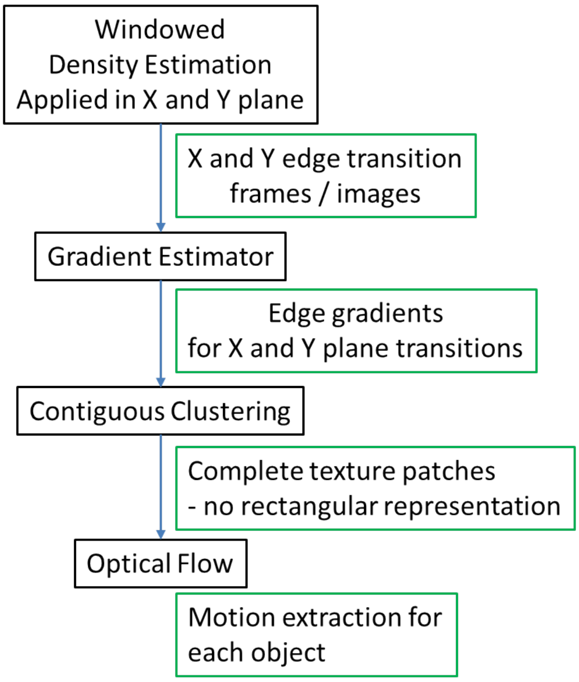


Fig. 4.30 WISE components with optical flow addition

To maintain consistency, the video sequences used in the WISE experimentation are also used to show the results of optical flow added to the WISE components.

4.9.1 Performing Experiments with the Motion Perception Restoration

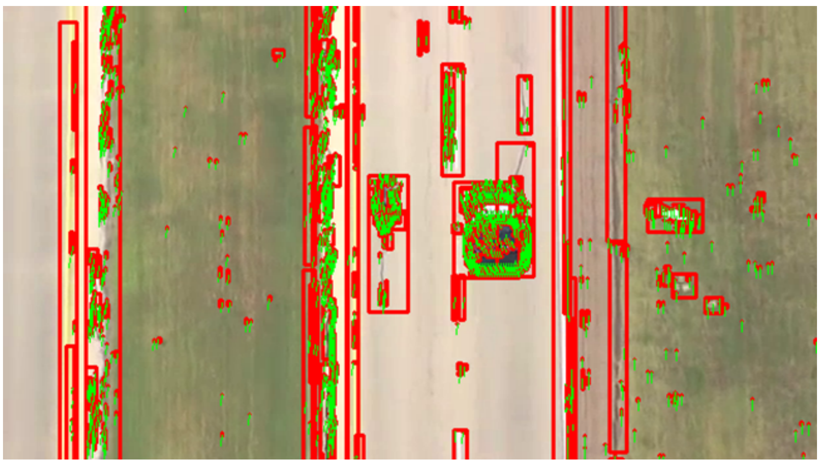


Fig. 4.31 WISE and Optical Flow applied to the helicopter video

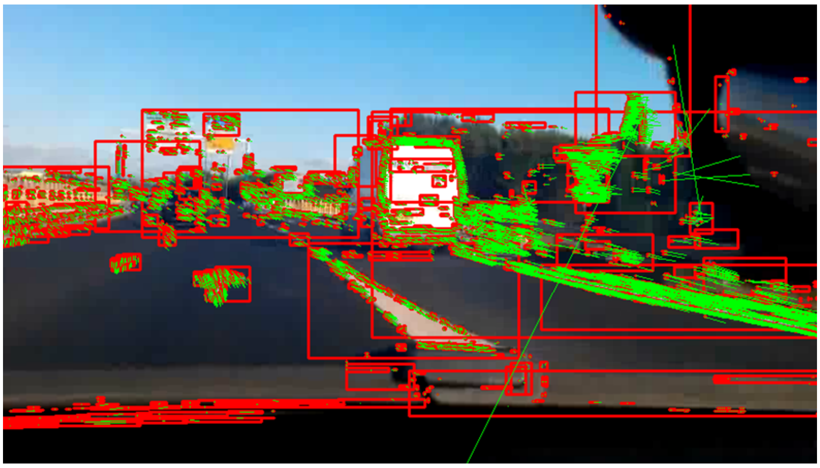


Fig. 4.32 WISE and Optical Flow applied to the dashboard video

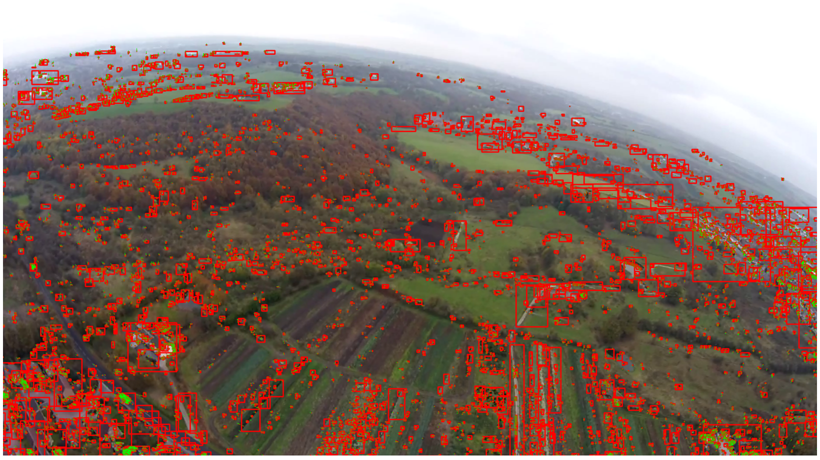


Fig. 4.33 WISE and Optical Flow applied to the UAV video

Fig. 4.34 Additional frame with greater UAV motion

4.9.2 Analysis of the Motion Perception Component

To remove some clutter, only the centre point optical flow is shown in the images (the optical flow was applied to 5 points of each object, extreme corners and the centre point). The results show point wise optical flow applied to each of the candidate objects. In figure 4.31 the motion of the camera gives stationary objects such as disturbed earth and the white object on the side of the road a flow in the opposite direction to the camera movement. Meanwhile the motion of the car and bike are different due to their motion relative to the camera. The motion perception will allow the separation of the car and bike objects from occluded scene objects around them (such as the crack in the road). The second test video, figure 4.32 contains more motion variables as the dashboard is completely stationary relative to the camera, the road furniture is moving in a z-axis direction toward the camera and the cars are moving at a similar speed to the dashboard camera, with some lateral motion as well. There are also six erroneous points that are not representative of objects motion (identified by the large optical flow vectors on the left of the image), and can be considered as noise. Because the car is travelling around a bend in the road, the motion vectors of the road furniture on the left are of greater magnitude than that on the right of the frame. The UAV video, figure 4.33, has multiple axis of motion, however in this particular frame it happens that the UAV motion is minimal (hence barely visible motion vectors on the stationary objects). The moving objects to the bottom left have a magnitude that is different because these objects are moving along the road, and thus relatively different to the UAV motion. The full videos with the optical flow of other frames which show the motion of the UAV is available from Morris [96]. An additional frame has been included in these results to show the usual motion vectors seen by the UAV. In figure 4.34, the UAV is moving in a rotational pattern. Thus the motion vectors of stationary objects in each quadrant of the image are different. Objects that are moving do not have the rotational motion vectors, and could be separated out from the other objects. The application of optical flow is not image size dependent, it is number of objects dependent. Thus in a high resolution frame of only a few objects, the performance penalty is the same as a low resolution frame with the same number of objects. In these experiments, it happens that the UAV scene, which is 1920x1080p resolution also has the highest number of objects in it. An estimate for the optical flow penalty per object is 3 ms per object. This is based on a calculation of the time increase of processing a frame with optical flow divided by the number of objects in the scene, and averaged over each result.

4.9.3 Conclusions on Motion Perception

The addition of optical flow to the processing shows motion perception of each object, which will be useful when characterising objects later on. Isolation of static objects and other moving objects can become difficult in rotational scenarios where static objects in different quadrants of the image exhibit different motion directions, figure 4.34. The usage of optical flow for motion perception is suboptimal with the quadrant issues and further work to improve the motion perception could be carried out to solve this problem. One possible area for exploration is the unification the Edge Flow (time domain) and WISE (spatial domain) methods so that the optical flow calculation for motion perception is not necessary or required. This would remove the estimation error factor of optical flow and reduce the processing resources needed. This proposition could aid in solving the motion disparity issue of 2-dimensional cameras. The motion perception as it is, can be used as features for building semantic reasoning models that can characterise and distinguish between the texture patches across the scene however care will need to be taken in complex motion scenarios on the interpretation of motion.

Chapter 5

Comparative Results

The previous chapter analysed the functional performance of Edge Flow and WISE in isolation, with some references to existing techniques for illustration. This chapter provides a comprehensive set of comparisons with existing techniques and the analysis of performance of WISE in the context of what each algorithm sets out to achieve. In some of the comparisons there is a strict detection requirement (object segmentation methods), that have been predefined by the authors of the work. They predominately use some form of training to achieve the object selection. In each case, the data set used in the corresponding work is applied to WISE, such that a representative comparison can be provided with the work presented in their respective papers.

5.1 Edge Detection

5.1.1 Experiments With Different Edge Detection Methods

Fig. 5.1 (a) Original Image (b) Fuzzy Edge Detector [65] (c) ACO edge detector [79] (d) Neural Network edge detector [13] (e) Genetic Algorithm edge detector [17] (f) Universal Gravity edge detector [139]. Images obtained from [1]

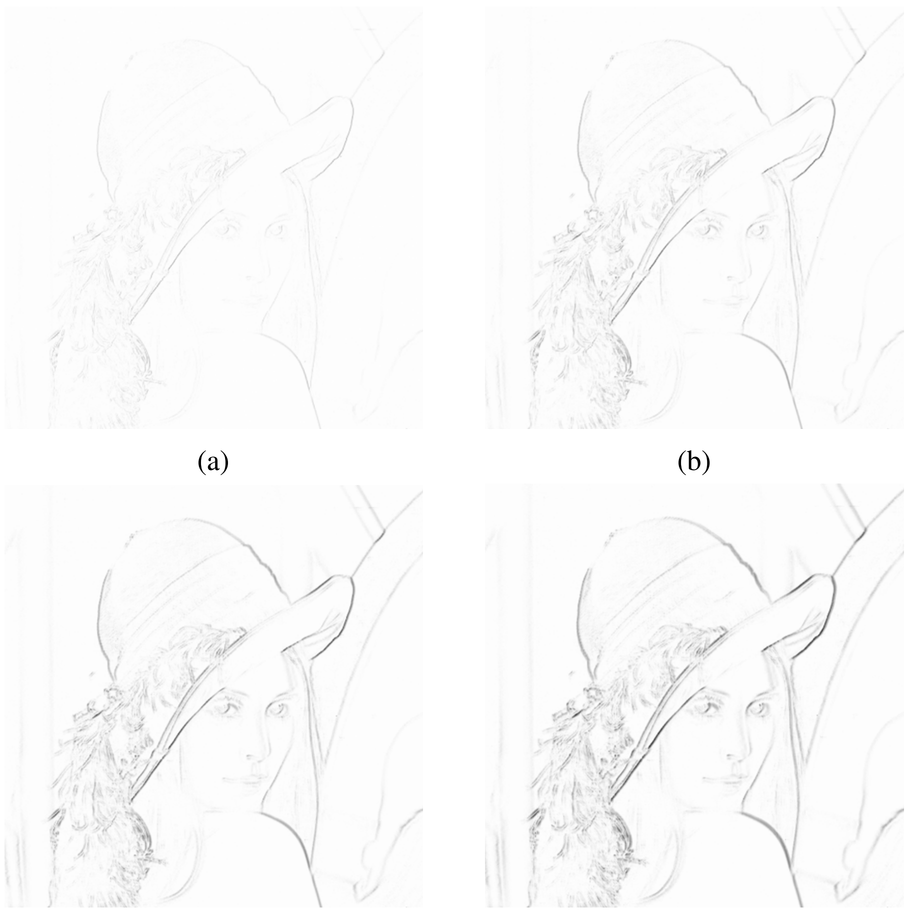


Fig. 5.2 (a) WIDE with window 2, (b) WIDE with window 3, (c) WIDE with window 4, (d) WIDE with window 5

Fig. 5.3 Edge detection results from [33] for fast edge detection using structured forests

(a) (b) (c)

Fig. 5.4 (a) WIDE on people with window of 4, (b) WIDE on barn picture with window of 4, (c) WIDE with window of 6 on coyote picture

5.1.2 Performance Analysis of Edge Detection Results

The overall technique of WISE is designed for object detection. WIDE, a component of WISE, is an edge detection method and forms the basis of object detection (much like the edge

Table 5.1 The performance of different edge detection techniques on the BSD500 [10] data set obtained from [33] and compared with the WIDE technique over 3 different window sizes

Algorithm	ODS	OIS	AP	FPS
Human	.80	.80	-	-
Canny	.60	.64	.56	15
Felz-Hutt [39]	.61	.64	.56	10
Hidayat-Green [53]	.62	-	-	20
BEL [34]	.66	-	-	1/10
gPb + GPU [25]	.70	-	-	1/2
gPb [10]	.71	.74	.65	1/240
gPb-owt-ucm [10]	.73	.76	.73	1/240
Sketch tokens [76]	.73	.75	.78	1
SCG [119]	.74	.76	.77	1/280
SE-SS, T=1 [33]	.72	.74	.77	60
SE-SS, T=4 [33]	.73	.75	.77	30
SE-MS, T=4 [33]	.74	.76	.78	6
WIDE, (a)	.76	-	-	250
WIDE, (b)	.78	-	-	250
WIDE, (c)	.69	-	-	250

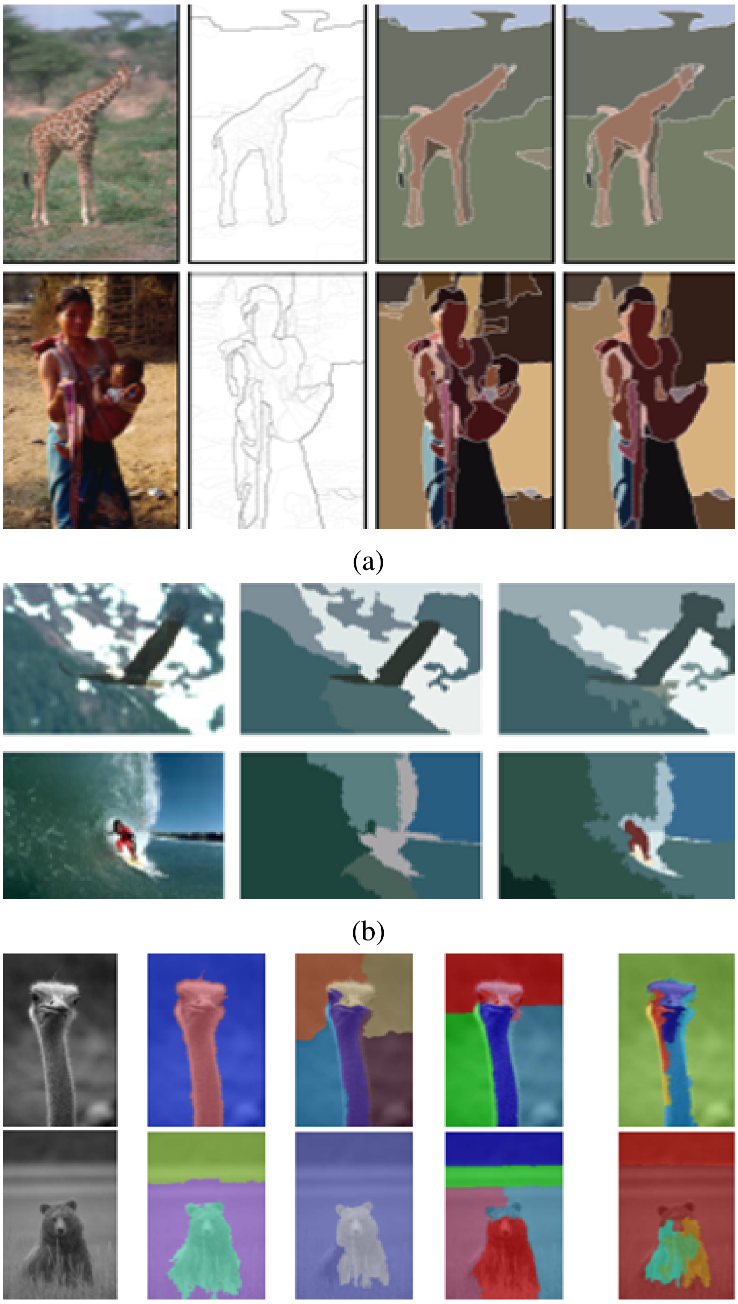
detection in some image segmentation methods [10]). Because of this, a comparison with existing edge detection methods was made. The comparison is over a range of different edge detection methods, from early methods such as shown by Canny [23] to the latest hierarchical learning methods [10]. Figure 5.1 shows the methods when applied to a stationary image obtained from the OpenCV database. The methods presented here apply their edge detection to greyscale images and as such the source image must be converted into greyscale, the other detection methods presented work with contrast and brightness boundaries. In each of these samples, clarity of the object boundaries in the image is lost with the universal gravity detector performing the best in terms of clarity. Figure 5.2 shows the WIDE technique applied to the same image over 2, 3, 4 and 5 window sizes. These sizes were selected to show the progression of edge clarity over several window sizes. The window size of two provides very faint edges which highlight texture edges more than the outline object boundaries (for example the hair is more pronounced than the figure outline). Window sizes of 3 and 4 provide a clearer output of edges with the window 4 giving slightly better defined edges around the hat rim and background textures. A window size of 5 provides the clearest outline of the edges, but the problem of overly thick edges begins to creep in and distorts the clarity of the hair texture due to this over thickness, much like that seen in figure 5.1 where the thickness of edge boundaries interfere with object textures. The edge detection methods shown in figure 5.1 outperform the WIDE method in terms of definition of each edge that is detected. This is because the edge is defined as binary - present or not, where as the WIDE method provides a measure of significance to each edge shown by the varying greyscale output.

A newer edge detection technique, using Hough Forests [33], is shown in figure 5.3. In this figure, obtained from [33], there are three source images, three ground-truth images obtained by manual input by humans, a Sparse Code Gradients (SCG) method developed in [119] and the results for the Hough Forests method. Figure 5.4 shows the results for WIDE applied to the source images. Table 5.1 shows the comparison results of these methods, and a selection of other edge detection methods. The table lists two F-measures (defined in nomenclature) of the results and the average performance (AP). The first F-measure is the optimal dataset scale (ODS) and the second f-measure is the optimum image scale (OIS). A number of the techniques use image scales to aid with edge and boundary definition, similar to the keypoint detection techniques of SIFT [78] or SURF [14] reviewed in chapter 2. WIDE does not use scale space measurements, and thus the result compared to the ground truth is listed in the ODS column only, a separate result for each image; the table results obtained from Dollár and Zitnik [33] show the optimum results from the three images, and the average amalgamates the results from each source. Because there is no scaling in WIDE, it was decided to list each image result separately, given the varying difference between the results for WIDE. In the first two images (left to right) the WIDE technique performs better than the SCG method, and is comparable to the Hough Forests (SE) method at both single scale and multi-scale. The SE method has better definition of the primary edges much like the ground truth image whilst WIDE provides greater detail on the internal textures of the boundaries, at the same time maintaining good discrimination of key edges. The ground-truth images do not take texture edges into consideration and thus the comparison is limited to the primary edges in this case. In the third image, the WIDE technique needed to have an extended window size to produce any meaningful results, and the window size of 6 was arrived at through experimentation (4 did not produce significant boundary edges). Compared to the ground truth and the SCG method, WIDE produces a similar level of result with some extra texture boundary information. SE performs better in this case, with well defined edges around the animals. One possibility of the poorer performance of WIDE in this image is the texture similarity. Given that the method is designed to extract texture differentials, the gradients of the changes across the image will be smaller. Thus the greyscale produces much fainter lines, that whilst not white, are close enough to look like missed edges.

The final column in the table shows the processing time of each method. This is where the WIDE method shows the designed performance characteristics by being significantly faster (more than 30 fps) than any other method tested here. The hardware used for the performance comparison was a 2.6 Ghz Intel i7 using a single processing core. The headroom afforded by the efficient processing allows additional components to be introduced into the processing to extract objects from the scene, as described in chapter 4.

5.2 Image Segmentation

5.2.1 Experiments With Image Segmentation Methods

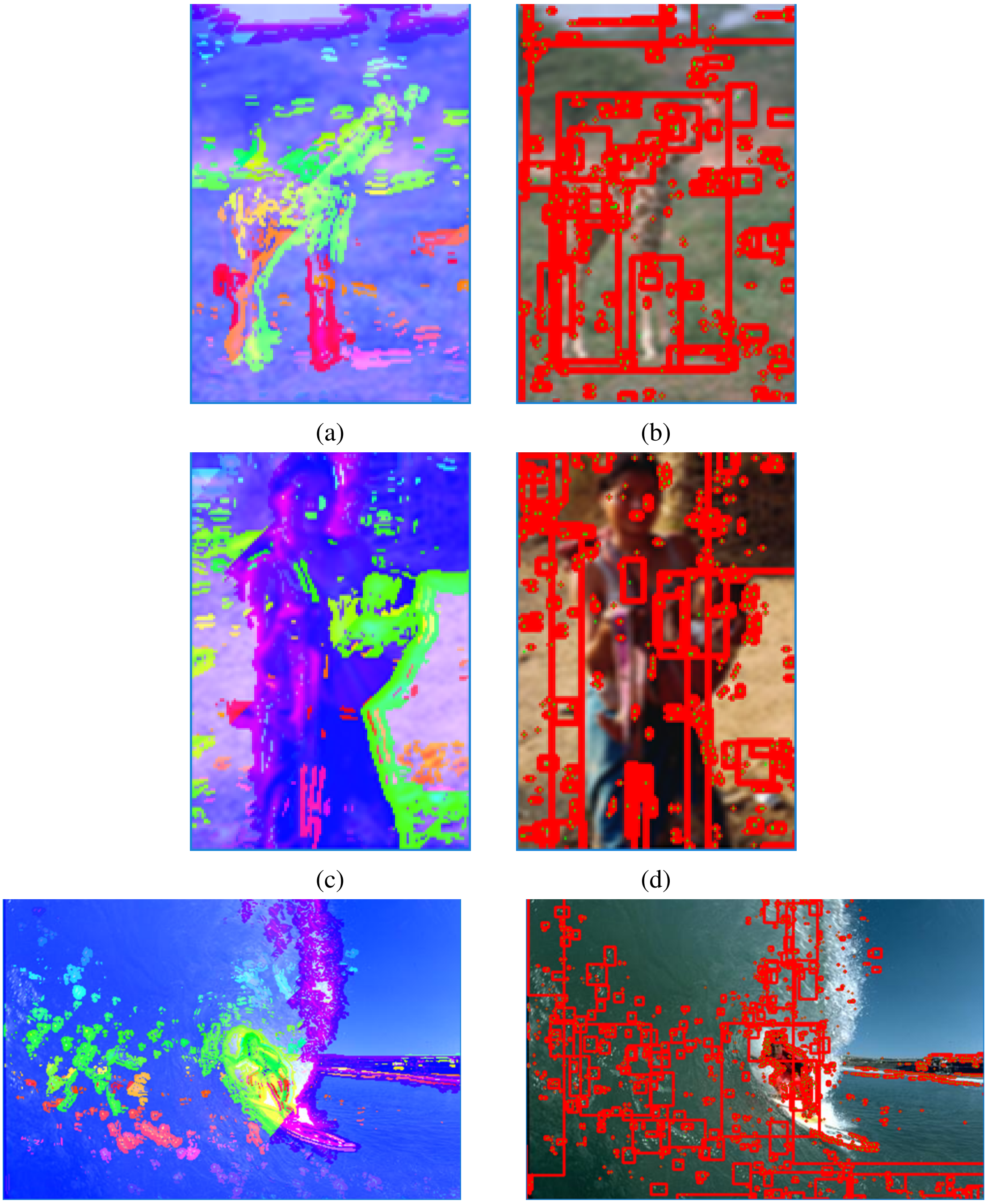


(a)

(b)

(c)

Fig. 5.5 A selection of varying Image Segmentation techniques applied to the BSD database. (a) Ultrametric Contour Map [10], (b) Edison and IHS Image Segmentation methods [158], (c) Bottom Up Aggregation [3], SWA V1 [46], Normalised cuts [84], Mean-shift [27]

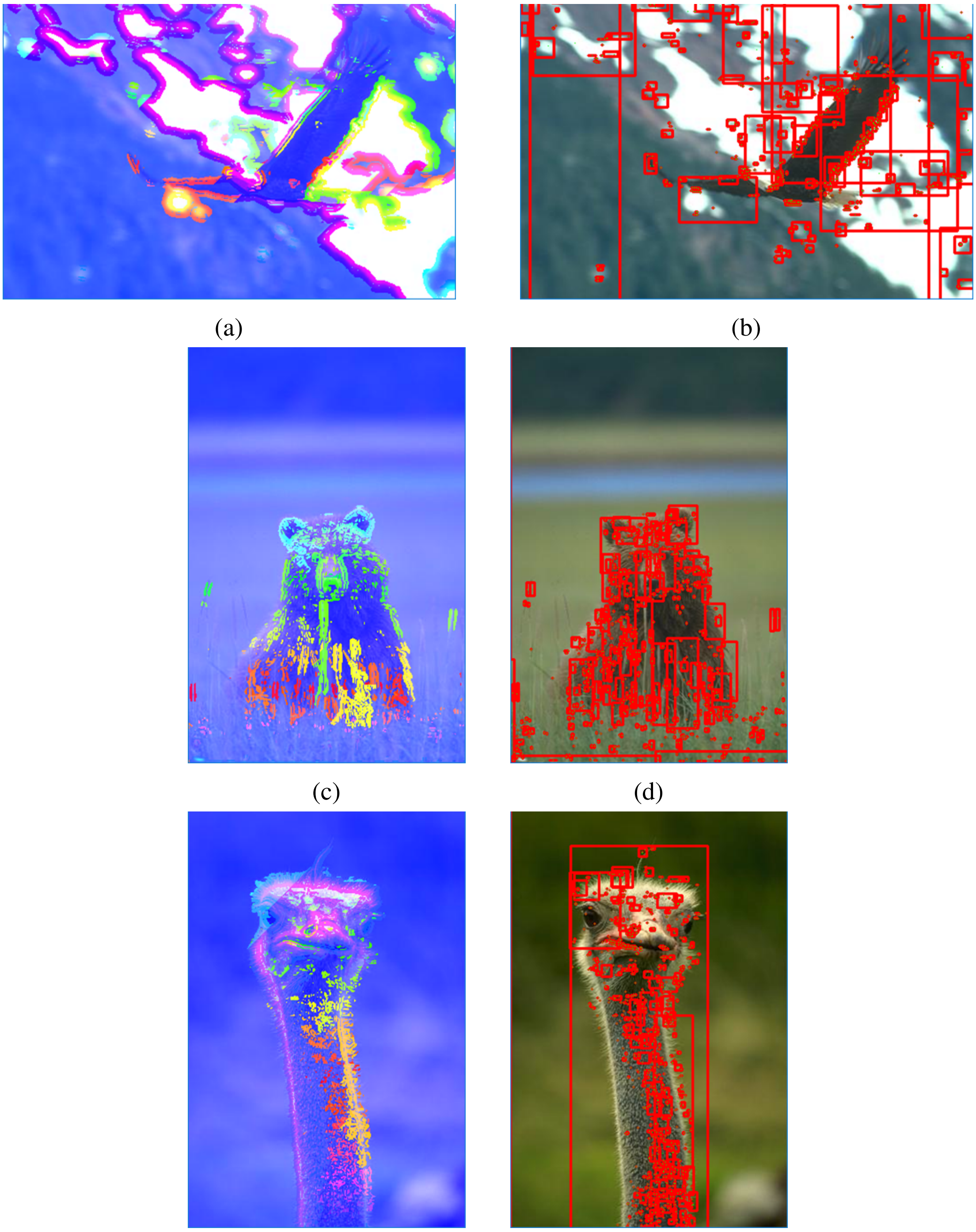


(a) (b)

(c) (d)

(e) (f)

Fig. 5.6 WISE applied to the images used by other techniques. The leftmost image is the actual pixels of the detected texture patches overlaid on the original image. The rightmost image is the individual texture patch boundaries. (a) and (b) Giraffe, (c) and (d) Woman with a baby, (e) and (f) Surfer



(a) (b) (c) (d) (e) (f)

Fig. 5.7 WISE applied to the images used by other techniques. The leftmost image is the actual pixels of the detected texture patches overlaid on the original image. The rightmost image is the individual texture patch boundaries. (a) and (b) Eagle, (c) and (d) Bear, (e) and (f) Ostrich

Table 5.2 The performance of different image segmentation techniques on the BSD500 [10] data set obtained from [10] and compared with the WISE technique over 3 different window sizes

Algorithm	ODS	OIS	Best
Human	.73	.73	-
gPb-owt-ucm [10]	.59	.65	.75
Mean Shift [27]	.54	.58	.66
Felz-Hutt [39]	.51	.58	.68
NCuts [84]	.44	.53	.66
SWA [3]	.47	.55	.66
Total Var [35]	.57	-	-
T+B Encode [117]	.54	-	-
Av. Diss [16]	.47	-	-
ChanVese [16]	.49	-	-
WISE, W=3	.73	-	-
WISE, W=4	.79	-	-
WISE, W=5	.78	-	-

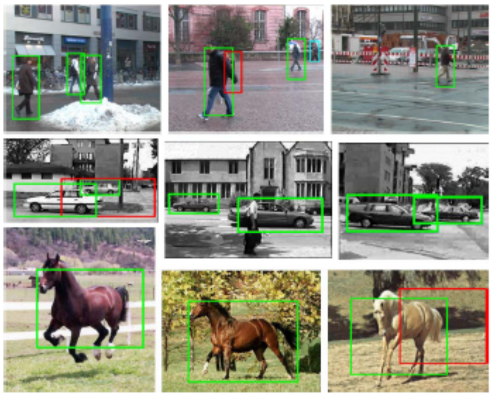
5.2.2 Analysis of the Image Segmentation Methods

Figure 5.5 shows a image results of a range of image segmentation techniques applied to the BSD 500 dataset. Figures 5.6 and 5.7 show the results of WISE applied to the images of the dataset. Each of the image segmentation techniques manages to separate the main textures of each object in the scene, however details on the textures of the objects is lost. In the case of figure 5.5 (a) the woman, baby and giraffe are well segmented as well as some of the objects in the background. However any facial details and background texture details are lost such that earth disturbances are not detected (dark earth to the bottom left of the image of the woman). The Edison method in figure 5.5 (b) does well at segmenting the eagle and the sky with better cloud resolution compared with the IHS technique. The result is different in the surfer picture, with no surfer detection with the Edison method, but a good surfer detection with IHS. This suggests that the Edison method is better at segmenting when the textures of the objects in the image are similar. The bottom-up aggregation method in figure 5.5 (c) performs better than the other comparative image segmentation techniques, with the animals being clearly separated from the background.

When compared with WISE, there are clear differences in what is detected. Similar to the edge detection, the methods of image segmentation provide outlines of the major features but the detail of each object is lost. WISE detects major objects but also separates the major objects into smaller texture patches, such as eyes, hair and other object textures. Furthermore, the technique of WISE does not need a greyscale input image as is the case with some of the techniques presented here (Bottom Up Aggregation, SWA, Normalised Cuts, Mean shift). The table 5.2 shows the empirical comparison of WISE with the methods presented in the images as well as some other older techniques. The results in the table were obtained from Arbeláez et al [10]. Compared to the ground truth, obtained from a human, the techniques do not perform as well apart from UCM in its best possible outcome. WISE on the other hand performs better than both the human and the other image segmentation techniques in all image types. In terms of performance speed, WISE performs at 10 frames per second on a single core of an Intel i7 2.6 Ghz processor. In the literature for the image segmentation techniques presented here, there was no indication of the computational speed, however it is possible to derive that the UCM method (the best of the rest in terms of accuracy) does not perform at this frame rate (the Edge Detection component runs at 0.5 fps - see table 5.1).

5.3 Detection by Classifier

5.3.1 Results of Image Segmentation by Classifier



(a)

(b)

Fig. 5.8 Results from different object detection by classifiers methods, applied to the YouTub-Objects database [113]. (a) Hough Forest Object Detection [45], (b) Fast object segmentation [109]

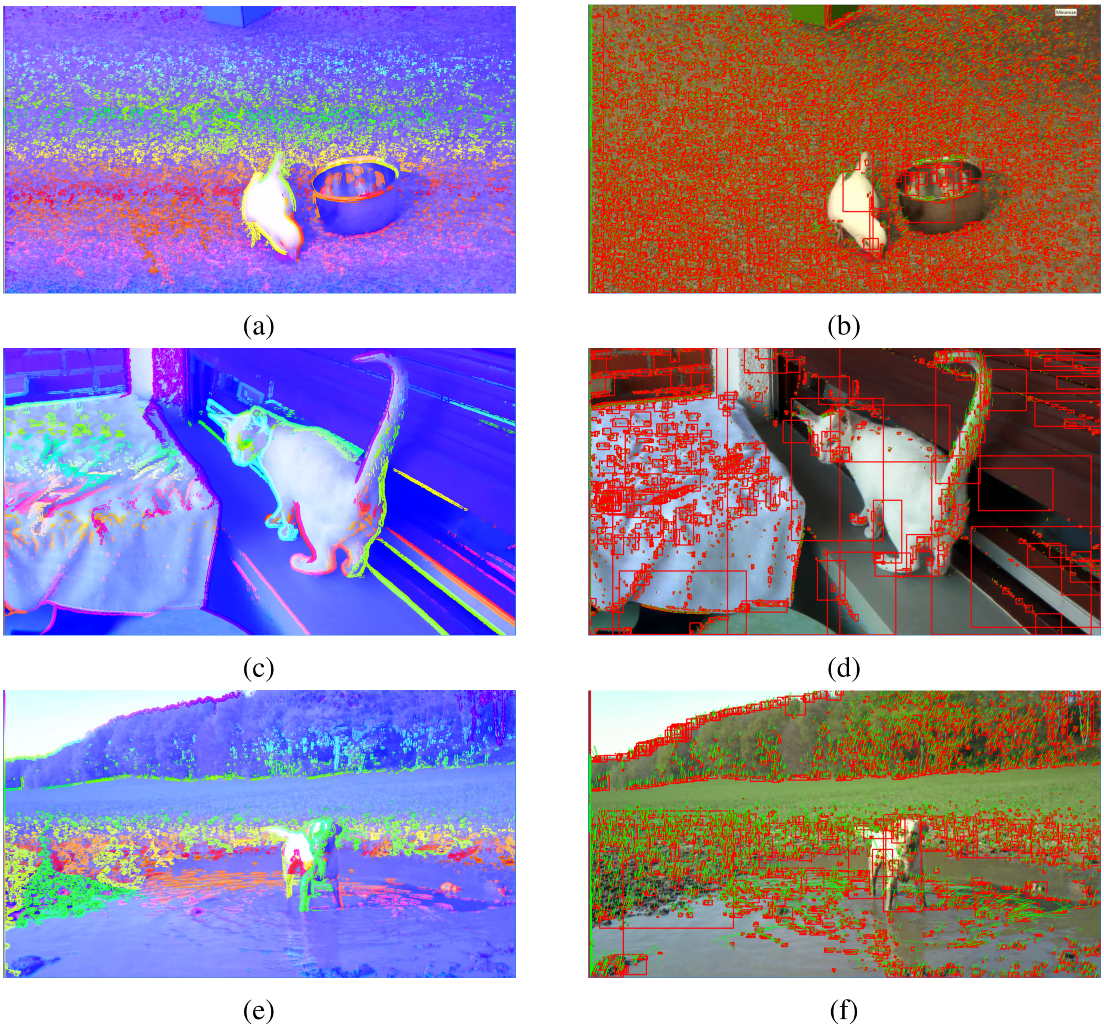


Fig. 5.9 WISE applied to the datasets used by the methods presented above. The leftmost image is the actual pixels of the detected texture patches overlaid on the original image. The rightmost image is the individual texture patch boundaries.

5.3.2 Analysis of Detection By Classifier Methods

The detection by classifier methods are inherently offline methods - training data is required before successful object classification can be made. Therefore the focus of this comparison is more on the accuracy of the detections than speed of processing. The methods shown in figure 5.8 show results where specific objects have been required to be detected. The Hough Forest object detection shows that the classifier can detect specific objects such as people, cars and horses with a low error rate. Three of the frames here show erroneous detections. The fast object segmentation frames also show the capability of separating specific objects from the scene. The accuracy of the methods are compared in table 5.3 and 5.4, with the

Table 5.3 Comparison of the results presented by [45] with WISE over 3 different window sizes

Algorithm	UIUC-Single	UIUC-Multi
Implicit Shape Model [70]	.91	-
ISM+verification [70]	.975	.95
Boundary Shape Model [106]	.85	-
LayoutCRF [154]	.93	-
Mutch and Lowe [100]	.999	.906
Lampert et al. [69]	.985	.986
Hough Forest [45]	.985	.986
HF - Weaker supervision [45]	.944	-
WISE, W=3	.95	-
WISE, W=4	.98	-
WISE, W=5	.98	-

Table 5.4 Comparison of the results presented by [109] with WISE over 3 different window sizes

Algorithm	Average F-measure
Clustering Tracks [109] .347	
Automatic Segment Selection [21]	.267
Fast [113]	.653
WISE, W=3	.73
WISE, W=4	.79
WISE, W=5	.78

former showing the classifier accuracy applied across a series of frames, and the latter the f-measure for the fast object segmentation method - the measures are presented based on the information available in the papers of each method. To compare with classifier accuracy, the WISE method accuracy is determined on whether the object was detected or not - despite other objects being detected by WISE. The results show that WISE consistently detects the desired object at the same or better accuracy rating than Hough Forests, with the added advantage that many other objects are separated and detected in each scene. The accuracy with a smaller window size of 3 drops off, which is likely due to less well defined edge boundaries before the edge linking process (see examples of smaller window size in section 5.1.1. The less than perfect accuracy can be attributed to the over-detection in some frames by the WISE algorithm. For example in the cat video, the example shown shows several texture detections on the cat, but no overall detection of the cat as a whole object. This can be because the smaller detections do not have contiguity of an edge when the edge linking process was carried out, thus forming several smaller texture patches of the main object. The binary classification measure (detected or not) is insufficient to include the detections out with the main object in the image, thus the F-measure is used (as seen in the other comparisons) and this was also used in the fast object segmentation paper [109]. The measure for the fast segmentation method is solely for the moving objects, static objects are not included in the measure. For the WISE results, the measure was extended to all scene objects. When compared with the Fast object detection method, WISE consistently outperforms the method in all video sequences by detecting the moving objects as well as objects in the background. In some frames the green box in the bird scene is a missed detection, which is because there are several surrounding texture patches for the carpet detections. When there are as many detections as there are for the carpet, the edge linking process can link to the incorrect edges isolating some objects from registering as texture patches.

5.4 Conclusions on the Comparisons With WISE

The comparisons shown here demonstrate that WISE is an efficient new technique for object detection, that operates in real-time whilst maintaining accuracy levels comparable to competing techniques. The existing approaches shown here have specific, constrained, detection criteria that limits the flexibility of these techniques in unknown environments. The image segmentation and classifier techniques both need to be trained to detect or segment an image according to predefined criteria. In contrast, WISE does not need training and enables the detection of multiple object types in all scenarios regardless of camera or object motion. Detections by classifier also make the assumption that the objects of interest are moving and do not perform well on objects that are static. Both the Edge Detection and Image Segmentation methods here are applied to single image frames, and motion information on the objects is not retained - this needs to be calculated separately. WISE on the other hand maintains motion information through the usage of optical flow. The edge detection performance of structured forests [33] out performs the WIDE technique in terms of accuracy, however WiDE operates significantly faster in terms of frame rate. In applications where computational speed is not a limiting factor it could be beneficial to explore the usage of structured forests in the edge detection component of WISE. Some limitations of WISE were exposed in the edge linking process where some texture patches were missed, and in some cases the object is divided into several smaller texture patches but not a complete object. The complete object formation from texture patches can be improved through using feature analysis to link the similar and overlapping texture patches together.

Chapter 6

Working with WISE features

A UAV mounted video source conventionally sends the image to a ground based operator who will identify objects of interest and make appropriate decisions. The more information available on each object, the better the decision the operator can make, reducing operator load and increasing system performance. This chapter describes the application of some experimental concepts to the texture patches defined by WISE to better define objects in the scene, and also to impart more information to each object. The methods described here are not fully tested, however some example testing is presented to provide a demonstration of concept. In each case, the focus is on real-time performance and an assumption free methodology.

6.1 Online clustering providing temporal linkage

For any given video sequence, WISE detects objects and the instantaneous velocity of the objects through optical flow (Chapter 4). The limitation is that the link between objects detected in frame m and objects detected in frame n, is the calculated optical flow on the pixels. This means a link between consecutive frames exists but not over a series of frames. If the object between frames was to change size or move in an unexpected way such that optical flow on some points produces errors, the link between objects, and thus contiguity of recognised objects over a series of frames would fail. Linking the motion values over a series of frames will also maintain the contiguity of texture patches forming the objects (see chapter 3), instead of new texture patches having to be linked at each optical flow iteration. The methodology proposed here offers a solution to this problem, by using a clustering technique that evolves as new samples arrive. That is, should the direction and magnitude of the motion from optical flow change, the cluster that currently represents that object will also change to reflect the new spatial and temporal information for the object, yet maintaining the same object (cluster) identification. The limitations of the clustering techniques that operate in an on-line, sequential operation are that they can be order dependant, do not run in real-time, and cannot form clusters of arbitrary shapes (they are always a regular polygon shape). A recently developed method by Hyde and Angelov [57] for clustering, named CEDAS, operates differently and forms arbitrary shaped clusters that update in real-time. The clustering of this method is also not order dependant. Applied to the candidate objects outputted by the WISE method, the on-line clustering technique takes the spatial location and motion magnitude and direction as input features. As the candidate objects move in the scenario, the cluster centres are updated in an on-line manner with the new location and optical flow information. By applying clustering in this context, it will allow tracking of objects through a scene maintaining a cohesive object link between frames.

The concept was tested on a UAV video that produces a lot of WISE objects due to mottled grass surface, and a person running in eccentric patterns. The mottled grass texture also produces some optical flow values due to the vibration and slight motion of the camera on board a test UAV. The video sequence was used because it is a simple scene with one object of interest. The person runs in different motion patterns, in order to show that the clustering of motion and location evolve along with the motion of the person. The results are shown in figures 6.1 and 6.2.

6.1.1 Experimenting With CEDAS Clustering

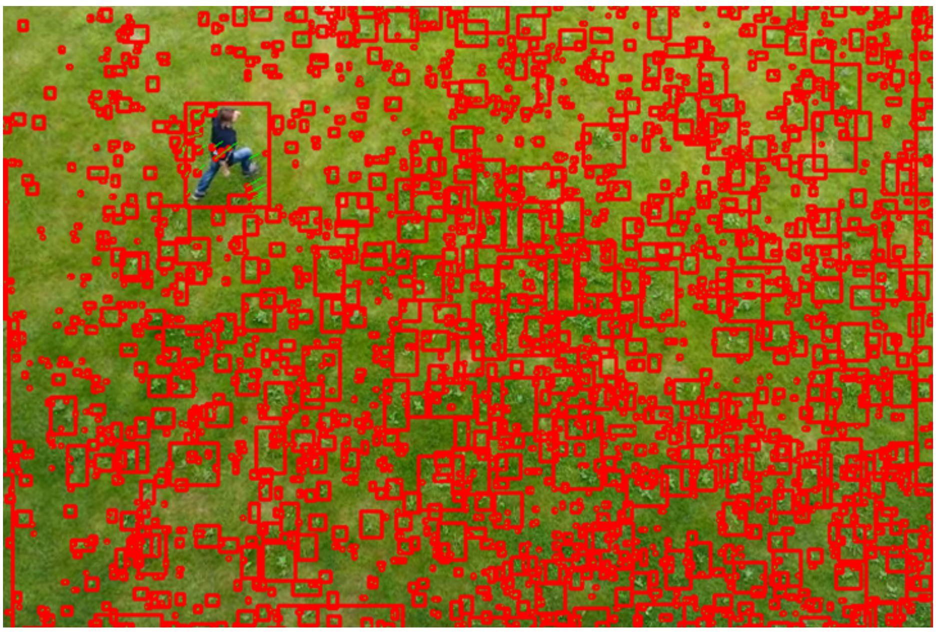


Fig. 6.1 WISE objects detected, with optical flow and object boundaries

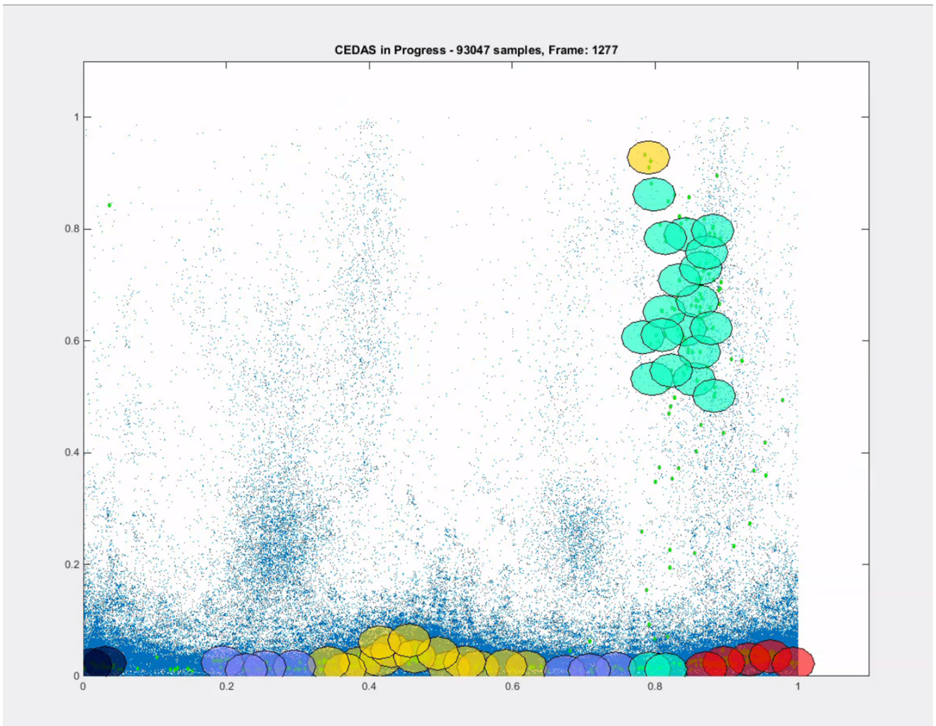
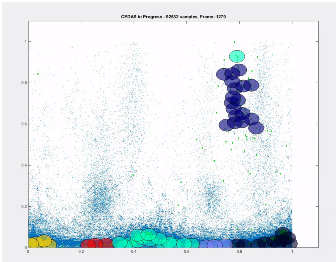
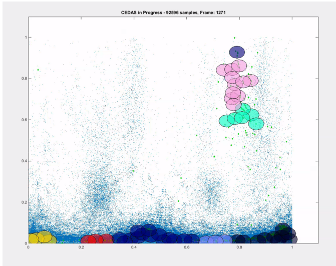
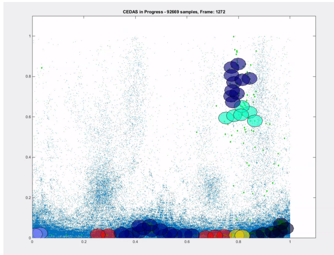
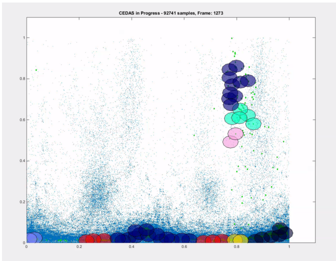


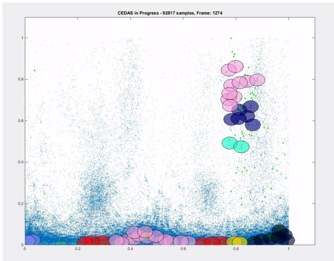
Fig. 6.2 Clustering of the optical flow results using CEDAS. The x-axis shows angle of motion ($-\pi$ to π), and y-axis shows magnitude of motion, normalised between 0 and 1.

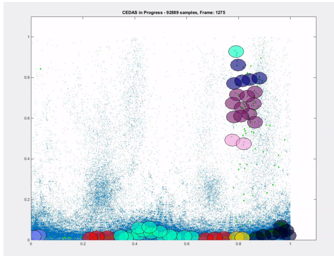

(a) Frame 1270

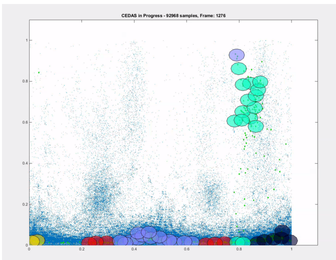

(b) Frame 1271

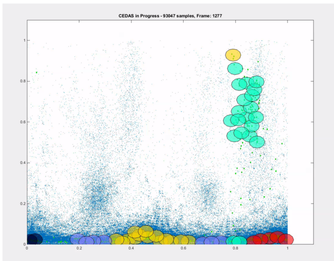

(c) Frame 1272

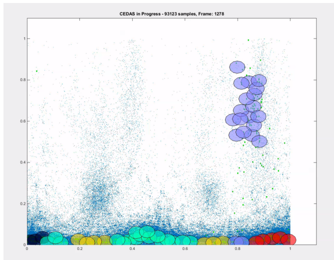

(d) Frame 1273

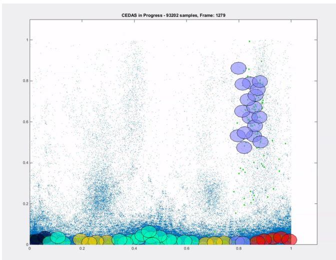

(e) Frame 1274

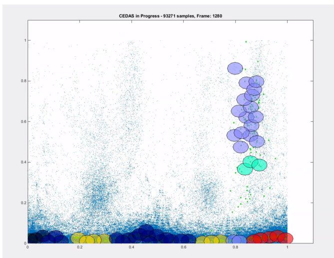

(f) Frame 1275


(g) Frame 1276


(h) Frame 1277


(i) Frame 1278


(j) Frame 1279


(k) Frame 1280

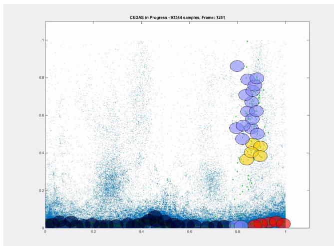

(l) Frame 1281

Fig. 6.3 A series of consecutive frame analysis results from CEDAS, applied to the scene with the person running in circles

6.1.2 Analysis of the CEDAS Results

Figure 6.1 shows the objects detected in the video sequence along with the optical flow applied to each object. There are several detections because the grass has large tufts that are different in texture to the regular grass patches. The person is running in a circle, and generates different motion perception to the stationary objects such as the grass. The clustering method separates the large motion differences from the stationary objects, and creates separate clusters to illustrate this, figure 6.2. This graph shows all the samples over all the frames (dark blue points), green points represent the active samples for the frame, and the coloured circles represent the micro-clusters that make up a cluster (clusters of the same colour are the same cluster). The samples over the entire sequence are shown such that over each of the frames here it can be seen where the samples are overall, and the evolution of clusters as active samples influence the clustering. There are two outlier micro clusters with the moving person, and the stationary grass creates separate clusters along the x-axis. Clusters close to the x-axis are likely to be due to camera shake because the magnitude of motion due to shake is small, but the angle of motion will be in different directions; if it is rotational shake top left objects will appear to move in a different direction to bottom right pixels. There are a number of active samples that are not clustered (green). They are not clustered because the density is insufficient to create a new micro-cluster. A possible explanation is that whilst the person was running quickly (high value on y-axis) the micro clusters were formed, but as the person is slowing down (lower values on the y-axis) new samples are yet to be incorporated until the person remains for a few frames at this slower speed.

6.1.3 Discussing the CEDAS Method

The test results from using CEDAS shows that an adaptive separation of object movement is possible. As the person moves around the scene the clusters update and mostly stay as a cohesive cluster - but updating as the motion of the person changes. This has the benefit of being able to maintain objects as being the same object over a series of frames as opposed to just between two consecutive frames. In this particular test, motion magnitude (y-axis) and motion direction (x-axis) were the features used. If there were two objects moving with the same motion pattern but in different spatial locations, this method would have clustered them together, which is undesirable unless we just want to characterise motion and not separate the objects. By adding spatial location information to the feature set of the clustering it should be possible to separate out the two objects.

6.2 Characterisation of objects

The aim of object characterisation is to apply a type to each object detected, based on its feature set. This is achieved through the use of clustering. The objects that are detected by WISE have a rich feature set that could be used to discriminate and assign a type to each of the objects. A large feature set implies that there can be greater division (number of clusters) between objects, due to the increase in variance higher dimensions bring.

6.2.1 Available Features for Object Characterisation

The output from the WISE algorithm has several different features which provide a detailed description of candidate objects, the internal texture and the perceived motion. There are also features that can be mathematically derived to provide more dimensions to improve object type separation (where needed). The following features have been extracted from the detected objects:

- Length - A pixel-wise measure of the objects length. This is in the 2-dimensional perspective of the camera
- Width - A pixel-wise measure of the objects width. This is in the 2-dimensional perspective of the camera
- Area - A pixel-wise measure of the objects area. This is in the 2-dimensional perspective of the camera
- Number of pixels - The number of pixels that constitute the object.
- Size ratio - The height - width ratio of the object, in the 2 dimensional perspective of the camera. Used in conjunction with motion, this could be used to derive 3-dimensional size.
- Motion magnitude (pixels) - A measure of the optical flow magnitude for the object.
- Motion direction - A 2-dimensional orientation for the optical flow of an object. A 3-dimensional orientation could be achieved through the use of homography, similar to that in 3
- Mean edge gradient - the mean gradient value constituting the perimeter of the object
- Standard deviation of edge gradient - the standard deviation of the perimeter gradient of the object
- Mean colour (RGB) - the mean colour of the pixels constituting the object
- Standard deviation colour (RGB) - the standard deviation of the pixels constituting the object

- Spatial location (x, y) - the location of the object in the frame. This location is the centre of the object, defined by the intersection of diagonal lines from the maximum and minimum x,y coordinates of the object.

6.2.2 Clustering of features

In order to establish separation between object types, some method of clustering the object features is required. If all the features are clustered it is likely that every object that has been detected will be determined as a different type (similar objects will likely have a different background). This means the features being clustered need to be pre-selected based on some characterisation requirement based on operator or user interest. For example, a user may be looking for small regular sized objects and thus the clustering could be performed on the length, width, area and size ratio features. The clustering method used needs to be real-time, and parameter free such that the selection of cluster parameters does not influence the characterisation. An example of undesirable parameter selection could be cluster radius or number of expected clusters because the spread of the data or the number of object types in a scene is unknown. This removes some clustering techniques from consideration such as k-means and fuzzy c-means which both use parameter selection to define the number of clusters expected. Subtractive clustering and similar derivatives are not real-time and also use a cluster radius parameter. Evolving c-means clustering is data order dependant which is undesirable in an unknown environment, and mean-shift clustering is also dependant on known data. The data is clustered on a frame by frame basis, so the technique used does not have to be adaptive or on-line; all the data samples that need clustering will be available at the time of clustering. On-line clustering methods typically require samples to arrive sequentially, and with several objects and pixels to analyse, this can slow the processing down. Therefore a clustering method that processes the entire frame of samples as a batch is required. A new density based clustering technique named DDC is capable of clustering in real-time and does so in a batch manner. It requires an initial parameter of cluster radius but the radius adapts based on the data distribution, and therefore this initial parameter is not as limiting as previously suggested.

6.2.3 Test videos

The results here are testing the capability of DDC applied to some of the features of objects extracted by WISE.

Helicopter Video

The helicopter police chase video was used as it is a fairly simple scenario with little object interference or complexity. It also has a large variation of sizes in objects in order to test the characterisation using size based features. Additionally this video sequence has been used in previous tests and will provide some level of comparison with previous steps in the chain.

Fig. 6.4 Helicopter police chase video

UAV Video This video is a UAV video with multiple axis of motion. This video has an extremely small moving aircraft, along with other some moving objects originating from the ground such as smoke and cars. This video was used to test the discrimination in a complex moving environment, with objects of interest that are extremely difficult detect. The original image is in greyscale, and this will also test the performance of the WISE and characterisation algorithms on non-RGB frames.

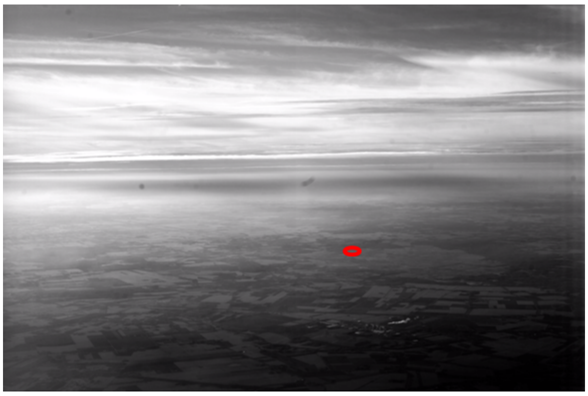
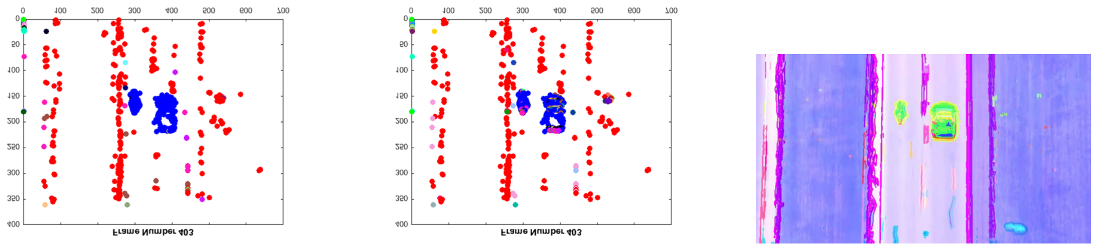


Fig. 6.5 UAV flight video

6.3 Results of Object Characterisation

6.3.1 Clustering on Helicopter video



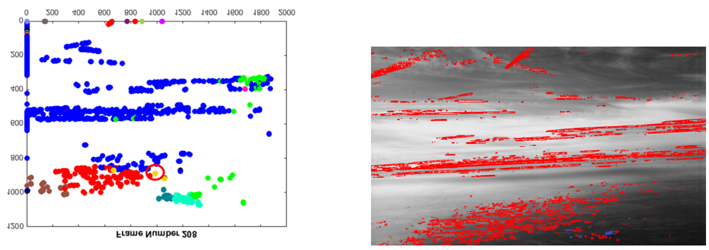
(a) Clustering of objects based on motion

(b) Clustering of objects based on dimensions and size

(c) Overlay of clustering when both motion and dimension features are used

Fig. 6.6 Clustering of objects in the helicopter video, resulting in the characterisation of different object types.

6.3.2 Clustering on UAV video



(a) Clustering of the UAV video based on motion features

(b) Clustering of motion features overlayed on the orignal image

Fig. 6.7 Clustering of objects in the UAV video based on motion features

6.4 Analysis of Object Characterisation Results

In figure 6.6 there are three separate outputs showing the clustering results in two different feature sets. The first two images are both cluster plots obtained from Matlab, whilst the third is an overlay image of cluster results from the second feature set. Figure 6.6(a), shows the clustering on perceived motion magnitude and direction of the objects. In this result, the car and bike are characterised as the same type, as they are the moving objects, and the majority of the other objects detected are clustered separately. There are also some outliers shown, indicating some kind of motion separation. Figure 6.6(b) shows the clustering results using motion features and the length, width and size ratio of the objects. The results show that some increased object separation is achieved, with the white object on the side of the road characterised differently than when just motion features are applied. Some of the road cracks are also defined separately and more clearly. The samples on the Matlab figures are individual objects, such that the size of each object and their bounds is not clear. As seen in the WISE edge linking phase, the car and bike consist of several separate objects that are defined by the textures of each surface (the car has a white roof and dark bonnet) Figure 6.6(c) shows the result of this clustering clearer with the colour overlay on the original image, so that the extent of the characterisation can be seen overlayed onto visual objects. In this image, the long lines that are the road verge, the markings and long road cracks are characterised as the same type of object. The white object and disturbed earth are typed differently and the smaller cracks on the road are also separate from the larger, longer cracks. Both the car and the motorbike are typed the same, different to the rest of the surroundings. The second video sequence is limited in the number of permitted frames, figure 6.7. The clustering is applied to motion features only because in this scenario, there is one small moving object of interest with the remainder being object detections of scene objects inherent to the WISE technique. The Matlab plot, figure 6.7(a) shows several different motion types. The motion type of the object of interest has been circled (orange cluster). Because the motion of the UAV (and hence the camera), is in all 8 degrees of freedom for three-dimensional movement, when the camera rotates or pans with lateral motion, many of the objects in the scenario are moving in the same real direction but in a different relative direction. A filter is applied to compensate for this differential in relative motion, and the results of the remaining motion are shown in figure 6.7(b), where the results have been overlayed onto the original frame, with the objects separated by motion highlighted. The result in this case, shows that there are a couple of moving road objects, the moving object of interest, and detects the motion of the smoke emanating from the chimneys.

6.5 Discussing Object Characterisation

The additional clusters seen in figure 6.6 that do not represent moving objects yet are clustered as moving separately to the other stationary objects can be explained by three dimensional camera movement. For the most part, the camera motion appears to be translational, but there are some small deviations in the rotational and z-axis planes. With objects being in different locations, the direction and magnitude of the motion of each object will vary when the camera movement is outside translational movement. Some objects will appear to move differently to the other stationary background, but the motion is the relative differential perceived by the camera when moving in a three dimensional plane. Some of this incorrect motion perception is carried over to the second feature set that also uses object size features. The only difference between the two cluster groups is the white object on the side of the road, meaning the disturbed earth and the smaller cracks in the road appear as separate object characterisations because of the camera motion differential, not as a result of sufficient size deviation. The filter to counteract the motion differential in the UAV scenario helps to remove this misconception of motion due to camera movement, and can be explained using an example. In a simple rotational motion the objects in the top left have similar magnitude but different relative rotation (in a 360 degree sense) to objects in the other corners. The filter used reduces the direction range to a single quartile (90 degrees). This is achieved by inverting directions in the opposite quartile (flipping), and offsetting the directions in the adjacent quartiles by either adding or subtracting 90 degrees depending on the quartile. This does not compensate for all variations in three dimensional movements, but it does allow the filtering of rotational variances based on location from the frame centre. Figure 6.7(b) shows an overlay of the UAV frame, with the direction filtering applied to the clusters. The frame only shows objects identified to have different-to-stationary motions, and is not showing clustering differences between these other types. That is because when clustering is applied after the direction filtering, even a slight deviation of an object from another object can cause a separate cluster. This may be useful in some scenarios but it was considered to confuse the point the frame overlay is making.

Chapter 7

Conclusions and Future Work

7.1 Summary of the research

The research developed a real-time detection algorithm in a moving camera environment capable of feature rich object analysis and identification. The direction of the research ended up focussing on the development of the novelty detection algorithms Edge flow and WISE. The result of the research is real-time novelty detection algorithms that detect both static and moving objects in moving or static camera scenarios. As demonstrated in chapter 5, WISE is capable of real-time performance operating in the region of ten frames per second, without the need for a GPU or brute force processing. Due to the lengthy investigation work into the novelty detection aspect, limited progress was made in object identification and analysis. Built into the WISE method is the output of rich features that describe the characteristics of the objects, such as texture gradient, object composition, size and ratio, along with 2D image based velocity components.

7.2 Addressing the Research Questions

The research initially explored existing techniques to understand if extending the algorithms can help in answering the research questions.

7.2.1 Experimenting with existing work

Experimentation with existing work showed that there was a trade-off with speed and accuracy. It also laid the foundation work to adapt the techniques such that assumptions about detections were not made (RDE greyscale and WIDE). It was found that each method individually had limitations, and could not definitively answer any of the research questions. The outcomes of the existing methods, and experimentation with them, showed that extending them or adjusting the processing methods had limited effect on improving accuracy or processing time. The existing research could not be speeded up sufficiently to not need brute force processing at higher resolutions, nor could it detect static or moving objects.

7.2.2 Framework review

The review and experimentation with some of the existing methods highlighted a that the computer vision framework was linear, and prone to errors by passing noise to the next level of processing (detection to identification for example). The identification of multiple components led to an analysis of the computer vision framework with the aim of better inter-component data passing such that any assumptions made at a detection phase are not carried through permanently to the semantic reasoning components. A cyclic framework was proposed such that feedback from higher level semantic reasoning components can automate parameter value selection to optimise application specific performance.

7.2.3 Edge Flow and WISE

For temporal based methods, the limitations were that they required too much computational power (optical flow for example) or they could only detect objects that were moving. The system would fail as soon as the objects were motionless. For image segmentation based methods they were able to derive objects from static frames, without motion information. In all cases assumptions were being made about the scenarios before any detections were made. In an unknown scenario, this can lead to the removal of information that would otherwise have been useful (the example of the cordon tape blowing in the wind in chapter 3). The algorithms developed in this research was addressed the research questions by not making assumptions on the scene detection, providing real-time performance that scales well with resolution changes, able to detect both static and moving objects in any sequence of frames regardless of the camera motion and combined the best features of the existing background subtraction and image segmentation techniques (contour detection, pixel-wise density estimation, and efficient use of optical flow). These enhancements to computer vision allow the algorithms to run on low power systems and also the minimises of assumptions made at the detection phase. The concepts of Edge Flow and WISE are a new way of addressing computer vision detection problems. One of the most important steps with the Edge Flow and WISE is that they do not discriminate between detections. The decision tree of "interesting" objects or detections has been removed somewhat from the lower level detection phase and enables higher level semantic reasoning sections to decide if a detection is interesting or uninteresting. The other advantage of the new methodology is that it lends itself to adoption of the feedback mechanism proposed in chapter 3. Each component can be fed input from a higher level to adjust the detection parameters. For example the window size of WiDE can be changed if the feature extraction layer requires more granular detail of the texture changes, or the cluster membership values changed to isolate a subset of objects based on their features. One of the limitations with the research is the visualisation of the detections. It is difficult to show the edge linked boundaries and the arbitrary shapes of the candidate objects in one cohesive image which is why there are different representations presented through this work.

7.3 Performance Achievements

Chapter 1 and 2 illustrated the various angles the research could focus on. Each objective that was detailed also had a performance indicator associated with it. This section assesses the performance of the research compared with the objectives highlighted in chapter 2.

7.3.1 Detection of stationary objects

The first objective set out for the work was to develop a reliable novelty detector that detects both static and dynamic objects within a scene. In addition there was a requirement to perform the analysis in real-time within an envelope of 10ms per frame on a 2 megapixel image. This is so that object detection algorithms built on the novelty detection have sufficient available processing time such that the overall result remains real-time. By developing the WIDE approach in chapter 3, the novelty (object boundary) detection capability was able to meet each of the criteria. The hardest part was to detect camouflaged object boundaries. In the comparison chapter 5, the scene with the two coyotes demonstrated that by adjusting the parameters of WIDE, camouflaged object boundaries can be detected by this method. During the comparison phase, it was found that there are some better object boundary detectors, such as the Hough Forests, in terms of overall accuracy and detail. However, to attain this level of detail significant processing time had to be sacrificed. The level of accuracy achieved by WIDE is only slightly less than that of the Hough Forests whilst maintaining the faster-than 10 ms processing time per frame.

7.3.2 Novelty detection without image stitching

The second objective was to extend the concept of detection of stationary objects to the moving camera domain. When image stitching is used to create an overlapped region errors are introduced. The aim was to continuously analyse the frames of a scene, maintaining relative motion information, without stitching or manipulating the scene such that unwanted noise was introduced. The complete WISE algorithm successfully avoids image manipulation whilst detecting both static and moving objects irrespective of camera motion. The WISE algorithm also operates in real-time, capable of processing frames faster than 100ms for a 2 mega-pixel frame. One current difficulty with the WISE method is that it is parametrized, so to detect specific novelties for an application the parameters need to be adjusted. The parameter design is for future development of feedback from semantic reasoning components such that the detection regime is adaptable to unknown environments.

7.3.3 Rich feature extraction

The specific region analysis goal was replaced with the extraction of a rich-feature set. The specific region analysis would have focused on analysing small regions of the image that contained an object of interest. This was more applicable to methods that are high in processor usage such that reducing the processing area improves their speed. Edge Flow and WISE do not need this reduction in processing area to maintain real-time processing, so the focus switched to obtaining a rich feature set on the global scene (not narrowing focus, which would potentially remove points of interest unintentionally) The new objective was designed to promote the development of an algorithm that extracts a wealth of features such that object analysis and semantic reasoning can be performed at a later stage. Direct features such as size and movement of objects was extracted as well as derived features such as edge gradient profile and mean and variance of the boundary density change. As the object characterisation phase showed, these features can be used to characterise and separate out the objects detected by the WISE algorithm. There is no additional processing required for the direct features, and minimal processing for the derived features meaning that the rich feature set has been extracted within the processing envelope of WISE. Chapter 5 showed the detection improvement of WISE over other techniques. The rich feature set provides another advantage that WISE has over competing techniques as the extraction of this feature set is part of the algorithm.

7.3.4 Object detection accounting for occlusion

By utilising WISE which includes frame analysis and optical flow to give object motion perception, separation of objects that are moving in different directions has been possible. Occlusion occurs when two or more objects cross paths because they are on a different trajectory to others. Examples in chapter 4 and 5 show the separation of objects despite crossing one another. In the motorbike and car video, the cracks on the road are maintained as separate objects despite the car and bike passing over them frequently. This is in contrast to existing techniques which group occluded objects together.

7.3.5 Characterisation of objects

The characterisation of objects is a way of separating each object detected by WISE by their features. Chapter 6 describes the utilisation of clustering which groups features together into different object types. The utilisation of CEDAS clustering allowed the separation of a person moving from the minor movements of grass blades due to camera shake. The clustering process also separated out arms of the person moving if they were moving differently to the person's body. The prototyping done in this area showed the capability of the output of the WISE algorithm to detect and separate objects by type. Further work and experimentation is needed to find out the limitations of the object characterisation, and the robustness in a wide range of scenarios.

7.3.6 Classification

This is an element that will form part of the future development of the project. At present, a selection of behavioural assessments are made with the rich feature set (chapter 6) through separating object types. However there is no determination on *what* the objects are. The rich feature set retained in the output from WISE should be sufficient to enable a start on developing an object classification capability.

7.3.7 Computational performance

The theme throughout the work has focussed on the constraints provided by the UAV application area. These were assumption free, detection of objects in unknown environments without assuming specific movement patterns or texture make up of the objects of interest. The performance envelope of the work is real-time - processing the same or faster of the rate at which frames are received; without the need for a GPU brute-force approach. The work also has aspects of autonomy maintained, with the only input needed being initial parameters to specify the edge linking tolerance. This is usually application specific and does not need constant adaptation.

7.4 The research applied in the UAV context

The established path of the research was informed by a set of constraints from the context of Unmanned Aerial Vehicles (UAVs) such as low power and a moving camera. In chapter 1 the intended objectives and workload were outlined. The algorithms that this work has yielded paves the way for the development of application specific improvements to the following areas of UAV operation:

- Reduction of bandwidth

  Less volume of video data returned by the UAV to the ground station

  Lower data size of the images returned in the data stream.

  Send information, not data, back to the ground station.

- Increased image analysis performance and capability

  Analysis of high resolution imagery

  Fast accurate detection of novelties

  Tracking of objects across imagery

  Real-time online analysis of the video data (links in with reducing data volume).

- Reduced Operator Load

  The imagery sent back to the operator needs to be pre-processed and have novelties or objects identified to limit analysis required by operator.

  Autonomous online identification and classification of interesting objects / novelties so that constant supervised input by operators is not necessary is a key requirement.

7.5 With reference to the Hypotheses

The original hypotheses stated:

1. By combining the benefits of image segmentation with background subtraction, a solution that is capable of detecting static and moving objects in real-time should be possible.

2. Removing assumptions on detections will mean that the algorithms will detect all object transitions in an image. It is therefore reasonable to predict that the algorithm will be able to operate irrespective of the camera motion.

3. Once the objects are detected in an image, with the number of features available, it should be possible to type each object based on their features (cluster each object). The type association may not correlate with human differentiation of objects due to the underlying features that are being clustered.

4. The algorithms should allow for a feedback mechanism such that a higher semantic reasoning section can adapt or tune a previous layer based on detection and identification objectives.

The hypotheses were based on knowledge of the field, life experience and with the knowledge that nature has had a robust vision system for animals in place for Aeons. Some of the hypotheses were able to be answered completely by the research whilst some were only partially shown. The algorithm separately detects objects, and then restores motion to each object to provide temporal information and achieves this in a real-time manner. The realisation of a visual feedback method was not able to be shown, however a framework has been laid out for future work to explore the possibility. The ability to clearly distinguish between types of objects that have been detected was partially realised through the research shown in 6, where the clustering of object features separates the object types. Additionally the CEDAS algorithm is able to track the changes of the object features whilst maintaining an appreciation of the objects themselves such that should the features of an object change sufficiently over time, the object is still identified as the same object as previous merely with a different feature set.

7.6 Further work

The exploration of a model of the human vision system has led to a new technique that operates in real-time and detects objects in a video sequence without the assumption of any object features. There are many desirable extensions to the work that were out with the time scale of the project. A real-world velocity calculation capability was discussed using the 3D affine transform of optical flow, and also scope for relative trajectories - even with a 2D frame (having the degrees of freedom for each object could allow 3D inference of the environment). There is also further validation work to be done with the object characterisation and behaviour analysis. The usage of online clustering for the behaviour analysis also lends itself to tracking objects through a scene. Given the feature set and the characterisation, the online clustering method could be used to track not just objects, but also the evolution of objects as their characteristics change in the environment - for example texture changes based on changes in illumination, or changes in object dimensions like the unfurling of a missile platform. The research was able to establish a framework for the future development of a feedback system which would remove the need for manual setting of the WISE parameters. The idea is that the parameters can be autonomously tuned based on the criteria or requirements of another module - feature extraction for example. The input criteria might be, for example, to refine the detections to a person with a red jacket. The system could have an autonomous component that took this feedback information and adapted the parameters of the algorithms in an autonomous fashion. There is also scope for further research into the human eye replication concept. The methodology followed in this thesis was based on work that theorises about how the eye system works. The theories have allowed for a level of replication in the computer vision world, however there may be more definitive ways to demonstrate that the separation of object detection and motion is the correct theory to apply for human replication. In association with this, one area the research did not explore was the effect of shadows in a scene. The human vision system is able to detect objects, and motion, and also separate out shadows from actual objects. This provides for challenging future questions on how to handle shadows, and how the human system manages to cope with them. Is it a simple case of learned experience such that the system ignores shadows when one is perceived? Or is there a component within the vision system that specifically handles shadows?

References

[1] Amir Abarghouei, Afshin Ghanizadeh, and Siti Mariyam Shamsuddin. Advances of Soft Computing Methods in Edge Detection. *International Journal of Advances in Soft Computing and its Applications*, 1(2):162–203, 2009.

[2] A Alahi, R Ortiz, and P Vandergheynst. FREAK: Fast Retina Keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 510–517, 2012. ISBN 1063-6919. doi: 10.1109/CVPR.2012.6247715.

[3] Sharon Alpert, Meirav Galun, Achi Brandt, and Ronen Basri. Image segmentation by probabilistic bottom-up aggregation and cue integration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(2):315–327, 2012. ISSN 01628828. doi: 10.1109/TPAMI.2011.130.

[4] P Angelov and X Zhou. Evolving Fuzzy-Rule Based Classifiers From Data Streams. *IEEE Transactions on Fuzzy Systems*, 16(6):1462–1474, 2008. ISSN 1063-6706. doi: 10.1109/TFUZZ.2008.925904.

[5] Plamen Angelov. *Autonomous learning systems: from data streams to knowledge in real-time / Plamen Angelov*. John Wiley & Sons Ltd., 2012.

[6] Plamen Angelov, Ramin Ramezani, and Xiaowei Zhou. Autonomous novelty detection and object tracking in video streams using evolving clustering and Takagi-Sugeno type neuro-fuzzy system. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1456–1463, 2008. ISBN 9781424418213. doi: 10.1109/IJCNN.2008.4633989.

[7] Plamen Angelov, Pouria Sadeghi-Tehran, and Ramin Ramezani. An approach to automatic real-time novelty detection, object identification, and tracking in video streams based on recursive density estimation and evolving Takagi-Sugeno fuzzy systems. *International Journal of Intelligent Systems*, 26(3):189–205, 2011. doi: 10.1002/int.20462.

[8] Plamen Angelov, Pouria Sadeghi-Tehran, and Christopher Clarke. AURORA: autonomous real-time on-board video analytics, 2016. ISSN 09410643.

[9] Sebastian Anthony. DARPA shows off 1.8-gigapixel surveillance drone, can spot a terrorist from 20,000 feet. *Extreme Tech*, page 1, 2013. URL https://www.extremetech.com/extreme/146909-darpa-shows-off-1-8-gigapixel-surveillance-drone-can-spot-a-terrorist-from-20000-feet.

[10] Pablo Arbeláez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 2011. ISSN 01628828. doi: 10.1109/TPAMI.2010.161.

[11] Christian Balkenius. Natural Intelligence For Autonomous Agents. *Halmstad University*, pages 181–196, 1994.

[12] O Barnich and M Van Droogenbroeck. ViBE: A powerful random technique to estimate the background in video sequences. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 945–948, 2009. ISBN 1520-6149. doi: 10.1109/ICASSP.2009.4959741.

[13] Alper Baştürk and Enis Günay. Efficient edge detection in digital images using a cellular neural network optimized by differential evolution algorithm. *Expert Systems with Applications*, 36(2 PART 2):2645–2650, 2009. ISSN 09574174. doi: 10.1016/j.eswa.2008.01.082.

[14] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. doi: http://dx.doi.org/10.1016/j.cviu.2007.09.014. URL http://www.sciencedirect.com/science/article/pii/S1077314207001555.

[15] Jr Bergen, P Anandan, K Hanna, and R Hingorani. Hierarchical model-based motion estimation. *Computer Vision ECCV 1992*, 588:237–252, 1992. doi: 10.1007/3-540-55426-2_27. URL http://www.springerlink.com/index/FM3672W48668LT05.pdf$\delimiter"026E30F$nhttp://link.springer.com/chapter/10.1007/3-540-55426-2{_}27.

[16] Luca Bertelli, Baris Sumengen, B. S. Manjunath, and Frédéric Gibou. A variational framework for multiregion pairwise-similarity-based image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(8):1400–1414, 2008. ISSN 01628828. doi: 10.1109/TPAMI.2007.70785.

[17] Suchendra M. Bhandarkar, Yiqing Zhang, and Walter D. Potter. An edge detection technique using genetic algorithm-based optimization. *Pattern Recognition*, 27(9):1159–1180, 1994. ISSN 00313203. doi: 10.1016/0031-3203(94)90003-5.

[18] Irving Biederman. Recognition by components: A theory of human image understanding. *Psychological Review*, 94(2):115–117, 1987. ISSN 0033-295X. doi: 10.1037/0033-295X.94.2.115.

[19] J Bigun and G H Granlund. Optimal Orientation Detection of Linear Symmetry. *Proceedings of the IEEE First International Conference on Computer Vision*, 54:433–438, 1987.

[20] Zachary Davies Boren. There are officially more mobile devices than people in the world. *The Independent*, page 1, 2014. URL http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html.

[21] Thomas Brox and Jitendra Malik. Object Segmentation by Long Term Analysis of Pont Trajectories. *Eccv 2010*, pages 282–295, 2010. ISSN 03029743. doi: 10.1007/978-3-642-15555-0_21.

[22] U Büttner and A Straube. Ego- and object-motion perception: Where does it take place? *Behavioral and Brain Sciences*, 17(02):316–317, 1994.

[23] J Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851.

[24] V Caselles, L Garrido, and L Igual. A Contrast Invariant Approach to Motion Estimation. *Scale Space and PDE Methods in Computer Vision SE - 21*, 3459:242–253, 2005. doi: 10.1007/11408031_21.

[25] B Catanzaro, B Y Su, N Sundaram, Y Lee, M Murphy, and K Keutzer. Efficient, high-quality image contour detection. *2009 IEEE 12th International Conference on Computer Vision*, pages 2381–2388, 2009. ISSN 1550-5499. doi: 10.1109/ICCV.2009.5459410.

[26] Q. Chen, L. Zhao, J. Lu, G. Kuang, N. Wang, and Y. Jiang. Modified two-dimensional Otsu image segmentation algorithm and fast realisation. *IET Image Processing*, 6(4): 426, 2012. ISSN 17519659. doi: 10.1049/iet-ipr.2010.0078.

[27] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5): 603–619, 2002. ISSN 01628828. doi: 10.1109/34.1000236.

[28] Nancy J. Cooke, Jamie Christopher Gorman, Jasmine L. Duran, and Amanda R. Taylor. Team cognition in experienced command-and-control teams. *Journal of Experimental Psychology: Applied*, 13(3):146–57, 2007. ISSN 1076-898X. doi: 10.1037/1076-898X.13.3.146. URL http://www.ncbi.nlm.nih.gov/pubmed/17924800.

[29] Marco Cristani and Vittorio Murino. A spatial sampling mechanism for effective background subtraction. In Alpesh Ranchordas, Helder Araújo, and Jordi Vitrià, editors, *VISAPP (2)*, pages 403–412. INSTICC - Institute for Systems and Technologies of Information, Control and Communication, 2007. ISBN 978-972-8865-74-0. URL http://dblp.uni-trier.de/db/conf/visapp/visapp2007-2.html{#}CristaniM07.

[30] John G Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of the Optical Society of America. A, Optics and image science*, 2(7):1160–1169, 1985. ISSN 1084-7529. doi: 10.1364/JOSAA.2.001160.

[31] Glade David. Unmanned Aerial Vehicles: Implications for Military Operations. *Occasional Paper, Center for Strategy and Technology*, 1(16):1–27, 2000.

[32] Torgeir Dingsoyr, Sridhar Nerur, Venugopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6):1213–1221, 2012. ISSN 01641212. doi: 10.1016/j.jss.2012.02.033.

[33] Piotr Dollár and C. Lawrence Zitnick. Fast Edge Detection Using Structured Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37 (8):1558–1570, 2014. ISSN 0162-8828. doi: 10.1109/TPAMI.2014.2377715. URL http://arxiv.org/abs/1406.5549$\delimiter"026E30F$nhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6975234.

[34] Piotr Dollár, Zhuowen Tu, and Serge Belongie. Supervised learning of edges and object boundaries. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1964–1971, 2006. ISBN 0769525970. doi: 10.1109/CVPR.2006.298.

[35] Michael Donoser, Martin Urschler, Martin Hirzer, and Horst Bischof. Saliency driven total variation segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 817–824, 2009. ISBN 9781424444205. doi: 10.1109/ICCV.2009.5459296.

[36] Elan Dubrofsky. Homography Estimation. *Optical Engineering*, 15:977, 2009. ISSN 0091-3286. doi: 10.1117/1.3364071. URL http://link.aip.org/link/OPEGAR/v49/i3/p037202/s1{&}Agg=doi.

[37] A Elgammal, R Duraiswami, D Harwood, and L S Davis. Background and foreground modeling using nonparametric kernel density estimation for visual surveillance. *Proceedings of the IEEE*, 90(7):1151–1163, 2002. doi: 10.1109/JPROC.2002.801448.

[38] J. Elston, E. Frew, and B. Argrow. Networked UAV command, control and communication. In *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2006*, volume 5, pages 3357–3365, 2006. ISBN 1563478196 | 9781563478192.

[39] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004. ISSN 09205691. doi: 10.1023/B:VISI.0000022288.19776.77.

[40] Rob Fergus, Yair Weiss, and Antonio Torralba. Semi-Supervised Learning in Gigantic Image Collections. *Nips*, pages 522–530, 2009. URL http://cs.nyu.edu/{~}fergus/pmwiki/pmwiki.php?n=PmWiki.Publications???

[41] Martin A. Fischler and Robert C. Bolles. Random sample consensus. *Communications of the ACM*, 24(6):381–395, 1981. ISSN 00010782. doi: 10.1145/358669.358692. URL http://dl.acm.org/citation.cfm?id=358669.358692.

[42] D. V. Forrest. Understanding Intelligence. *American Journal of Psychiatry*, 157(12): 2074–2075, 2000. ISSN 0002953X. doi: 10.1176/appi.ajp.157.12.2074.

[43] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. Probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8(3):325–344, 2000. ISSN 09295593. doi: 10.1023/A:1008937911390.

[44] William T. Freeman and Craig D. Weissman. Television Control by Hand Gestures. *Proceedings of the International Workshop on Automatic Face and Gesture Recognition*, pages 179–183, 1995. doi: 10.1.1.152.8786. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.152.8786{&}rep=rep1{&}type=pdf.

[45] Juergen Gall and Victor Lempitsky. Class-specific hough forests for object detection. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, pages 1022–1029, 2009. ISBN 9781424439935. doi: 10.1109/CVPRW.2009.5206740.

[46] Meirav Galun and Eitan Sharon. Texture segmentation by multiscale aggregation of filter responses and shape elements. *Proceedings Ninth IEEE International Conference on Computer Vision*, M(Iccv):716–723 vol.1, 2003. doi: 10.1109/ICCV.2003.1238418. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1238418.

[47] D M Gavrila. Pedestrian Detection from a Moving Vehicle. *ECCV European Conference on Computer Vision*, 2:37–49, 2000. ISSN 16113349. doi: 10.1007/3-540-45053-X-3. URL http://www.springerlink.com/index/TCLR58FUL1FUP0LL.pdf.

[48] Stephen Gould, Tianshi Gao, and Daphne Koller. Region-based segmentation and object detection. *Advances in Neural Information Processing Systems (NIPS)*, 22:1–9, 2009. ISSN <null>.

[49] Shweta Gupte, Paul Infant Teenu Mohandas, and James M. Conrad. A survey of quadrotor unmanned aerial vehicles. In *Conference Proceedings - IEEE SOUTHEASTCON*, 2012. ISBN 9781467313742. doi: 10.1109/SECon.2012.6196930.

[50] Isabelle Guyon and André Elisseeff. *An Introduction to Feature Extraction*, volume 207. 2006. ISBN 978-3-540-35487-1. doi: 10.1007/978-3-540-35488-8. URL http://www.springerlink.com/content/j847w74269401u31/$\delimiter"026E30F$nhttp://link.springer.com/10.1007/978-3-540-35488-8.

[51] Chris Harris and Mike Stephens. A Combined Corner and Edge Detector. *Procedings of the Alvey Vision Conference 1988*, pages 147–151, 1988. ISSN 09639292. doi: 10.5244/C.2.23. URL http://www.bmva.org/bmvc/1988/avc-88-023.html.

[52] M Heikkila and M Pietikainen. A texture-based method for modeling the background and detecting moving objects. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):657–662, 2006. doi: 10.1109/TPAMI.2006.68.

[53] Ray Hidayat and Richard Green. Real-time texture boundary detection from ridges in the standard deviation space. *Procedings of the British Machine Vision Conference 2009*, pages 5.1–5.10, 2009. doi: 10.5244/C.23.5. URL http://www.bmva.org/bmvc/2009/Papers/Paper182/Paper182.html.

[54] Derek Hoiem, Alexei a. Efros, and Martial Hebert. Putting objects in perspective. *International Journal of Computer Vision*, 80(1):3–15, 2008. ISSN 09205691. doi: 10.1007/s11263-008-0137-5.

[55] Berthold K Horn and Brian G Schunck. Determining Optical Flow. *Artificial Intelligence*, 17:185–203, 1981.

[56] Berthold K P Horn. Robot Vision. *Robot Vision*, 4931:509, 1986. doi: 10.1007/978-3-540-78157-8. URL http://www.amazon.com/dp/0262081598.

[57] Richard Hyde and Plamen Angelov. A fully autonomous Data Density based Clustering technique. In *IEEE SSCI 2014 - 2014 IEEE Symposium Series on Computational Intelligence - EALS 2014: 2014 IEEE Symposium on Evolving and Autonomous Learning Systems, Proceedings*, pages 116–123, 2014. ISBN 9781479944958. doi: 10.1109/EALS.2014.7009512.

[58] Ramesh Jain, Rangachar Kasturi, and Brian G. Schunck. Machine Vision. In *Image Processing*, pages 2,12–16,. 1995. ISBN 978-0070320185. URL http://www.cse.usf.edu/{~}r1k/MachineVisionBook/MachineVision.pdf.

[59] O. Javed, K. Shafique, and M. Shah. A hierarchical approach to robust background subtraction using color and gradient information. In *Proceedings - Workshop on Motion and Video Computing, MOTION 2002*, pages 22–27, 2002. ISBN 0769518605. doi: 10.1109/MOTION.2002.1182209.

[60] Jianbo Shi and C. Tomasi. Good features to track. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, pages 593–600, 1994. ISBN 0-8186-5825-8. doi: 10.1109/CVPR.1994.323794. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=323794.

[61] P; Bowden KaewTraKulPong R. An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection. In *2nd European Workshop on Advanced Video Based Surveillance Systems*, volume 1, 2001.

[62] Yan Ke Yan Ke and R. Sukthankar. PCA-SIFT: a more distinctive representation for local image descriptors. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, 2:2–9, 2004. ISSN 1063-6919. doi: 10.1109/CVPR.2004.1315206.

[63] Hina Keval and Martina Angela Sasse. “Not the Usual Suspects”: A study of factors reducing the effectiveness of CCTV. *Security Journal*, 23(2):134–154, 2010. ISSN 0955-1662. doi: 10.1057/palgrave.sj.8350092. URL http://link.springer.com/10.1057/palgrave.sj.8350092.

[64] R. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6):1153–1160, 1981. ISSN 0096-3518. doi: 10.1109/TASSP.1981.1163711.

[65] Dong Su Kim, Wang Heon Lee, and In So Kweon. Automatic edge detection using 3 ?? 3 ideal binary pixel patterns and fuzzy-based edge thresholding. *Pattern Recognition Letters*, 25(1):101–106, 2004. ISSN 01678655. doi: 10.1016/j.patrec.2003.09.010.

[66] Laszlo B. Kish. End of Moore’s law: Thermal (noise) death of integration in micro and nano electronics. *Physics Letters, Section A: General, Atomic and Solid State Physics*, 305(3-4):144–149, 2002. ISSN 03759601. doi: 10.1016/S0375-9601(02)01365-8.

[67] Hans Knutsson. Representing local structure using tensors. *Proceedings of 6th Scandinavian Conference on Image Analysis*, pages 244–251, 1989. doi: 10.1007/978-3-642-21227-7_51.

[68] WengKin Lai and ImranM. Khan. An Improved Particle Swarm Optimisation for Image Segmentation of Homogeneous Images. *PRICAI 2012: Trends in Artificial Intelligence SE - 21*, 7458:218–228, 2012. doi: 10.1007/978-3-642-32695-0_21.

[69] Christoph H. Lampert, Matthew B. Blaschko, and Thomas Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008. ISBN 9781424422432. doi: 10.1109/CVPR.2008.4587586.

[70] Bastian Leibe, Aleš Leonardis, and Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *International Journal of Computer Vision*, 77(1-3):259–289, 2008. ISSN 09205691. doi: 10.1007/s11263-007-0095-3.

[71] Victor Lempitsky, Andrew Blake, and Carsten Rother. Image Segmentation by Branch-and-Mincut. *Computer Vision – ECCV 2008*, 5305:15–29, 2008. ISSN 03029743. doi: 10.1007/978-3-540-88693-8_2.

[72] Brian Leninger, Jonathan Edwards, John ANTONIADES, David Chester, Dan Haas, Eric LIU, Mark STEVENS, Charlie Gershfield, Mike BRAUN, James D. TARGOVE, Steve Wein, Paul BREWER, Donald G. MADDEN, Khurram Hassan Shafique, and Brian LEININGER. Autonomous Real-time Ground Ubiquitous Surveillance : Imaging System (ARGUS-IS). In *Proceedings of SPIE, the International Society for Optical Engineering*, pages 69810H.1–69810H.11, 2008. ISBN 978-0-8194-7172-7. doi: 10.1117/12.784724. URL http://cat.inist.fr/?aModele=afficheN{&}cpsidt=20931895$\delimiter"026E30F$nhttp://www.darpa.mil/uploadedImages/Content/Our{_}Work/I2O/Programs/Autonomous{_}Real-time{_}Ground/Images/full/ARGUS{_}Mission14{_}Poster{_}12-06-0920(rev1)[1].JPG.

[73] Vincent Lepetit and Pascal Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006. ISSN 01628828. doi: 10.1109/TPAMI.2006.188.

[74] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. BRISK: Binary Robust invariant scalable keypoints. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2548–2555, 2011. ISBN 9781457711015. doi: 10.1109/ICCV.2011.6126542.

[75] Yonghong Li and Curt H. Davis. Pixel-based invariant feature extraction and its application to radiometric co-registration for multi-temporal high-resolution satellite imagery. In *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, volume 4, pages 348–360, 2011. ISBN 1939-1404. doi: 10.1109/JSTARS.2010.2062490.

[76] Joseph J. Lim, C. Lawrence Zitnick, and Piotr Dollar. Sketch tokens: A learned mid-level representation for contour and object detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3158–3165, 2013. ISBN 978-0-7695-4989-7. doi: 10.1109/CVPR.2013.406.

[77] Yang Liu, Rong Feng, and Hong Zhang. Keypoint matching by outlier pruning with consensus constraint. In *Proceedings - IEEE International Conference on Robotics*

and Automation, volume 2015-June, pages 5481–5486, 2015. ISBN 9781479969227. doi: 10.1109/ICRA.2015.7139965.

[78] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. ISSN 09205691. doi: 10.1023/B:VISI.0000029664.99615.94.

[79] De-Sian Lu and Chien-Chang Chen. Edge detection improvement by ant colony optimization. *Pattern Recognition Letters*, 29(4):416–425, 2008. ISSN 01678655. doi: 10.1016/j.patrec.2007.10.021.

[80] Zhong-Lin Lu and George Sperling. The functional architecture of human visual motion perception. *Vision Research*, 35(19):2697–2722, 1995. doi: http://dx.doi.org/10.1016/0042-6989(95)00025-U. URL http://www.sciencedirect.com/science/article/pii/004269899500025U.

[81] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. *IJCAI*, 81:674–679, 1981.

[82] Yvonne Hsu Lin Luo and Lyndon da Cruz. The Argus® II Retinal Prosthesis System, 2016. ISSN 18731635.

[83] C.A. Mack. Fifty Years of Moore's Law. *IEEE Transactions on Semiconductor Manufacturing*, 24(2):202 – 207, 2011. ISSN 0894-6507. doi: 10.1109/TSM.2010.2096437. URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=5696765.

[84] Jitendra Malik, Serge Belongie, Thomas K. Leung, and Jianbo Shi. Contour and Texture Analysis for Image Segmentation. *International Journal of Computer Vision*, 43(1):7–27, 2001. ISSN 0920-5691. doi: 10.1023/A:1011174803800. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.1476.

[85] S. Marčelja. Mathematical description of the responses of simple cortical cells*. *Journal of the Optical Society of America*, 70(11):1297, 1980. ISSN 0030-3941. doi: 10.1364/JOSA.70.001297. URL http://www.osapublishing.org/abstract.cfm?uri=josa-70-11-1297$\delimiter"026E30F$nhttps://www.osapublishing.org/josa/abstract.cfm?uri=josa-70-11-1297.

[86] D Marr and E Hildreth. Theory of Edge Detection. *Proceedings of the Royal Society of London B: Biological Sciences*, 207(1167):187–217, 1980. ISSN 0080-4649. doi: 10.1098/rspb.1980.0020. URL http://rspb.royalsocietypublishing.org/content/207/1167/187.

[87] Matthew I. McCartney, Saleh Zein Sabato, and Mohan Malkani. Image registration for sequence of visual images captured by UAV. In *2009 IEEE Symposium Computational Intelligence for Multimedia Signal and Vision Processing, CIMSVP 2009 - Proceedings*, pages 91–97, 2009. ISBN 9781424427710. doi: 10.1109/CIMSVP.2009.4925653.

[88] R. Mehrotra, K. R. Namuduri, and N. Ranganathan. Gabor filter-based edge detection. *Pattern Recognition*, 25(12):1479–1494, 1992. ISSN 00313203. doi: 10.1016/0031-3203(92)90121-X.

[89] Ajmal S Mian, Mohammed Bennamoun, Robyn Owens, A S Mian, M Bennamoun, · R Owens, and R Owens. Keypoint Detection and Local Feature Matching for Textured 3D Face Recognition. *Int J Comput Vis*, 79:1–12, 2008. ISSN 0920-5691. doi: 10.1007/s11263-007-0085-5.

[90] Richa Mishra. Fingerprint Recognition using Robust Local Features. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(6): 2277–128, 2012.

[91] B B Misra and S Dehuri. Functional link artificial neural network for classification task in data mining. *Journal of Computer Science*, 3(12):948, 2007.

[92] Nabil Mohammed, Ali Munassar, and A Govardhan. A Comparison Between Five Models Of Software Engineering. *International Journal of Computer Science*, 7(5): 94–101, 2010. ISSN 09736107. doi: 10.1.1.403.3201.

[93] Gordon E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998. ISSN 00189219. doi: 10.1109/JPROC.1998.658762.

[94] Hans Moravec. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, volume 2, pages 584–584, 1977. doi: 10.1007/SpringerReference_40189. URL http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:TOWARDS+AUTOMATIC+VISUAL+OBSTACLE+AVOIDANCE{#}0.

[95] Hans Peter Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. *tech. report CMU-RI-TR-80-03*, page 175, 1980. doi: ADA092604. URL https://www.ri.cmu.edu/publication{_}view.html?pub{_}id=22.

[96] Gruffydd Morris. Dashboard video sequence, 2014. URL https://www.youtube.com/watch?v=e0U95V3ydmM.

[97] Gruffydd Morris and Plamen Angelov. Real-time novelty detection in video using background subtraction techniques: State of the art a practical review. *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014-Janua(January): 537–543, 2014. ISSN 1062922X. doi: 10.1109/SMC.2014.6973963. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84938058193{&}partnerID=tZOtx3y1.

[98] Gruffydd Morris and Plamen Angelov. Edge Flow. *Systems, Man and Cybernetics (SMC), 2015 IEEE International Conference on*, 1(1):200–208, 2015.

[99] Philip Morrison, John Billingham, and John Wolfe. The search for extraterrestrial intelligence-SETI, 1979. ISSN 00945765.

[100] Jim Mutch and David G. Lowe. Multiclass object recognition with sparse, localized features. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 11–18, 2006. ISBN 0769525970. doi: 10.1109/CVPR.2006.200.

[101] R Muthukrishnan and M Radha. Edge Detection Techniques for Image Segmentation. *International Journal of Computer Science & Information Technology*, 3(6):259–267, 2011. doi: 10.5121/ijcsit.2011.3620.

[102] H. P. Narkhede. Review of Image Segmentation Techniques. *International Journal of Science and Modern Engineering*, pages 2319–6386, 2013.

[103] Kenzo Nonami, Farid Kendoul, Satoshi Suzuki, Wei Wang, and Daisuke Nakazawa. *Autonomous flying robots: Unmanned aerial vehicles and micro aerial vehicles*. 2010. ISBN 9784431538554. doi: 10.1007/978-4-431-53856-1.

[104] W.D. Nordhaus. The progress of computing. *papers.ssrn.com*, pages 1–45, 2001. URL http://papers.ssrn.com/sol3/papers.cfm?abstract{_}id=285168.

[105] P Noriega and O Bernier. Real Time Illumination Invariant Background Subtraction using Local Kernel Histograms. In M; Fisher Chantler B; Trucco, M;, editor, *British Machine Conference*, pages 100.1–100.10, 2006.

[106] Andreas Opelt, Axel Pinz, and Andrew Zisserman. Learning an alphabet of shape and appearance for multi-class object detection. *International Journal of Computer Vision*, 80(1):16–44, 2008. ISSN 09205691. doi: 10.1007/s11263-008-0139-3.

[107] Ahmed A. Othman, Hamid R. Tizhoosh, and Farzad Khalvati. EFIS-evolving fuzzy image segmentation. *IEEE Transactions on Fuzzy Systems*, 22(1):72–82, 2014. ISSN 10636706. doi: 10.1109/TFUZZ.2013.2246761.

[108] Paritosh Pandya. Principles of concurrent and distributed programming, 1991. ISSN 01676423.

[109] Anestis Papazoglou and Vittorio Ferrari. Fast object segmentation in unconstrained video. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1777–1784, 2013. ISBN 9781479928392. doi: 10.1109/ICCV.2013.223.

[110] Karl Pauwels, Matteo Tomasi, Javier Díaz Alonso, Eduardo Ros, and Marc M. Van Hulle. A Comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features. *IEEE Transactions on Computers*, 61(7):999–1012, 2012. ISSN 00189340. doi: 10.1109/TC.2011.120.

[111] M Piccardi. Background subtraction techniques: a review. *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, 4:3099–3104, 2004. doi: 10.1109/ICSMC.2004.1400815.

[112] J Ponce and D Forsyth. *Computer vision: a modern approach*. 2012. ISBN 9780136085928. doi: 10.1016/j.cbi.2010.05.017. URL http://www.inria.fr/centre/paris-rocquencourt/actualites/computer-vision-a-modern-approach.

[113] Alessandro Prest, Christian Leistner, Javier Civera, Cordelia Schmid, and Vittorio Ferrari. Learning object class detectors from weakly annotated video. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3282–3289, 2012. ISBN 9781467312264. doi: 10.1109/CVPR.2012.6248065.

[114] Jms Prewitt. Object enhancement and extraction. *Picture processing and Psychopictorics*, 10(1):15–19, 1970.

[115] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv*, pages 1–15, 2015. ISSN 0004-6361. doi: 10.1051/0004-6361/201527329. URL http://arxiv.org/abs/1511.06434.

[116] R Ramezani, P Angelov, and Z Xiaowei. A fast approach to novelty detection in video streams using recursive density estimation. In *Intelligent Systems, 2008. IS '08. 4th International IEEE Conference*, volume 2, pages 14–17, 2008. doi: 10.1109/IS.2008.4670523.

[117] Shankar R. Rao, Hossein Mobahi, Allen Y. Yang, S. Shankar Sastry, and Yi Ma. Natural image segmentation with adaptive texture and boundary encoding. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5994 LNCS, pages 135–146, 2010. ISBN 3642123066. doi: 10.1007/978-3-642-12307-8_13.

[118] Pudsey Rc. AEE SD23 Magicam FPV Cam on Skywalker FPV, 2013. URL https://www.youtube.com/watch?v=GV6FPU0U4oo.

[119] Xiaofeng Ren and Deva Ramanan. Histograms of sparse codes for object detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3246–3253, 2013. ISBN 9781424442966. doi: 10.1109/CVPR.2013.417.

[120] Ronald A Rensink. CHANGE DETECTION. *Annual Review of Psychology*, 53(1): 245–277, 2002. doi: 10.1146/annurev.psych.53.100901.135125. URL http://dx.doi.org/10.1146/annurev.psych.53.100901.135125.

[121] Lawrence GIlman Roberts. Machine perception of three-dimensional solids. In *PhD Thesis*, pages 159–197. 1965. ISBN 0-8240-4427-4. URL http://oai.dtic.mil/oai/oai?verb=getRecord{&}metadataPrefix=html{&}identifier=AD0413529.

[122] AZRIEL ROSENFELD. *Readings in Computer Vision*. 1987. ISBN 9780080515816. doi: 10.1016/B978-0-08-051581-6.50006-4. URL http://www.sciencedirect.com/science/article/pii/B9780080515816500064.

[123] Richard M Russell. The CRAY-1 Computer System. *Proc. Comm. ACM Computer Proc. WJCC Comm. ACM McCarthy, J. Time Sharing Computer Systems Pt. I, AFIPS Press N.J. N.J. N.J*, 36(12):657–675, 1971. ISSN 00010782. doi: 10.1145/359327.359336.

[124] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2010. ISBN 0137903952. doi: 10.1017/S0269888900007724. URL http://amazon.de/o/ASIN/0130803022/.

[125] P Sadeghi-Tehran and P Angelov. A real-time approach for novelty detection and trajectories analysis for anomaly recognition in video surveillance systems. In *Evolving and Adaptive Intelligent Systems (EAIS), 2012 IEEE Conference on*, pages 108–113, 2012. doi: 10.1109/EAIS.2012.6232814.

[126] P Sadeghi-Tehran and P Angelov. ARTOD: Autonomous Real Time Objects Detection by a Moving Camera using Recursive Density Estimation. *Novel Applications of Intelligent Systems*, 586(1), 2014.

[127] Georgia Sandbach, Stefanos Zafeiriou, Maja Pantic, and Lijun Yin. Static and dynamic 3D facial expression recognition: A comprehensive survey. *Image and Vision Computing*, 30(10):683–697, 2012. ISSN 02628856. doi: 10.1016/j.imavis.2012.06.005.

[128] Rob Schneiderman. Unmanned drones are flying high in the military/aerospace sector [Special Report]. *IEEE Signal Processing Magazine*, 29(1):8–11, 2012. ISSN 10535888. doi: 10.1109/MSP.2011.943127.

[129] M J Schoelles and W D Gray. Argus: a suite of tools for research in complex cognition. *Behavior research methods, instruments, & computers : a journal of the Psychonomic Society, Inc*, 33(2):130–40, 2001. ISSN 0743-3808. doi: 10.3758/BF03195358. URL http://www.ncbi.nlm.nih.gov/pubmed/11447665.

[130] Kang G. Shin and Parameswaran Ramanathan. Real-Time Computing: A New Discipline of Computer Science and Engineering. *Proceedings of the IEEE*, 82(1): 6–24, 1994. ISSN 15582256. doi: 10.1109/5.259423.

[131] Xin Shuai Xin Shuai, Chao Zhang Chao Zhang, and Pengwei Hao Pengwei Hao. Fingerprint indexing based on composite set of reduced SIFT features. *2008 19th International Conference on Pattern Recognition*, pages 1–4, 2008. ISSN 1051-4651. doi: 10.1109/ICPR.2008.4761873.

[132] Tom Simonite. Moore's Law Is Dead. Now What? *MIT Tecnology Review*, 2016. URL https://www.technologyreview.com/s/601441/moores-law-is-dead-now-what/.

[133] Gavin J D Smith. Behind the Screens : Examining Constructions of Deviance and Informal Practices among CCTV Control Room Operators in the UK. *Surveillance & Society*, 2(2/3):376–395, 2004. ISSN 14777487.

[134] Sm Smith and Jm Brady. SUSAN—a new approach to low level image processing. *International journal of computer vision*, 23(1):45–78, 1997. ISSN 1573-1405. doi: 10.1023/A:1007963824710. URL http://link.springer.com/article/10.1023/A:1007963824710.

[135] I Sobel and G Feldman. A 3x3 isotropic gradient operator for image processing. *Pattern Classification and Scene Analysis*, 1(1):271–272, 1973.

[136] John A. Stankovic. A Serious Problem for Next-Generation Systems. *Computer*, 21 (10):10–19, 1988. ISSN 00189162. doi: 10.1109/2.7053.

[137] Chris Stauffer and W E L Grimson. Adaptive background mixture models for real-time tracking. *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, 2:246–252, 1999. doi: 10.1109/CVPR.1999.784637.

[138] Luc Steels. When are robots intelligent autonomous agents? *Robotics and Autonomous Systems*, 15(1-2):3–9, 1995. ISSN 09218890. doi: 10.1016/0921-8890(95)00011-4.

[139] Genyun Sun, Qinhuo Liu, Qiang Liu, Changyuan Ji, and Xiaowen Li. A novel approach for edge detection based on the theory of universal gravity. *Pattern Recognition*, 40(10):2766–2775, 2007. ISSN 00313203. doi: 10.1016/j.patcog.2007.01.006.

[140] M Tabb and N Ahuja. Multiscale image segmentation by integrated edge and region detection. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 6(1):642–55, 1997. ISSN 1057-7149. doi: 10.1109/83.568922.

[141] Shen-Chuan Tai and Shih-Ming Yang. A fast method for image noise estimation using Laplacian operator and adaptive edge detection. *2008 3rd International Symposium on Communications, Control and Signal Processing*, (March):1077–1081, 2008. doi: 10.1109/ISCCSP.2008.4537384. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp={&}arnumber=4537384{&}contentType=Conference+Publications{&}searchField=Search{_}All{&}queryText=noise+estimation+gradient+image.

[142] James Temperton. One Nation Under CCTV, 2015. URL http://www.wired.co.uk/article/one-nation-under-cctv.

[143] Punam Thakare. A Study of Image Segmentation and Edge Detection Techniques. *International Journal on Computer Science and Engineering (IJCSE)*, 3(2):899–904, 2011.

[144] Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Contextual models for object detection using boosted random fields. *Advances in Neural Information Processing Systems (NIPS)*, 17:1401–1408, 2005. doi: doi=10.1.1.167.1284.

[145] Egor Tsinko. *Background Subtraction with a Pan/Tilt Camera*. PhD thesis, British Columbia, IEEE, 2006.

[146] T Vaudrey, A Wedel, C Rabe, J Klappstein, and R Klette. Evaluation of moving object segmentation comparing 6D-vision and monocular motion constraints. In *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*, pages 1–6, 2008. doi: 10.1109/IVCNZ.2008.4762126.

[147] Louise A. Wallace and C. Robert Matthews. Sequential vs. parallel protein-folding mechanisms: Experimental tests for complex folding reactions. *Biophysical Chemistry*, 101-102:113–131, 2002. ISSN 03014622. doi: 10.1016/S0301-4622(02)00155-2.

[148] Mike Warner. Police Chase Harley, 2011. URL https://www.youtube.com/watch?v=W8e-JjJXTU8.

[149] Andreas Wedel, Clemens Rabe, Tobi Vaudrey, Thomas Brox, Uwe Franke, and Daniel Cremers. *Efficient dense scene flow from sparse or dense stereo data*. Springer, 2008. ISBN 3540886818.

[150] Andreas Wedel, Thomas Pock, Christopher Zach, Horst Bischof, and Daniel Cremers. An Improved Algortihm for TV-L1 Optical Flow. *Statistical and Geometrical Approaches to Visual Motion Analysis*, 5604(1):23–45, 2009. doi: 10.1007/978-3-642-03061-1_2.

[151] Juyang Weng, Thomas S Huang, and N Ahuja. 3-D Motion Estimation, Understanding, and Prediction from Noisy Image Sequences. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9(3):370–389, 1987. doi: 10.1109/TPAMI.1987.4767920.

[152] Alexander H Wertheim. Motion perception: Rights, wrongs and further speculations. *Behavioral and Brain Sciences*, 17(02):340–355, 1994.

[153] Alexander H. Wertheim. Motion perception during selfmotion: The direct versus inferential controversy revisited. *Behavioral and Brain Sciences*, 17(02):293, 1994. ISSN 0140-525X. doi: 10.1017/S0140525X00034646.

[154] John Winn and Jamie Shotton. The layout consistent random field for recognizing and segmenting partially occluded objects. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 37–44, 2006. ISBN 0769525970. doi: 10.1109/CVPR.2006.305.

[155] Andrew P Witkin. Scale-space filtering. *International Joint Conference on Artificial Intelligence*, 2:1019–1022, 1983. doi: 10.1109/ICASSP.1984.1172729. URL http://portal.acm.org/citation.cfm?id=1623607.

[156] C R Wren, A Azarbayejani, T Darrell, and A P Pentland. Pfinder: real-time tracking of the human body. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):780–785, 1997. doi: 10.1109/34.598236.

[157] C Zach, T Pock, and H Bischof. A Duality Based Approach for Realtime TV-L 1 Optical Flow. *Pattern Recognition*, 4713:214–223, 2007. doi: 10.1007/978-3-540-74936-3_22.

[158] Hui Zhang, Jason E. Fritts, and Sally A. Goldman. Image segmentation evaluation: A survey of unsupervised methods. *Computer Vision and Image Understanding*, 110:260–280, 2008. ISSN 1077-3142. doi: 10.1016/j.cviu.2007.08.003. URL http://www.sciencedirect.com/science/article/pii/S1077314207001294$\delimiter"026E30F$nhttp://www.sciencedirect.com/science/article/pii/S1077314207001294/pdfft?md5=f0d8b20c4a7c9def389b3d0188d348dd{&}pid=1-s2.0-S1077314207001294-main.pdf.

[159] Xiaojin Zhu. Semi-Supervised Learning Literature Survey. *Sciences-New York*, pages 1–59, 2007. ISSN 1520-5126. doi: 10.1.1.146.2352. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.146.2352{&}rep=rep1{&}type=pdf.

[160] Z Zivkovic. Improved adaptive Gaussian mixture model for background subtraction. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 28–31 Vol.2, 2004. ISBN 1051-4651. doi: 10.1109/ICPR.2004.1333992.

Appendix A

Motion Estimation Experiments

						Processing Speed (ms)			
Experiment	Video	Frame	NumKeyPoints	NumMatches	Post outlier	Feature Extraction	Overall	Match	Notes
SIFT (4)	StillBase	45	1010	1010		171.559	278.051		
SURF (4)	StillBase	45	1216	1216		68.226	183.793		
BRISK (4)	StillBase	45	816	816		21.0074	107.629		
ORB (4)	StillBase	45	500	500		8.31369	92.0585		
SIFT (4)	Helicopter	124	206	206		120.705	200.217		
SIFT (4)	Z-axis	104	1118	1118		255.366	402.975		
SURF (4)	Helicopter	124	552	552		43.7393	125.551		
SURF (4)	Z-axis	104	1551	1551		98.3831	261.273		
BRISK (4)	Helicopter	124	101	101		6.36801	74.9733		
BRISK (4)	Z-axis	104	1269	1269		33.5285	170.101		
ORB (4)	Helicopter	124	304	304		5.26065	78.6608		
ORB (4)	Z-axis	104	500	500		8.31132	105.431		
FLANN	StillBase	45	1216	1216		71.8985	200.402	26.9211	
FLANN	Helicopter	124	552	552		40.4291	123.297	9.99923	
BruteForce	StillBase	45	1216	1216		64.0866	185.151	16.0036	
BruteForce	Helicopter	124	552	552		45.506	123.274	5.12633	
BruteForce	StillBase	45	1216	890		71.8152	200.873	31.7875	KNN Cross Match
BruteForce	Helicopter	124	552	533		46.56	123.062	7.51843	KNN Cross Match
BruteForce	StillBase	45	1216	979		64.0486	185.619	31.8562	Radius Cross Match
BruteForce	Helicopter	124	552	544		45.4203	140.106	11.5915	Radius Cross Match
Outlier	StillBase	45	1216	1216	974	68.008	185.218	15.9459	
Outlier	Helicopter	124	552	552	538	47.7551	123.226	5.40879	
Homo_Lin	StillBase	45	1216	1216		64.1182	169.437	14.7003	
Homo_Lin	Helicopter	124	552	552		46.7315	123.542	15.2664	
Homo_Cubi	StillBase	45	1216	1216	974	70.5743	186.471	16.1015	
Homo_Cubi	Helicopter	124	552	552		44.3963	123.871	16.0411	
Homo_Cubi	StillBase	200	1216	1216	974	70.5743	186.471	16.1015	
Homo_Lin	StillBase	200	1216	1216		64.1182	169.437	14.7003	

Fig. A.1 Summary results of the experiments conducted with Motion Estimation

Appendix B

WISE Performance Analysis

This section contains a table of results showing the WISE performance over several hundred frames, and the processing time per object

Avg time per obj over all frames:	0.029803515	Standard Deviation:	0.00606977

Helicopter Video (640 x 360)			
Frame No	Num Obj	Time in S	Time per obj (ms)
1	542	0.024711	0.045592435
2	533	0.025619	0.048065478
3	533	0.022805	0.042785178
4	541	0.024581	0.045435675
5	536	0.025251	0.047109515
6	537	0.02395	0.0446
7	567	0.024551	0.0433
8	555	0.024131	0.043479459
9	556	0.023366	0.042025
10	554	0.025325	0.045712996
11	600	0.026264	0.0437735
12	580	0.025199	0.043447241
13	595	0.027789	0.046703529
14	557	0.025463	0.045713645
15	557	0.025751	0.046231418
16	545	0.024838	0.045574128
17	547	0.02457	0.044917733
18	540	0.025946	0.048047407
19	540	0.023803	0.044079259
20	541	0.026465	0.048919039
21	527	0.025516	0.048417078
22	527	0.024293	0.046096964
23	546	0.025709	0.047086813
24	564	0.02499	0.04430922
25	563	0.023614	0.041942451
26	566	0.024932	0.04405
27	548	0.026204	0.047818248
28	548	0.024126	0.044024818
29	556	0.027484	0.049430935
30	538	0.0259	0.048140892
31	597	0.025811	0.043234338
32	608	0.026547	0.043662664
33	569	0.025114	0.044137083
34	732	0.0264	0.036065574
35	742	0.025531	0.034408625
36	742	0.024061	0.032427493
37	708	0.026523	0.037462147
38	708	0.027696	0.039118927
39	550	0.026481	0.048147273
40	552	0.026072	0.047231341
41	550	0.024179	0.043961636
42	532	0.024638	0.04631203
43	693	0.026783	0.038647619
44	693	0.025941	0.037433333
45	698	0.025818	0.036987966
46	564	0.027383	0.04855195
47	561	0.024044	0.042858824
48	581	0.02724	0.046885198
49	690	0.028023	0.040613478
50	712	0.026543	0.037279494
51	573	0.02523	0.044031763
52	573	0.026238	0.045789878
53	655	0.026568	0.040561679
54	638	0.026739	0.041911129
55	639	0.026335	0.04121252
56	708	0.02738	0.038672458
57	707	0.026058	0.036856436
58	726	0.02803	0.038609366
59	718	0.027765	0.038670334
60	718	0.026614	0.037066992
61	618	0.027486	0.044475081
62	611	0.026643	0.043606056
63	610	0.028964	0.047481148
64	613	0.027907	0.045524796
65	627	0.028075	0.044777352
66	633	0.028076	0.044353555
67	637	0.024526	0.038502041
68	577	0.027048	0.046876776
69	622	0.025547	0.04107299
70	627	0.027141	0.043287241
71	628	0.024301	0.03869586
72	655	0.026715	0.040786565
73	606	0.025429	0.041962541
74	609	0.025649	0.042116585
75	621	0.026319	0.042381643
76	621	0.024674	0.039733172
77	642	0.025683	0.04000514
78	573	0.027614	0.048191274
79	573	0.023408	0.040852007
80	561	0.025435	0.045338324
81	550	0.025524	0.046406727
82	581	0.02563	0.044113597
83	633	0.027378	0.043251343
84	624	0.027288	0.043730929
85	641	0.025584	0.039912949
86	603	0.026033	0.043172471
87	657	0.026805	0.040799543
88	637	0.027397	0.043009576
89	584	0.027155	0.046497432
90	583	0.026285	0.045086449
91	646	0.025574	0.039587461
92	643	0.025685	0.039946034
93	662	0.026222	0.03960997
94	670	0.02464	0.036776119
95	689	0.024882	0.036113353
96	671	0.024724	0.036846051
97	739	0.026223	0.035483897
98	738	0.024675	0.033434553
99	745	0.027294	0.036636242
100	744	0.029088	0.039096237
101	744	0.02677	0.03598078
102	840	0.026449	0.031486905
103	857	0.028179	0.032880863
104	906	0.028973	0.031978587
105	893	0.02885	0.032306495
106	894	0.027137	0.03035481
107	963	0.028135	0.029216096
108	908	0.029365	0.032340198
109	907	0.028465	0.031383241
110	944	0.028839	0.03054947
111	943	0.027667	0.029338812
112	960	0.027475	0.028619688
113	1031	0.028401	0.027547333
114	1031	0.027455	0.026629292
115	1034	0.028841	0.02789236

Dashboard Video (848 x 480)			
Frame No	Num Obj	Time in S	Time per obj (ms)
1	2272	0.090669	0.039907262
2	2051	0.065034	0.031708532
3	2104	0.060556	0.028781274
4	2087	0.061708	0.029567897
5	2095	0.056979	0.027197375
6	1972	0.055523	0.028155527
7	2022	0.054243	0.026826261
8	1880	0.053547	0.028482447
9	1848	0.053251	0.02881553
10	1750	0.051142	0.029223714
11	1700	0.052191	0.030700647
12	1732	0.051468	0.02971582
13	1640	0.050975	0.031082378
14	1714	0.050514	0.029471237
15	1540	0.054805	0.035587338
16	1633	0.050226	0.030756705
17	1531	0.05045	0.032952384
18	1486	0.050421	0.03393035
19	1439	0.049196	0.034187491
20	1439	0.047099	0.032730229
21	1621	0.048846	0.030133128
22	1421	0.047619	0.033510978
23	1367	0.055538	0.040627725
24	1585	0.053214	0.033573754
25	1613	0.057564	0.035687291
26	1533	0.054891	0.035806001
27	1551	0.054154	0.034915667
28	1358	0.051616	0.038009057
29	1364	0.05369	0.039361877
30	1339	0.05714	0.042673413
31	1332	0.050239	0.037717117
32	1352	0.050537	0.037379142
33	1381	0.05148	0.037277625
34	1377	0.052462	0.03809862
35	1452	0.055171	0.03799635
36	1493	0.055881	0.037428667
37	1498	0.054778	0.036567156
38	1576	0.059982	0.038059518
39	1538	0.049336	0.032078023
40	1634	0.050698	0.031026805
41	1587	0.049988	0.03149811
42	1589	0.051421	0.03236073
43	1741	0.053758	0.030877657
44	1715	0.056256	0.032802274
45	1935	0.061139	0.031596124
46	1828	0.055399	0.030306018
47	1847	0.059904	0.032433297
48	1895	0.061901	0.032665435
49	1842	0.057272	0.031092074
50	1824	0.056461	0.030954715
51	1963	0.057653	0.029369995
52	2002	0.054473	0.027209291
53	2142	0.057681	0.026928525
54	1757	0.059708	0.033982698
55	1573	0.052336	0.033271265
56	2134	0.063398	0.029708294
57	1953	0.057077	0.029225243
58	2104	0.059121	0.028099477
59	1952	0.056444	0.028915984
60	2151	0.057626	0.026790516
61	2051	0.055212	0.026919649
62	2004	0.055422	0.027655489
63	2062	0.057327	0.027801406
64	2115	0.056213	0.026578203
65	2140	0.057335	0.026791822
66	2117	0.057867	0.027334247
67	2124	0.057503	0.027072834
68	2127	0.057766	0.02715858
69	2073	0.057426	0.027701737
70	1987	0.057062	0.028717614
71	1973	0.054878	0.027814496
72	2309	0.06109	0.026457471
73	2099	0.057587	0.027435588
74	2290	0.058945	0.025740218
75	2017	0.057613	0.028563907
76	2150	0.05907	0.027474419
77	1741	0.057412	0.032976163
78	1846	0.054703	0.029633207
79	1755	0.053495	0.030481197
80	1771	0.052408	0.029592095
81	1770	0.05173	0.029226045
82	1464	0.05092	0.034781557
83	1658	0.051651	0.031152413
84	1824	0.055279	0.03030625
85	1493	0.048006	0.032153717
86	1457	0.048825	0.033510501
87	1371	0.047235	0.034453027
88	1425	0.047952	0.033650316
89	1397	0.049708	0.035582105
90	1470	0.048098	0.032719388
91	1594	0.050895	0.031928858
92	1370	0.050319	0.036729197
93	1457	0.05903	0.040515031
94	1405	0.048662	0.034634947
95	1449	0.04839	0.033395652
96	1415	0.048917	0.034570247
97	1392	0.046859	0.033662787
98	1349	0.046223	0.03426464
99	1491	0.048603	0.03259772
100	1590	0.051109	0.032144088
101	1637	0.050573	0.030893708
102	1599	0.050013	0.031277548
103	1643	0.052379	0.031880219
104	1617	0.050302	0.031107978
105	1658	0.052393	0.031600121
106	1724	0.05216	0.030255162
107	1673	0.052649	0.031469576
108	1854	0.057211	0.030858037
109	1744	0.053628	0.030750229
110	1755	0.05361	0.030546895
111	1889	0.054913	0.029069931
112	1903	0.056651	0.029769522
113	1876	0.053847	0.028703198
114	1923	0.056689	0.029479199
115	1941	0.055923	0.02881118

Drone Video (1920 x 1080)			
Frame No	Num Obj	Time in S	Time per obj (ms)
1	4601	0.21079	0.045813519
2	4021	0.20715	0.05151803
3	4330	0.20092	0.046401155
4	4177	0.20645	0.049424707
5	4335	0.20847	0.048090196
6	4764	0.20813	0.043688917
7	4737	0.20437	0.043142284
8	4874	0.20883	0.042845917
9	5117	0.21175	0.041381278
10	5061	0.20744	0.040988342
11	5258	0.20912	0.039771206
12	5620	0.20866	0.037128648
13	5472	0.20938	0.038264254
14	5796	0.21242	0.036648551
15	6481	0.22411	0.034579849
16	6450	0.21422	0.033212713
17	6744	0.23608	0.035006524
18	7250	0.23785	0.03280731
19	7356	0.24802	0.03371683
20	7603	0.25372	0.033371169
21	8411	0.28171	0.033493164
22	8487	0.25766	0.030359844
23	8897	0.26417	0.029691694
24	9550	0.25632	0.026839791
25	9498	0.27685	0.029148136
26	9817	0.27018	0.027521239
27	10626	0.27541	0.025918408
28	10405	0.25686	0.02468592
29	10391	0.28351	0.027284284
30	10926	0.27099	0.024802581
31	10683	0.26564	0.024865862
32	10880	0.27926	0.025667371
33	11699	0.28206	0.024109582
34	11535	0.2902	0.025158474
35	11329	0.30207	0.026663165
36	11997	0.30741	0.025623989
37	11922	0.2999	0.025155511
38	12086	0.28963	0.023964008
39	12581	0.29974	0.023824815
40	12301	0.28844	0.023448094
41	12643	0.2949	0.023324765
42	12872	0.30944	0.024040009
43	12890	0.29377	0.022790225
44	12860	0.29315	0.022795179
45	13494	0.3084	0.022854528
46	13550	0.31906	0.02354679
47	13728	0.31769	0.023141827
48	13776	0.31837	0.023110119
49	13882	0.30723	0.022131609
50	13784	0.30198	0.021907719
51	14329	0.32751	0.022856724
52	14473	0.32641	0.022552961
53	14145	0.31312	0.022136727
54	13625	0.32751	0.024037651
55	13947	0.30854	0.02212232
56	13547	0.30631	0.022611132
57	13358	0.30346	0.022717323
58	13809	0.29982	0.021711565
59	13396	0.30043	0.022426471
60	12610	0.30589	0.024258049
61	12855	0.31625	0.024601089
62	12639	0.29859	0.023624575
63	11832	0.30233	0.02555164
64	12431	0.29792	0.02396557
65	12217	0.30301	0.024801997
66	12168	0.30326	0.024922337
67	12745	0.30268	0.023748529
68	11697	0.29493	0.025213901
69	11191	0.27503	0.024576088
70	11661	0.28299	0.024268416
71	11406	0.27642	0.024234613
72	11259	0.28293	0.025129052
73	11506	0.29293	0.025459152
74	11845	0.28568	0.02411794
75	11188	0.27279	0.024382106
76	11154	0.28776	0.025798817
77	11763	0.29743	0.025285386
78	10367	0.27656	0.026676859
79	10160	0.27485	0.027052264
80	10819	0.27565	0.02547851
81	10595	0.27612	0.026060878
82	10402	0.27116	0.026067583
83	10269	0.27281	0.026566754
84	11254	0.28137	0.025002133
85	11392	0.28663	0.025160376
86	11454	0.29845	0.026056138
87	11946	0.28838	0.024140549
88	11902	0.29245	0.024571333
89	11858	0.29526	0.024899983
90	12193	0.2928	0.024014024
91	10441	0.27729	0.026557705
92	9896	0.26719	0.026999697
93	10676	0.26632	0.024945392
94	10219	0.26263	0.025699971
95	10008	0.2782	0.027797562
96	10780	0.2622	0.02432282
97	9745	0.26717	0.027416521
98	10404	0.2774	0.026663014
99	10379	0.27282	0.026285288
100	10213	0.26177	0.025631058
101	9881	0.26346	0.026663293
102	10109	0.27272	0.026977743
103	9684	0.25937	0.026783354
104	9847	0.26898	0.027315832
105	10522	0.26179	0.024879871
106	10161	0.27964	0.027521012
107	10218	0.26377	0.025814641
108	10564	0.26531	0.025114729
109	9947	0.25417	0.025552428
110	10374	0.26622	0.025662425
111	10521	0.25981	0.024694421
112	10108	0.26245	0.025964088
113	10567	0.27578	0.02609823
114	10586	0.25826	0.024396278
115	10551	0.26232	0.024861719

				Avg time per obj over all frames:			0.029803515	Standard Deviation:	0.00606977		
Helicopter Video (640 x 360)				Dashboard Video (848 x 480)				Drone Video (1920 x 1080)			
Frame No	Num Obj	Time in S	Time per obj (ms)	Frame No	Num Obj	Time in S	Time per obj (ms)	Frame No	Num Obj	Time in S	Time per obj (ms)
116	1036	0.027953	0.026981274	116	1930	0.055546	0.028780104	116	10464	0.26255	0.025090979
117	1035	0.027634	0.026699614	117	2058	0.05807	0.02821691	117	10923	0.26302	0.02407974
118	1018	0.031152	0.030600982	118	1783	0.059775	0.033525014	118	10258	0.25392	0.024753071
119	969	0.030736	0.031719092	119	1737	0.054141	0.031169142	119	10130	0.25778	0.025447581
120	969	0.029214	0.030148194	120	1938	0.057819	0.029834469	120	10954	0.2681	0.024474895
121	963	0.027724	0.028789408	121	1828	0.0545	0.029814168	121	10943	0.26694	0.024393402
122	1014	0.030952	0.030524556	122	1961	0.058096	0.029625548	122	10443	0.25591	0.024505602
123	1048	0.029683	0.028322996	123	1841	0.057738	0.031362412	123	11272	0.26523	0.023530252
124	1002	0.030147	0.030086527	124	1889	0.05586	0.029571255	124	10556	0.25859	0.024496495
125	1005	0.027907	0.027768259	125	1997	0.056022	0.028052929	125	10758	0.25929	0.024101785
126	1085	0.031352	0.028895945	126	2014	0.056254	0.027931281	126	11681	0.2688	0.023011386
127	1161	0.029716	0.025595263	127	2080	0.054999	0.026441731	127	11402	0.26491	0.02323338
128	1162	0.030899	0.026590792	128	2045	0.056737	0.027744108	128	11283	0.26046	0.023084197
129	1138	0.032353	0.028429525	129	2030	0.055927	0.027550345	129	12077	0.26457	0.021906517
130	1141	0.02904	0.025451183	130	2228	0.0566	0.02540377	130	11307	0.26135	0.023114354
131	1199	0.029736	0.02480025	131	2083	0.058431	0.028051224	131	11600	0.27529	0.023731983
132	1205	0.032019	0.026572033	132	1961	0.059631	0.03040821	132	12473	0.27573	0.02210647
133	1214	0.029901	0.024630478	133	1879	0.054354	0.028926929	133	11420	0.26548	0.023246848
134	1270	0.032367	0.025485827	134	2038	0.056262	0.027606428	134	11567	0.26988	0.023332065
135	1307	0.031785	0.024319051	135	1874	0.055775	0.029762593	135	12122	0.26571	0.021919898
136	1307	0.029189	0.0223329	136	1862	0.056145	0.030153008	136	11679	0.26522	0.022708794
137	1157	0.031553	0.027271219	137	1946	0.057444	0.029518756	137	11956	0.2722	0.022766477
138	1323	0.032102	0.024264777	138	1907	0.056033	0.029382853	138	12645	0.28683	0.022682958
139	1337	0.033442	0.025012341	139	1860	0.055667	0.029928226	139	11918	0.26115	0.021911982
140	1355	0.033724	0.024888561	140	1797	0.056964	0.031699444	140	12282	0.28109	0.022886338
141	1364	0.033328	0.024433871	141	2003	0.056796	0.028355267	141	13302	0.27286	0.02051263
142	1473	0.033677	0.022863136	142	2074	0.057429	0.027690116	142	12548	0.2704	0.021549012
143	1506	0.034272	0.022756707	143	2087	0.059742	0.028625922	143	12793	0.27478	0.021478699
144	1503	0.032371	0.021537525	144	2073	0.060045	0.028965461	144	13669	0.28958	0.021185164
145	1408	0.032753	0.02326179	145	2117	0.058656	0.027707085	145	12712	0.2722	0.021412681
146	1410	0.031172	0.02210773	146	2063	0.058527	0.028369898	146	13021	0.27855	0.021392597
147	1433	0.032296	0.022537544	147	2150	0.061543	0.028624465	147	13627	0.29558	0.021690688
148	1570	0.034505	0.021977962	148	2033	0.05703	0.028052287	148	12915	0.27777	0.021507859
149	1569	0.03254	0.020739069	149	2097	0.062856	0.029974154	149	12967	0.27773	0.021418215
150	1561	0.035408	0.022683024	150	2173	0.061187	0.028157708	150	13646	0.27778	0.020356222
151	1360	0.034812	0.025597059	151	2188	0.059887	0.027370795	151	12966	0.27925	0.021537406
152	1362	0.031682	0.023261307	152	2354	0.063232	0.02686147	152	12952	0.28483	0.02199143
153	1342	0.034022	0.025351639	153	2368	0.061482	0.025963598	153	13784	0.29311	0.021264292
154	1290	0.033035	0.025608217	154	2226	0.063625	0.028582839	154	12749	0.27621	0.021664993
155	1289	0.031599	0.024514042	155	2219	0.059975	0.027027941	155	12761	0.27857	0.021830107
156	1306	0.033371	0.025551914	156	2141	0.060362	0.028193274	156	13707	0.29207	0.021308383
157	1282	0.033636	0.026237051	157	2026	0.058185	0.028719348	157	13006	0.28644	0.022023528
158	1329	0.033564	0.02525538	158	2113	0.056297	0.026642972	158	12904	0.27796	0.021540453
159	1333	0.033311	0.024989347	159	2020	0.056892	0.028164356	159	13617	0.28407	0.02086157
160	1333	0.031473	0.023610653	160	1970	0.055906	0.028378426	160	13081	0.28743	0.021973091
161	1444	0.040055	0.027739127	161	1901	0.055272	0.029075329	161	13317	0.28806	0.021630923
162	1477	0.033456	0.022651185	162	2030	0.05611	0.027640296	162	14097	0.29399	0.020854721
163	1481	0.032946	0.022245847	163	1924	0.054443	0.02829657	163	13190	0.27784	0.021064519
164	1468	0.034977	0.02382609	164	2066	0.057511	0.027836689	164	13558	0.28389	0.020938708
165	1467	0.032674	0.022272597	165	2141	0.057056	0.026649183	165	14106	0.29666	0.021031051
166	1487	0.034048	0.022897108	166	2223	0.057754	0.025980387	166	13296	0.28232	0.021233228
167	1437	0.034269	0.02384746	167	2263	0.061218	0.027051834	167	13683	0.29607	0.021637726
168	1427	0.032419	0.02271808	168	2194	0.062447	0.028462397	168	14336	0.29625	0.02066469
169	1445	0.034885	0.024141661	169	2280	0.060486	0.026529035	169	13541	0.28086	0.020741526
170	1462	0.035946	0.024587004	170	1984	0.059151	0.029814163	170	13342	0.2937	0.022012892
171	1439	0.031846	0.022130577	171	2079	0.058362	0.02807215	171	14125	0.28759	0.020360071
172	1412	0.034173	0.024201983	172	2080	0.055607	0.026733894	172	13346	0.28943	0.021686348
173	1363	0.0344	0.025238151	173	1875	0.054664	0.02915424	173	13481	0.2975	0.022068244
174	1316	0.034197	0.02598579	174	2167	0.057576	0.026569543	174	14059	0.30059	0.021380895
175	1378	0.035241	0.025574311	175	2146	0.058782	0.027391426	175	13471	0.2792	0.020726004
176	1304	0.0328	0.025153374	176	2125	0.057553	0.027083529	176	13339	0.28337	0.021243571
177	1258	0.03242	0.025771304	177	2033	0.058047	0.028552238	177	14071	0.29559	0.021006965
178	1322	0.035359	0.026746445	178	2098	0.058212	0.027746568	178	13305	0.27931	0.02099316
179	1328	0.032982	0.024835617	179	2136	0.05674	0.026563624	179	13199	0.27634	0.020936738
180	1358	0.033206	0.024452209	180	2071	0.058161	0.028083679	180	13901	0.297	0.021365441
181	1344	0.034217	0.025459375	181	2226	0.058136	0.026116801	181	13197	0.27257	0.020653558
182	1346	0.032135	0.02387422	182	2156	0.056046	0.025995269	182	13092	0.27922	0.021327452
183	1418	0.034587	0.024391537	183	2077	0.05703	0.027457824	183	13751	0.30062	0.021861319
184	1417	0.031767	0.022418701	184	2150	0.057324	0.026662419	184	12978	0.27565	0.021240022
185	1429	0.032933	0.023046256	185	2010	0.057042	0.028379154	185	13216	0.28025	0.021205281
186	1487	0.035262	0.023713786	186	2007	0.056798	0.02829995	186	14101	0.28771	0.02040373
187	1478	0.031649	0.021413261	187	1941	0.056154	0.028930654	187	13045	0.28802	0.022079264
188	1339	0.035432	0.026461763	188	1951	0.055864	0.028633316	188	13211	0.29242	0.022134282
189	1394	0.033747	0.02420868	189	1865	0.056169	0.030117426	189	13551	0.27907	0.020594052
190	1394	0.033093	0.02373967	190	1934	0.056708	0.029321355	190	13145	0.2946	0.022411715
191	1367	0.035119	0.025690856	191	1763	0.053314	0.030240669	191	13137	0.27517	0.02094603
192	1338	0.033624	0.025129821	192	1788	0.055269	0.030910906	192	13674	0.28069	0.020527059
193	1355	0.033418	0.024662878	193	1837	0.052516	0.028587861	193	12945	0.30433	0.023509231
194	1349	0.033742	0.02501275	194	1750	0.05265	0.0300856	194	12811	0.27072	0.021131606
195	1333	0.032569	0.024433008	195	1810	0.053382	0.029492541	195	13403	0.28224	0.021058047
196	1321	0.032757	0.024796745	196	1788	0.055401	0.030985123	196	12593	0.27268	0.021653458
197	1339	0.033342	0.024900971	197	1645	0.051764	0.031467538	197	12581	0.2838	0.022557984
198	1338	0.031513	0.023552466	198	1621	0.050484	0.031143553	198	13175	0.27421	0.020813207
199	1275	0.033817	0.02652298	199	1673	0.051957	0.031055888	199	12467	0.27364	0.021949386
200	1280	0.032811	0.025633906	200	1740	0.051948	0.029855287	200	12412	0.27065	0.021805672
201	1424	0.035258	0.024759831	201	1781	0.05124	0.028770241	201	13088	0.2725	0.020820523
202	1425	0.031047	0.021787439	202	1651	0.049149	0.02976911	202	12264	0.27274	0.022238829
203	1421	0.034979	0.024615834	203	1639	0.050364	0.030728493	203	12341	0.2732	0.022137671
204	1421	0.032941	0.023181562	204	1886	0.053097	0.028153022	204	12914	0.27862	0.02157519
205	1466	0.033728	0.023006821	205	1826	0.051191	0.028034447	205	12142	0.26555	0.021869956
206	1346	0.033174	0.024646434	206	1661	0.049698	0.02992071	206	13183	0.28101	0.021316316
207	1417	0.033639	0.023739661	207	1619	0.049615	0.030645213	207	12564	0.27696	0.022043935
208	1328	0.034673	0.026108886	208	1558	0.051163	0.032839024	208	12559	0.28384	0.022600685
209	1266	0.034113	0.026945577	209	1574	0.048476	0.030798221	209	13244	0.28192	0.021286394
210	1286	0.033231	0.02584028	210	1525	0.048717	0.031945574	210	12394	0.27263	0.021997095
211	1412	0.03283	0.023250992	211	1605	0.048783	0.030394143	211	11967	0.26475	0.022123088
212	1418	0.032377	0.022832511	212	1671	0.051571	0.030862059	212	12622	0.27395	0.021704167
213	1347	0.034592	0.025680772	213	1689	0.049611	0.029372765	213	11809	0.26362	0.022323651
214	1358	0.033145	0.024407364	214	1766	0.050084	0.028359853	214	11853	0.26939	0.022727664
215	1260	0.032418	0.02572881	215	1752	0.054289	0.030986587	215	12321	0.27204	0.022079377
216	1306	0.033528	0.025672358	216	1894	0.053664	0.028333844	216	12351	0.26358	0.021340377
217	1307	0.030764	0.02353772	217	1780	0.053409	0.030005112	217	12064	0.26831	0.022240302
218	1343	0.034109	0.025397468	218	1799	0.051199	0.028459589	218	11931	0.26239	0.02199254
219	1334	0.034623	0.025954423	219	1812	0.051669	0.028514625	219	12295	0.26392	0.02146588
220	1334	0.031001	0.02323928	220	1764	0.053648	0.030412528	220	11518	0.27496	0.023872026
221	1462	0.034415	0.023539672	221	1826	0.051459	0.028181325	221	11576	0.26299	0.022718296
222	1431	0.034171	0.023879245	222	1799	0.053114	0.029523958	222	12272	0.26621	0.021692797
223	1445	0.03244	0.022449896	223	1782	0.052122	0.029249158	223	11366	0.25409	0.022355358
224	1330	0.031381	0.023594962	224	1827	0.056399	0.030869513	224	11541	0.27215	0.023581232
225	1343	0.034665	0.025811616	225	1801	0.052412	0.029101832	225	12159	0.27537	0.022647422
226	1365	0.03418	0.025040513	226	1856	0.052678	0.028382705	226	11477	0.27404	0.023877233
227	1302	0.033876	0.02601851	227	1815	0.052907	0.029149642	227	11790	0.26458	0.022441221
228	1366	0.034017	0.024902709	228	1907	0.055429	0.029065915	228	12129	0.26645	0.021968093
229	1294	0.032549	0.025153864	229	1828	0.053205	0.029105744	229	11744	0.26267	0.022366144
230	1318	0.031511	0.023908422	230	1729	0.052807	0.030541932	230	11409	0.25986	0.02277693

Avg time per obj over all frames:	0.029803515	Standard Deviation:	0.00606977

Helicopter Video (640 x 360)

Frame No	Num Obj	Time in S	Time per obj (ms)
231	1312	0.029522	0.022501601
232	1263	0.030671	0.024284244
233	1261	0.028954	0.022961142
234	1193	0.032223	0.027009891
235	1209	0.032453	0.026842928
236	1204	0.030563	0.025384635
237	1222	0.032486	0.026583879
238	1196	0.03312	0.027692642
239	1207	0.030225	0.025041425
240	1173	0.032254	0.027496675
241	1226	0.033615	0.027418108
242	1237	0.030942	0.025013743
243	1313	0.0324	0.024676161
244	1267	0.034462	0.027199684
245	1284	0.032638	0.025418692
246	1244	0.033347	0.026806029
247	1241	0.030989	0.024971313
248	1229	0.030783	0.02504703
249	1240	0.032845	0.026487581
250	1228	0.031296	0.025485016
251	1254	0.032833	0.026182456
252	1247	0.029387	0.023566159
253	1287	0.031723	0.024648485
254	1333	0.032956	0.024723481
255	1332	0.030734	0.023073724
256	1363	0.03439	0.025231181
257	1404	0.035044	0.024959758
258	1426	0.032831	0.023023001
259	1439	0.035431	0.024622029
260	1445	0.03756	0.025993149
261	1566	0.037499	0.023945849
262	1504	0.037371	0.024847872
263	1530	0.036541	0.023883137
264	1492	0.036149	0.024228753
265	1553	0.036073	0.023227688
266	1553	0.033529	0.021589955
267	1569	0.036789	0.023447355
268	1570	0.034577	0.022023503
269	1570	0.036771	0.023420828
270	1528	0.037648	0.024638874
271	1526	0.032859	0.021532634
272	1451	0.036784	0.02535093
273	1366	0.036049	0.02639041
274	1368	0.034384	0.025134357
275	1374	0.03485	0.025363828
276	1321	0.034397	0.026038759
277	1325	0.032772	0.024733509
278	1403	0.035559	0.025344761
279	1296	0.036507	0.028169136
280	1366	0.035001	0.025623133
281	1343	0.035191	0.026203202
282	1338	0.034992	0.026152317
283	1320	0.033905	0.02568553
284	1252	0.033719	0.026931709
285	1252	0.032209	0.025725958
286	1227	0.03421	0.027880929
287	1231	0.032201	0.026158408
288	1276	0.034249	0.026841223
289	1289	0.034408	0.026693173
290	1286	0.032296	0.025113453
291	1200	0.033903	0.028252167
292	1159	0.033054	0.028519068
293	1158	0.033462	0.028896114
294	1103	0.032971	0.029892112
295	1176	0.034503	0.029339371
296	1185	0.031965	0.026974515
297	1114	0.032288	0.028983483
298	1149	0.032962	0.028687119
299	1121	0.033897	0.03023818
300	1137	0.033072	0.029087247
301	1138	0.031557	0.027730316
302	1182	0.033827	0.02861819
303	1182	0.032543	0.027532318
304	1178	0.030953	0.026275637
305	1063	0.032999	0.031042992
306	1063	0.032227	0.030317121
307	1038	0.031333	0.030186031
308	1115	0.032453	0.029105919
309	1116	0.029255	0.026214247
310	1071	0.031464	0.029378058
311	1011	0.032307	0.031955391
312	1012	0.029141	0.028795751
313	1048	0.031969	0.03050458
314	1033	0.031988	0.030966215
315	958	0.032049	0.033453653
316	1021	0.031369	0.030723506
317	940	0.030213	0.03214117
318	940	0.032256	0.034314468
319	928	0.030137	0.032475431
320	927	0.02907	0.031359008
321	949	0.029666	0.03126059
322	949	0.027926	0.02942687
323	944	0.031486	0.033354237
324	861	0.029536	0.034304297
325	862	0.028217	0.032734687
326	956	0.030278	0.031671025
327	860	0.02986	0.034720814
328	861	0.028491	0.033091057
329	883	0.030261	0.034270102
330	821	0.029385	0.035791352
331	827	0.028004	0.033861548
332	829	0.030138	0.036355127
333	907	0.030608	0.033746196
334	976	0.031014	0.031776127
335	974	0.030672	0.031490965
336	938	0.030684	0.032712154
337	879	0.029333	0.033370762
338	894	0.030378	0.033979642
339	888	0.028572	0.032175676
340	905	0.030637	0.033852597
341	905	0.028965	0.032005193
342	875	0.030387	0.034727886
343	871	0.030459	0.03496992
344	872	0.028715	0.032930046
345	909	0.03062	0.033685369

Dashboard Video (848 x 480)

Frame No	Num Obj	Time in S	Time per obj (ms)
231	1839	0.054492	0.029631321
232	1768	0.054367	0.030750452
233	1860	0.053029	0.028510054
234	1872	0.053794	0.028735844
235	1741	0.05241	0.030103619
236	1783	0.054268	0.030436119
237	1863	0.052535	0.028199087
238	1786	0.054063	0.030270381
239	1772	0.052801	0.029797235
240	1985	0.058718	0.029581058
241	1873	0.054274	0.028976775
242	1872	0.054517	0.029122062
243	2069	0.05779	0.027931174
244	1957	0.057041	0.029147113
245	2026	0.055955	0.027618608
246	2047	0.056032	0.027372692
247	1893	0.056952	0.03008579
248	1863	0.055126	0.029589748
249	1779	0.051808	0.029122147
250	1832	0.055991	0.030562664
251	1792	0.054581	0.030458092
252	1970	0.058808	0.029851777
253	1904	0.053742	0.028225945
254	1939	0.05644	0.029107994
255	1892	0.05853	0.030935254
256	1959	0.054561	0.027851506
257	2018	0.056559	0.028027403
258	2005	0.054672	0.027267681
259	2010	0.057338	0.028526567
260	1887	0.05376	0.028489613
261	1820	0.053582	0.029440549
262	1840	0.054784	0.029773641
263	1898	0.054455	0.028690832
264	1872	0.056825	0.030355128
265	1917	0.056361	0.029400626
266	1939	0.05825	0.030041362
267	1994	0.056324	0.028246891
268	1928	0.059221	0.030716234
269	1964	0.057186	0.029117057
270	2053	0.056019	0.027286556
271	1919	0.054516	0.02840839
272	1772	0.054774	0.030910892
273	1900	0.054174	0.028512632
274	1876	0.052659	0.028069989
275	1836	0.054969	0.029939651
276	1985	0.056794	0.028611385
277	1877	0.054854	0.029224241
278	1819	0.053645	0.029491699
279	1907	0.053276	0.027937074
280	1924	0.054612	0.028384459
281	2030	0.055381	0.027281034
282	1934	0.057941	0.029959307
283	1977	0.055002	0.027821143
284	1952	0.054857	0.028103176
285	2056	0.059188	0.028787792
286	1970	0.053218	0.027014162
287	1930	0.053024	0.027473731
288	2174	0.058614	0.026961546
289	2100	0.056502	0.026905762
290	2040	0.055191	0.027054167
291	2046	0.054936	0.026850244
292	1903	0.053927	0.028338098
293	1825	0.05222	0.028613918
294	1905	0.053355	0.028007664
295	1835	0.054779	0.029852207
296	1678	0.052948	0.031554052
297	1727	0.053656	0.031068674
298	1619	0.050613	0.031261581
299	1648	0.053873	0.032689867
300	1793	0.055839	0.03114261
301	1534	0.049784	0.032453977
302	1558	0.049089	0.031507766
303	1626	0.050716	0.031190652
304	1765	0.051883	0.029395241
305	1745	0.050295	0.028822464
306	1829	0.052348	0.028621104
307	1711	0.050549	0.029543483
308	1623	0.050696	0.031236106
309	1771	0.052181	0.029464088
310	1956	0.053736	0.027472495
311	1950	0.054467	0.027931846
312	1904	0.053505	0.028101208
313	1891	0.05217	0.027588472
314	1728	0.052657	0.030472743
315	1706	0.055482	0.032521454
316	1709	0.054826	0.032080866
317	1701	0.050913	0.029931041
318	1675	0.050151	0.029941134
319	1677	0.050427	0.030069887
320	1729	0.051314	0.029678485
321	1741	0.050262	0.028869673
322	1782	0.050845	0.028532435
323	1811	0.051531	0.028454224
324	2117	0.056937	0.026895324
325	2103	0.054908	0.026109368
326	2145	0.056319	0.026255991
327	1997	0.055321	0.027701953
328	2074	0.054501	0.026278255
329	2127	0.053945	0.025362106
330	2239	0.054507	0.02434435
331	2164	0.055793	0.025782116
332	2130	0.057836	0.027152958
333	2282	0.057503	0.025198379
334	2189	0.058121	0.026551211
335	2192	0.056026	0.025559489
336	2307	0.05911	0.025621803
337	2219	0.054287	0.024464443
338	2187	0.057573	0.026324966
339	2139	0.057191	0.026737447
340	2153	0.055047	0.02556758
341	2304	0.055998	0.024304688
342	2164	0.056913	0.026300046
343	2079	0.056402	0.027129437
344	2092	0.055734	0.026641539
345	2139	0.057286	0.026781627

Drone Video (1920 x 1080)

Frame No	Num Obj	Time in S	Time per obj (ms)
231	12208	0.27535	0.022554472
232	11587	0.27296	0.023557435
233	11689	0.26044	0.022280691
234	12333	0.29164	0.023646801
235	11825	0.25975	0.021966258
236	11629	0.2577	0.022159945
237	12473	0.26874	0.021545659
238	11596	0.26648	0.022980683
239	11710	0.26685	0.022788386
240	12385	0.27066	0.021854098
241	11726	0.28277	0.024114702
242	11687	0.25765	0.022045692
243	12307	0.26826	0.021797676
244	11574	0.28396	0.024533869
245	11856	0.25749	0.02171778
246	13044	0.26925	0.020641751
247	11936	0.26241	0.02198492
248	12195	0.27679	0.022697335
249	12574	0.26817	0.021327581
250	11906	0.26452	0.022217201
251	12458	0.27419	0.022009392
252	12682	0.27002	0.02129191
253	12111	0.25846	0.021340517
254	11959	0.26533	0.022186303
255	12883	0.27181	0.021098269
256	11944	0.25974	0.021746735
257	12216	0.27067	0.022156843
258	12959	0.28926	0.022321398
259	12353	0.26387	0.021360479
260	11869	0.27033	0.022775887
261	13030	0.28162	0.021613047
262	12512	0.26537	0.021209159
263	11528	0.26444	0.022938931
264	12023	0.26506	0.022045912
265	11075	0.25767	0.023265463
266	10909	0.26321	0.024127601
267	11533	0.26263	0.022772219
268	11064	0.26077	0.023569234
269	11258	0.25747	0.022869693
270	12223	0.26343	0.021551747
271	11136	0.26102	0.023439655
272	11452	0.26614	0.023239783
273	12554	0.27164	0.021637645
274	12086	0.28296	0.023412212
275	12263	0.26865	0.021907445
276	13368	0.28078	0.02100389
277	12743	0.2691	0.021117084
278	13125	0.27122	0.020664
279	13645	0.27776	0.020356394
280	12896	0.27503	0.021326458
281	12862	0.27289	0.021216918
282	13623	0.30514	0.022398664
283	12876	0.27678	0.021495961
284	12834	0.27467	0.021401667
285	13351	0.30725	0.023012958
286	12567	0.27125	0.021584547
287	12851	0.26778	0.020837289
288	13652	0.27931	0.02045942
289	12676	0.28823	0.022738088
290	12694	0.26725	0.021053017
291	13615	0.27768	0.020395299
292	12874	0.27142	0.021082492
293	12900	0.26756	0.02074093
294	13463	0.27865	0.020697244
295	12660	0.47307	0.037367141
296	12544	0.28557	0.022765784
297	13466	0.27479	0.020406208
298	12294	0.41961	0.034131121
299	12380	0.26566	0.021459128
300	13211	0.28148	0.021306336
301	12384	0.37063	0.029928133
302	12484	0.28387	0.022738946
303	12774	0.26339	0.020618913
304	11808	0.3628	0.030725186
305	11947	0.27049	0.022640747
306	12896	0.2636	0.020440059
307	12169	0.38555	0.031682965
308	12133	0.27274	0.022479519
309	12833	0.2682	0.020898855
310	12033	0.38661	0.032128896
311	11799	0.25395	0.021523265
312	12399	0.26192	0.021124204
313	11356	0.38829	0.034192585
314	11083	0.27101	0.024452585
315	12080	0.27307	0.02260505
316	11273	0.34409	0.030523286
317	11692	0.26251	0.022452446
318	11951	0.25979	0.021738181
319	11188	0.38291	0.034225063
320	10933	0.25228	0.023075002
321	11550	0.25691	0.022243203
322	10594	0.26484	0.024998962
323	10910	0.40101	0.036755912
324	11329	0.26943	0.023782417
325	10393	0.4861	0.046771673
326	10781	0.25911	0.024033485
327	11319	0.26354	0.023282622
328	10375	0.26897	0.02592453
329	10525	0.28378	0.026962755
330	11070	0.28606	0.025841012
331	10338	0.42219	0.04083904
332	10479	0.27701	0.02643487
333	11007	0.27412	0.02490397
334	10511	0.36715	0.034929788
335	10887	0.27812	0.02554588
336	11815	0.28138	0.023815658
337	11048	0.28123	0.025455014
338	11319	0.2565	0.022660747
339	11439	0.24911	0.021777078
340	11211	0.24886	0.022198198
341	12175	0.28013	0.023008789
342	11611	0.25351	0.021833262
343	11343	0.27045	0.023842987
344	12260	0.40982	0.033427162
345	11410	0.27594	0.024183611

				Avg time per obj over all frames:				0.029803515	Standard Deviation:	0.00606977			
Helicopter Video (640 x 360)					Dashboard Video (848 x 480)					Drone Video (1920 x 1080)			
Frame No	Num Obj	Time in S	Time per obj (ms)		Frame No	Num Obj	Time in S	Time per obj (ms)		Frame No	Num Obj	Time in S	Time per obj (ms)
346	843	0.030097	0.035702254		346	2051	0.059917	0.029213749		346	11602	0.25264	0.021775556
347	844	0.028752	0.034066114		347	2236	0.058695	0.02625		347	12357	0.40141	0.032484098
348	775	0.029889	0.038566194		348	2166	0.058304	0.02691759		348	10930	0.25116	0.022979323
349	769	0.029256	0.038043563		349	2074	0.057344	0.027649132		349	11058	0.26133	0.023632935
350	764	0.027221	0.03562945		350	1992	0.056891	0.028559639		350	11867	0.29277	0.024671273
351	742	0.029165	0.039306469		351	1929	0.054765	0.02839015		351	11336	0.27658	0.02439873
352	741	0.029281	0.03951525		352	1859	0.056518	0.030402313		352	11404	0.27422	0.02404551
353	742	0.028448	0.038339084		353	1813	0.054632	0.03013337		353	10561	0.24773	0.023456964
354	752	0.028997	0.038559973		354	1724	0.054706	0.031731845		354	11360	0.27556	0.024256866
355	751	0.027202	0.036220373		355	1936	0.054078	0.027932851		355	10726	0.24302	0.022657375
356	717	0.028505	0.03975523		356	1857	0.054618	0.02941217		356	11141	0.36611	0.032861682
357	729	0.028948	0.039709328		357	1918	0.05703	0.029734046		357	11681	0.28365	0.024283109
358	731	0.027482	0.037594528		358	1959	0.054922	0.02803563		358	10859	0.28539	0.026281702
359	766	0.028048	0.036615927		359	1923	0.054943	0.028571555		359	10953	0.2941	0.026851
360	766	0.027248	0.03557154		360	1903	0.056604	0.029744824		360	11708	0.4064	0.034711565
361	731	0.027557	0.037697811		361	2020	0.059764	0.02958599		361	11094	0.27712	0.024979178
362	766	0.028667	0.037424543		362	1873	0.054835	0.029276508		362	11361	0.27804	0.024473374
363	766	0.027176	0.035478198		363	1731	0.052527	0.030344656		363	11861	0.28253	0.02382042
364	740	0.02735	0.036959054		364	1727	0.052803	0.030575043		364	10950	0.26923	0.02458758
365	748	0.027359	0.036575936		365	1732	0.051419	0.02968776		365	11129	0.25347	0.022775541
366	747	0.027887	0.037331995		366	1915	0.053741	0.028062977		366	11971	0.26656	0.02226748
367	775	0.029809	0.038463226		367	1818	0.052759	0.029020077		367	11290	0.25042	0.022180248
368	797	0.028886	0.036243538		368	1874	0.053729	0.028670598		368	11128	0.25301	0.02273661
369	783	0.028918	0.036932439		369	1830	0.054086	0.029554918		369	11672	0.25464	0.021816655
370	749	0.028326	0.037818825		370	1946	0.057676	0.029638335		370	10958	0.3952	0.036064519
371	740	0.029303	0.039597973		371	1787	0.053734	0.030069334		371	11073	0.26825	0.024225323
372	746	0.02956	0.039624263		372	1838	0.054409	0.029602285		372	11675	0.26268	0.022499186
373	794	0.028213	0.03553262		373	1836	0.053801	0.029303595		373	10851	0.27818	0.025635886
374	792	0.027217	0.034364394		374	1827	0.054761	0.02997318		374	11008	0.26733	0.024285065
375	809	0.029938	0.037006304		375	1644	0.051961	0.031606569		375	11570	0.25808	0.022306137
376	810	0.028241	0.034865802		376	1625	0.051717	0.031825908		376	10917	0.24211	0.02217688
377	800	0.029317	0.036646375		377	1634	0.053889	0.032979621		377	10465	0.27716	0.02648409
378	816	0.029559	0.036223652		378	1676	0.051364	0.030646957		378	11212	0.25253	0.022523279
379	816	0.028109	0.034446814		379	1654	0.050827	0.030729504		379	10506	0.25042	0.023835998
380	798	0.029581	0.037069424		380	1618	0.049975	0.030887021		380	10033	0.283	0.028207216
381	819	0.030992	0.037841514		381	1596	0.050405	0.031581892		381	11653	0.27907	0.023948511
382	819	0.028112	0.034324542		382	1569	0.048845	0.031131549		382	10939	0.24343	0.022253405
383	801	0.027612	0.034471411		383	1665	0.050116	0.03009988		383	11323	0.36749	0.032454915
384	697	0.028825	0.041355667		384	1710	0.052219	0.03053731		384	11918	0.29515	0.024765397
385	697	0.028175	0.040423242		385	1597	0.050733	0.03176794		385	11377	0.28903	0.025404676
386	799	0.029003	0.036299124		386	1613	0.049835	0.030896094		386	11266	0.29314	0.026019528
387	753	0.029983	0.039817795		387	1639	0.05113	0.031196095		387	12054	0.30224	0.025073668
388	788	0.027699	0.035150888		388	1599	0.050547	0.03161182		388	11293	0.27548	0.024394227
389	788	0.02935	0.037246574		389	1636	0.050868	0.031092726		389	11283	0.26522	0.023505805
390	797	0.029675	0.037232873		390	1715	0.051002	0.029739009		390	12127	0.26558	0.021900223
391	832	0.030373	0.036505649		391	1662	0.0486	0.029242118		391	11338	0.265	0.023373082
392	885	0.030972	0.03499661		392	1594	0.050777	0.031855207		392	11522	0.25331	0.021984985
393	885	0.027832	0.031448701		393	1659	0.05486	0.033068234		393	12019	0.3133	0.026066811
394	880	0.030532	0.034695341		394	1704	0.052817	0.030995599		394	11151	0.25291	0.022680746
395	880	0.028002	0.031820341		395	1647	0.051576	0.031314876		395	11189	0.26129	0.0233524
396	891	0.031031	0.034826599		396	1752	0.05354	0.030559532		396	12003	0.26485	0.022065484
397	890	0.02889	0.032460225		397	1924	0.053532	0.027823285		397	11136	0.27826	0.024987698
398	890	0.028705	0.032253034		398	1837	0.053402	0.029070005		398	11255	0.2631	0.023376544
399	862	0.029039	0.033688399		399	1701	0.051096	0.030038801		399	12134	0.36848	0.03036715
400	860	0.030801	0.035815465		400	1699	0.050409	0.029669806		400	11218	0.30871	0.027519522
401	872	0.027411	0.03143406		401	1602	0.049714	0.031032522		401	11581	0.2749	0.023736983
402	859	0.02908	0.033853434		402	1666	0.050282	0.030181152		402	12031	0.29895	0.024848142
403	859	0.029997	0.034920838		403	1617	0.048311	0.029876871		403	11491	0.26294	0.022882604
404	855	0.028847	0.033739649		404	1622	0.049534	0.030539088		404	11424	0.25648	0.022451331
405	836	0.028335	0.033893062		405	1605	0.049367	0.030757944		405	12156	0.25723	0.021161073
406	803	0.029308	0.03649863		406	1556	0.049856	0.03204081		406	11324	0.25963	0.022927499
407	817	0.028818	0.035273072		407	1506	0.047237	0.031365936		407	11538	0.26543	0.0230052
408	799	0.028812	0.036060325		408	1581	0.050769	0.032111891		408	12195	0.25895	0.021234194
409	808	0.028783	0.035622525		409	1531	0.049042	0.032032724		409	11423	0.24997	0.021882955
410	719	0.02876	0.040000139		410	1548	0.047893	0.030938307		410	11212	0.27023	0.024102123
411	758	0.028578	0.037701715		411	1562	0.049187	0.031489565		411	12063	0.40306	0.033412833
412	762	0.026735	0.035085827		412	1572	0.049113	0.031242494		412	11196	0.25988	0.023211861
413	849	0.030527	0.035956655		413	1539	0.049623	0.0322436		413	11395	0.27265	0.023927161
414	849	0.027253	0.032100471		414	1454	0.047484	0.032657497		414	12193	0.29936	0.02455212
415	853	0.027788	0.032577022		415	1583	0.049058	0.030990208		415	11926	0.29008	0.024323159
416	799	0.028776	0.036014894		416	1528	0.048475	0.031724607		416	11855	0.28721	0.024227246
417	799	0.027586	0.034525031		417	1540	0.048306	0.031367338		417	12451	0.28741	0.023083046
418	716	0.028818	0.040248743		418	1571	0.049534	0.03153049		418	11641	0.28619	0.024584658
419	804	0.030699	0.038182338		419	1546	0.047294	0.030590944		419	11681	0.28097	0.02405342
420	805	0.026086	0.032405342		420	1626	0.051697	0.031793665		420	12190	0.28119	0.023067104
421	802	0.028248	0.03522207		421	1637	0.049244	0.030082101		421	11329	0.27702	0.024452202
422	767	0.029365	0.038284876		422	1605	0.050367	0.031381184		422	11418	0.27631	0.024199772
423	733	0.028766	0.039243656		423	1692	0.049939	0.029514953		423	12526	0.27771	0.022170605
424	738	0.028303	0.038350407		424	1642	0.049245	0.029990987		424	11471	0.2518	0.02195092
425	767	0.028394	0.037019296		425	1592	0.050235	0.031554523		425	11613	0.25422	0.021890984
426	756	0.027595	0.036500794		426	1882	0.050771	0.026976886		426	12390	0.25718	0.020757224
427	808	0.028451	0.035211262		427	1920	0.052376	0.027279375		427	12131	0.25043	0.020643805
428	805	0.027547	0.034220124		428	1974	0.052147	0.026416667		428	11650	0.24985	0.021446009
429	807	0.02836	0.035142379		429	1985	0.052449	0.02642272		429	12287	0.25418	0.020687068
430	798	0.027716	0.034732206		430	1793	0.051693	0.028830229		430	11894	0.25826	0.021713301
431	811	0.029641	0.036549199		431	1650	0.048816	0.029585273		431	12156	0.25974	0.021367308
432	821	0.029572	0.036019367		432	1649	0.051491	0.031225652		432	13051	0.26668	0.020433913
433	821	0.028023	0.03413313		433	1696	0.050289	0.029651238		433	11992	0.29225	0.024370664
434	830	0.02995	0.036084819		434	1746	0.051662	0.029588889		434	11916	0.28768	0.024142665
435	905	0.029245	0.032315138		435	1722	0.049876	0.028964053		435	12865	0.27192	0.021136106
436	905	0.026799	0.029612597		436	1633	0.050878	0.031156154		436	11775	0.25514	0.02166828
437	874	0.029505	0.03375881		437	1765	0.052487	0.029737507		437	11545	0.31859	0.027595409
438	878	0.031078	0.035396697		438	1686	0.05079	0.030124733		438	12383	0.3859	0.031163369
439	879	0.027272	0.031025597		439	1736	0.052074	0.029996659		439	11344	0.26469	0.023332951
440	965	0.030681	0.031793264		440	1866	0.052647	0.028213666		440	11567	0.28886	0.024973113
441	937	0.031098	0.033188474		441	1764	0.056092	0.031798413		441	12612	0.29674	0.023528624
442	930	0.032222	0.034647312		442	1680	0.051401	0.030595833		442	11544	0.2934	0.025415367
443	821	0.029008	0.035332278		443	1724	0.051998	0.030161485		443	11899	0.28521	0.023969493
444	861	0.030447	0.035362834		444	1786	0.053554	0.029985274		444	12427	0.28873	0.023234248
445	888	0.029276	0.032968919		445	1932	0.053242	0.027557971		445	11684	0.28138	0.024082677
446	939	0.031105	0.033125453		446	1817	0.051217	0.028187397		446	12161	0.27946	0.022980018
447	937	0.028615	0.030539061		447	1845	0.051772	0.028060705		447	12691	0.28054	0.022105508
448	890	0.031283	0.035148876		448	1782	0.051805	0.029071156		448	11880	0.27571	0.023207828
449	849	0.028685	0.033787279		449	1755	0.050989	0.029053561		449	11791	0.28671	0.024315664
450	718	0.02826	0.039359749		450	1784	0.053154	0.029794787		450	12301	0.27332	0.022219088
451	743	0.028992	0.039020727		451	1848	0.052605	0.028466017		451	11586	0.25886	0.022342482
452	743	0.028515	0.038378331		452	1862	0.052286	0.028080559		452	11233	0.24501	0.02181127
453	737	0.028576	0.038772863		453	1795	0.05316	0.02961571		453	12244	0.25085	0.020487177
454	798	0.029003	0.036345113		454	1860	0.053233	0.028619839		454	11084	0.24782	0.022358625
455	789	0.028156	0.035686185		455	1923	0.054281	0.028227249		455	11036	0.24578	0.02227075
456	759	0.028062	0.036971805		456	2037	0.057255	0.028107707		456	12101	0.25059	0.020708289
457	769	0.029062	0.037792328		457	2081	0.059574	0.028627439		457	11338	0.27547	0.024295996
458	770	0.027417	0.035605844		458	2066	0.053661	0.025973282		458	10880	0.24359	0.022388327
459	859	0.028497	0.033174156		459	2063	0.05354	0.025952254		459	11651	0.26514	0.022756759
460	852	0.029881	0.035071479		460	2115	0.05692	0.026912719		460	11081	0.25762	0.023248714

				Avg time per obj over all frames:			0.029803515	Standard Deviation:	0.00606977		
Helicopter Video (640 x 360)				Dashboard Video (848 x 480)				Drone Video (1920 x 1080)			
Frame No	Num Obj	Time in S	Time per obj (ms)	Frame No	Num Obj	Time in S	Time per obj (ms)	Frame No	Num Obj	Time in S	Time per obj (ms)
461	852	0.033904	0.039793662	461	2033	0.054575	0.026844368	461	10812	0.26965	0.024939419
462	789	0.029273	0.037101267	462	1990	0.053762	0.027015829	462	11591	0.26658	0.022999224
463	789	0.027073	0.034312801	463	2019	0.054594	0.027040267	463	10510	0.24802	0.023598287
464	778	0.027673	0.035568766	464	1947	0.054807	0.028149563	464	11118	0.29036	0.026116028
465	820	0.029067	0.035448049	465	1864	0.054744	0.029369313	465	11701	0.26715	0.02283104
466	820	0.02817	0.034353415	466	1955	0.053854	0.027546803	466	10871	0.24382	0.022428479
467	719	0.027674	0.038490125	467	2018	0.054575	0.027044252	467	11083	0.36276	0.03273121
468	720	0.027104	0.037644444	468	2048	0.056984	0.027824219	468	11806	0.26809	0.022707776
469	783	0.028096	0.035882503	469	1961	0.054829	0.02795951	469	10880	0.24807	0.022800368
470	743	0.028386	0.038204441	470	2038	0.055556	0.027259814	470	10588	0.28015	0.026458916
471	743	0.027801	0.037417631	471	2019	0.054429	0.026958296	471	11826	0.31127	0.02632048
472	893	0.031024	0.034741097	472	1951	0.05571	0.028554587	472	11049	0.2826	0.025576704
473	958	0.030143	0.031464718	473	1981	0.055546	0.028039576	473	11427	0.29011	0.025388291
474	960	0.028052	0.029220521	474	1892	0.05413	0.028609778	474	12022	0.34259	0.028496922
475	978	0.030943	0.031639264	475	1822	0.052566	0.028850549	475	11021	0.28392	0.025761728
476	965	0.030119	0.031211088	476	1804	0.053572	0.029695953	476	11906	0.29797	0.025026541
477	948	0.029282	0.030887658	477	1931	0.053571	0.027742465	477	11178	0.27357	0.024474146
478	968	0.029574	0.03055155	478	2039	0.056397	0.027659343	478	11054	0.28826	0.026077076
479	993	0.030757	0.030973515	479	1848	0.05353	0.02896645	479	11822	0.2798	0.023667484
480	988	0.0304	0.030769231	480	1907	0.055906	0.029315941	480	10872	0.26395	0.02427787
481	984	0.030437	0.030931606	481	1939	0.055073	0.02840263	481	11255	0.26704	0.0237259
482	981	0.028555	0.029108053	482	1928	0.053327	0.027659025	482	12175	0.30127	0.024745216
483	936	0.030235	0.032302564	483	1916	0.053142	0.027735752	483	11031	0.25664	0.023265524
484	938	0.028623	0.030514499	484	1946	0.054277	0.027891418	484	10944	0.25563	0.023358279
485	967	0.030035	0.03106029	485	1935	0.054625	0.028229819	485	11550	0.2671	0.023125368
486	1021	0.03181	0.031155632	486	1937	0.053399	0.02756763	486	12154	0.25708	0.021151966
487	1021	0.028361	0.027778061	487	1887	0.052145	0.027633969	487	11324	0.31937	0.028202755
488	990	0.031951	0.032273333	488	1927	0.054249	0.02815205	488	11253	0.25374	0.022548831
489	961	0.031493	0.032771176	489	1956	0.053274	0.027235992	489	11789	0.26431	0.022420053
490	959	0.029231	0.030481126	490	1903	0.054037	0.028395849	490	11028	0.32727	0.029676369
491	955	0.031547	0.033033717	491	1957	0.052814	0.026987123	491	10916	0.36846	0.033754397
492	940	0.0303	0.032234468	492	1918	0.055569	0.028972106	492	11461	0.2603	0.022711456
493	939	0.028194	0.030025453	493	1850	0.053249	0.028783081	493	10485	0.24751	0.023605627
494	971	0.031742	0.032690422	494	1795	0.052922	0.029483231	494	10943	0.24663	0.022537969
495	966	0.030996	0.032086646	495	1758	0.055421	0.031524915	495	11777	0.25073	0.021289547
496	927	0.029792	0.032137648	496	1757	0.05218	0.029698179	496	10988	0.26669	0.024270841
497	877	0.030579	0.034867161	497	1709	0.052005	0.030430135	497	11512	0.25737	0.022356411
498	1010	0.030817	0.03051198	498	2011	0.054455	0.027078667	498	11966	0.26108	0.021818653
499	982	0.030386	0.030943075	499	2025	0.052769	0.026058815	499	11395	0.25391	0.022282492
500	934	0.029653	0.031748715	500	1963	0.052019	0.026499694	500	11300	0.24568	0.021741947
501	937	0.028995	0.03094461	501	1965	0.051294	0.026103613	501	12093	0.24835	0.020536426
502	933	0.028399	0.0304388	502	1899	0.050424	0.026552712	502	11174	0.24546	0.021967335
503	937	0.028764	0.030698399	503	1672	0.057411	0.034336543	503	11553	0.24618	0.021308924
504	900	0.030506	0.033895333	504	1677	0.052717	0.031435063	504	12177	0.26881	0.022074813
505	971	0.029106	0.029974974	505	1607	0.049251	0.030647604	505	11160	0.26955	0.024153047
506	962	0.02811	0.029220166	506	1622	0.049295	0.030391245	506	11210	0.25087	0.022378858
507	930	0.029138	0.03133086	507	1545	0.048331	0.031282201	507	11515	0.25834	0.02243465
508	877	0.03033	0.034583352	508	1513	0.048527	0.0320731	508	10785	0.35292	0.032723227
509	878	0.027976	0.031863212	509	1418	0.048846	0.034446968	509	10948	0.2471	0.022569876
510	934	0.030916	0.033100107	510	1372	0.047233	0.034426166	510	11607	0.26288	0.022648574
511	920	0.030223	0.032851522	511	1465	0.04883	0.033331331	511	10876	0.30218	0.027783744
512	922	0.026852	0.029123536	512	1501	0.048576	0.032362092	512	10929	0.28041	0.025657425
513	891	0.028881	0.032413692	513	1466	0.048035	0.032765825	513	11930	0.29178	0.024457837
514	938	0.030963	0.033009915	514	1461	0.04834	0.033086721	514	11034	0.39475	0.035775965
515	854	0.02868	0.033583021	515	1381	0.049109	0.035560681	515	10656	0.27042	0.025376877
516	855	0.030775	0.035993567	516	1616	0.050567	0.031291708	516	11654	0.27696	0.023764973
517	866	0.028562	0.032981178	517	1531	0.047074	0.030747159	517	10718	0.26598	0.024816104
518	872	0.029443	0.033764564	518	1391	0.046977	0.033772178	518	10680	0.2683	0.025121348
519	882	0.029809	0.033797052	519	1418	0.051159	0.036077927	519	11536	0.26995	0.023400745
520	886	0.027638	0.031193792	520	1417	0.046952	0.033134651	520	10862	0.26353	0.024261278
521	902	0.029397	0.032590798	521	1424	0.04703	0.033026475	521	10887	0.27072	0.024866079
522	902	0.028606	0.031713525	522	1481	0.046378	0.031315463	522	11646	0.26497	0.02275219
523	868	0.029713	0.034231106	523	1394	0.049378	0.035422023	523	10896	0.25107	0.023042768
524	903	0.032562	0.036059468	524	1434	0.046448	0.032390307	524	10971	0.24277	0.022128338
525	898	0.028198	0.03140078	525	1491	0.048803	0.032731522	525	11618	0.24682	0.02124419
526	875	0.028928	0.033060114	526	1485	0.046694	0.031443973	526	10749	0.24844	0.023112941
527	874	0.031194	0.035690618	527	1545	0.049163	0.031820971	527	11154	0.2432	0.021803748
528	875	0.028273	0.032312343	528	1627	0.052817	0.032462999	528	11493	0.24798	0.021576699
529	881	0.028887	0.032788309	529	1602	0.048779	0.030448564	529	10417	0.25795	0.024762504
530	844	0.029648	0.035128199	530	1630	0.050133	0.030756135	530	10454	0.24026	0.022982495
531	920	0.030615	0.033277065	531	1426	0.046317	0.032480084	531	11303	0.24427	0.021611431
532	900	0.031195	0.034660889	532	1546	0.047857	0.030955304	532	10414	0.2399	0.023036393
533	847	0.029309	0.03460307	533	1581	0.04649	0.029405693	533	10407	0.23911	0.022975593
534	868	0.028599	0.032948041	534	1484	0.047462	0.031982412	534	11214	0.24065	0.021459782
535	866	0.030777	0.035539145	535	1534	0.054706	0.035662125	535	10181	0.2501	0.024564974
536	867	0.026555	0.030628374	536	1701	0.049736	0.029239153	536	10352	0.27511	0.026575638
537	938	0.031384	0.033458529	537	1514	0.049385	0.032618626	537	11155	0.25848	0.023171762
538	869	0.031373	0.036102877	538	1714	0.049646	0.028965111	538	10714	0.24172	0.022561135
539	869	0.026977	0.031043959	539	1728	0.05079	0.029392303	539	10637	0.25257	0.023744383
540	915	0.031014	0.033894536	540	1689	0.0523	0.03096495	540	11255	0.27087	0.024066726
541	915	0.029179	0.031889836	541	1822	0.050552	0.027745554	541	10473	0.24675	0.02356068
542	896	0.028175	0.031445313	542	1774	0.049607	0.027963191	542	10629	0.24419	0.022973751
543	954	0.029843	0.031281866	543	1777	0.052393	0.029483737	543	11116	0.27195	0.024465005
544	954	0.027203	0.02851478	544	1736	0.050481	0.02907909	544	10506	0.26114	0.024856463
545	925	0.028672	0.030996757	545	1709	0.0497	0.029081042	545	10251	0.2434	0.023743537
546	941	0.028812	0.030618278	546	1663	0.050168	0.030167168	546	11161	0.3861	0.034593316
547	941	0.029127	0.030952922	547	1704	0.052298	0.030691432	547	10120	0.27847	0.027516897
548	975	0.0316	0.032410462	548	1827	0.052034	0.028480296	548	10229	0.27778	0.027156223
549	948	0.029577	0.031199051	549	1818	0.053204	0.029264961	549	10873	0.29007	0.02667755
550	934	0.029169	0.031229764	550	1920	0.051605	0.026877552	550	10367	0.27918	0.026929488
551	1019	0.030874	0.030298724	551	1782	0.057211	0.032105051	551	10092	0.27475	0.027224237
552	1017	0.030589	0.030077483	552	1733	0.053027	0.030598211	552	11332	0.27335	0.024121603
553	945	0.029146	0.030842328	553	1616	0.052473	0.032470792	553	10376	0.2796	0.026946415
554	908	0.02908	0.032026542	554	1568	0.049176	0.03136199	554	10275	0.26596	0.025884574
555	909	0.027458	0.030207151	555	1540	0.04905	0.031850649	555	10705	0.26836	0.02506894
556	943	0.029711	0.031507317	556	1502	0.049655	0.033058921	556	8364	0.21919	0.026206002
557	941	0.028152	0.029917216	557	1473	0.046467	0.031545621	557	6888	0.2165	0.031430749
558	953	0.029417	0.030868206	558	1475	0.048083	0.032598847	558	7487	0.21338	0.028499933
559	876	0.028945	0.033042237	559	1464	0.047463	0.03242015	559	7089	0.21544	0.030391169
560	877	0.029089	0.033168529	560	1469	0.048666	0.033128931	560	6750	0.21125	0.031296
561	903	0.029005	0.032120709	561	1489	0.046096	0.030957757	561	7959	0.217	0.027265234
562	902	0.030135	0.033408537	562	1463	0.046757	0.031959467	562	7818	0.23166	0.029631875
563	905	0.027757	0.03067116	563	1462	0.047001	0.032148564	563	8016	0.22506	0.028076347
564	930	0.029236	0.031436667	564	1495	0.049314	0.032985619	564	8525	0.21921	0.025713314
565	913	0.029276	0.032065717	565	1615	0.048525	0.03004644	565	7949	0.23123	0.029088816
566	914	0.027477	0.030062691	566	1564	0.047283	0.030232033	566	8306	0.22958	0.027640501
567	908	0.028518	0.031406938	567	1563	0.049223	0.03149245	567	8632	0.22543	0.026115964
568	923	0.030296	0.032823077	568	1537	0.047755	0.031070007	568	8049	0.2335	0.029009566
569	947	0.028119	0.029693031	569	1534	0.04871	0.031753259	569	8140	0.25158	0.030907125
570	964	0.029077	0.030162344	570	1574	0.048831	0.03102338	570	8532	0.22603	0.026491678
571	977	0.029303	0.029993142	571	1566	0.049149	0.03138493	571	7988	0.21279	0.026639084
572	952	0.030223	0.031746429	572	1720	0.051354	0.029856977	572	8028	0.22448	0.027961634
573	931	0.029277	0.031447046	573	1701	0.049393	0.029037801	573	7978	0.21684	0.027179118
574	925	0.027418	0.029640649	574	1748	0.051329	0.029364474	574	7306	0.21463	0.029377224
575	930	0.029617	0.031846237	575	1636	0.050294	0.03074187	575	7070	0.21414	0.030288967

				Avg time per obj over all frames:				0.029803515	Standard Deviation:		0.00606977		
Helicopter Video (640 x 360)					Dashboard Video (848 x 480)					Drone Video (1920 x 1080)			
Frame No	Num Obj	Time in S	Time per obj (ms)		Frame No	Num Obj	Time in S	Time per obj (ms)		Frame No	Num Obj	Time in S	Time per obj (ms)
576	927	0.028302	0.030530529		576	1819	0.054638	0.030037493		576	7556	0.30601	0.040498412
577	908	0.029651	0.032655617		577	1766	0.052271	0.029598301		577	6904	0.20606	0.029846031
578	896	0.029288	0.032687388		578	1654	0.05095	0.030803809		578	7237	0.21415	0.029590991
579	902	0.027439	0.030420177		579	1638	0.050811	0.031020269		579	7407	0.21185	0.028601188
580	985	0.030322	0.030783452		580	1707	0.050564	0.0296215		580	6839	0.22179	0.032430619
581	932	0.030792	0.033038948		581	1574	0.049005	0.031133799		581	6956	0.24769	0.035608683
582	935	0.028237	0.030200428		582	1579	0.048925	0.030984927		582	7293	0.24689	0.033853558
583	971	0.029883	0.030775283		583	1442	0.051753	0.035890014		583	6700	1.2467	0.186074627
584	975	0.029802	0.030565949		584	1571	0.051646	0.032874602		584	6940	0.2512	0.036195533
585	885	0.030141	0.034057401		585	1475	0.048528	0.032900407		585	7206	0.25211	0.034986261
586	870	0.029602	0.034025632		586	1380	0.047836	0.034663768		586	6750	0.24517	0.036320889
587	928	0.031217	0.033639009		587	1479	0.048808	0.033000541		587	7211	0.25661	0.035586465
588	864	0.028999	0.033563773		588	1514	0.052409	0.03461638		588	7757	0.25902	0.033391388
589	919	0.029675	0.032290533		589	1612	0.047334	0.029363462		589	6733	0.24593	0.036526066
590	919	0.02774	0.030184548		590	1437	0.046687	0.032489353		590	6565	0.24251	0.036939832
591	882	0.030093	0.034118594		591	1436	0.047534	0.033101532		591	7568	0.25122	0.0331949
592	835	0.028407	0.03401976		592	1450	0.046584	0.032127172		592	6574	0.25545	0.038857925
593	831	0.027608	0.033222262		593	1366	0.046234	0.033846559		593	6828	0.25548	0.037416374
594	880	0.029131	0.033102955		594	1436	0.046787	0.032581546		594	6898	0.25593	0.037101769
595	879	0.027305	0.031064164		595	1449	0.047144	0.03253568		595	6439	0.26319	0.040874049
596	818	0.029227	0.035730318		596	1465	0.047412	0.032363208		596	6507	0.24465	0.037597664
597	732	0.027853	0.03805		597	1465	0.046527	0.031758908		597	6807	0.24593	0.036128838
598	732	0.025028	0.034191667		598	1501	0.046486	0.03096982		598	6359	0.24323	0.038249253
599	742	0.02701	0.036400943		599	1414	0.046386	0.032804597		599	6419	0.25523	0.039761957
600	827	0.028862	0.034900121		600	1500	0.048957	0.032637867		600	7052	0.26701	0.037863585
601	826	0.027461	0.0332454		601	1579	0.046692	0.029570804		601	6222	0.2485	0.039939569
602	846	0.031037	0.036686998		602	1651	0.047725	0.028906723		602	6637	0.25877	0.038989152
603	855	0.029171	0.034117544		603	1648	0.050232	0.030480583		603	6763	0.31632	0.046771403
604	851	0.029051	0.03413772		604	1605	0.049186	0.03064567		604	6098	0.24019	0.039388816
605	748	0.027817	0.037188102		605	1476	0.046953	0.031810976		605	6110	0.2469	0.040408347
606	781	0.02827	0.036196927		606	1428	0.046528	0.032582283		606	6327	0.23793	0.037605658
607	785	0.028258	0.035997834		607	1463	0.047122	0.032208954		607	5966	0.23058	0.038649011
608	786	0.028939	0.036817557		608	1442	0.046803	0.032456657		608	5933	0.2278	0.038395247
609	779	0.027681	0.035534275		609	1467	0.047009	0.032044104		609	6198	0.22633	0.036516457
610	799	0.030359	0.03799612		610	1531	0.047014	0.030707969		610	5622	0.22588	0.040178051
611	799	0.027082	0.033895119		611	1467	0.049634	0.033833606		611	6409	0.22745	0.035488376
612	798	0.028444	0.035643609		612	1707	0.051096	0.02993345		612	6066	0.22311	0.036780415
613	938	0.02927	0.031204797		613	1689	0.048983	0.029001362		613	5784	0.22695	0.039237206
614	939	0.028518	0.030370075		614	1688	0.049477	0.029311078		614	6042	0.22571	0.037357001
615	964	0.029714	0.030823755		615	1634	0.051469	0.031498776		615	5478	0.21736	0.039678897
616	1016	0.031054	0.030564764		616	1653	0.050321	0.030442166		616	5510	0.21907	0.039758439
617	1015	0.02821	0.027792808		617	1681	0.049453	0.02941856		617	5809	0.22216	0.038244448
618	886	0.029659	0.033475395		618	1712	0.050581	0.029544918		618	5807	0.22187	0.038207164
619	880	0.029038	0.032997955		619	1469	0.046984	0.031983526		619	5316	0.21808	0.041022573
620	882	0.027438	0.031108617		620	1572	0.050677	0.032237468		620	5665	0.22129	0.039062489
621	1111	0.032257	0.029034293		621	1556	0.048711	0.031305527		621	5891	0.22393	0.038012052
622	1131	0.031213	0.027597701		622	1558	0.047198	0.03029371		622	5666	0.21874	0.038606248
623	1137	0.031057	0.027314512		623	1561	0.049132	0.031474824		623	5805	0.22119	0.038103704
624	1082	0.030868	0.028528373		624	1688	0.049697	0.029441173		624	5655	0.22271	0.039382317
625	1083	0.029944	0.027648846		625	1523	0.047669	0.031299606		625	5193	0.21515	0.04143058
626	1075	0.032302	0.030048558		626	1689	0.04872	0.028845293		626	5188	0.22239	0.042865459
627	945	0.030027	0.031775026		627	1869	0.053778	0.028773515		627	5474	0.21788	0.039803252
628	985	0.028086	0.028513807		628	1905	0.055415	0.029089344		628	5182	0.21554	0.041594558
629	984	0.030694	0.031192988		629	1763	0.051242	0.029065343		629	5351	0.21968	0.041054195
630	980	0.028949	0.029540102		630	1683	0.052039	0.03092038		630	5465	0.21804	0.039897164
631	1010	0.029728	0.029433267		631	1685	0.055369	0.032860178		631	5416	0.22266	0.041111152
632	1045	0.03111	0.029770239		632	1658	0.052444	0.031630639		632	5298	0.22578	0.042616082
633	1045	0.029143	0.027888325		633	1719	0.053867	0.031336475		633	5526	0.21627	0.039136446
634	889	0.030659	0.034487177		634	1697	0.051693	0.030461167		634	5128	0.21362	0.041657371
635	934	0.030301	0.03244197		635	1772	0.054179	0.030574887		635	5102	0.21962	0.043045276
636	934	0.028308	0.03030803		636	1805	0.057489	0.031849806		636	5479	0.29692	0.054192918
637	923	0.029665	0.032139545		637	1830	0.055831	0.030508634		637	5188	0.26381	0.050850039
638	890	0.03033	0.034079101		638	1829	0.056586	0.030938327		638	5749	0.21708	0.037758741
639	1046	0.03085	0.02949283		639	1772	0.057559	0.032482619		639	5704	0.21586	0.037844144
640	861	0.029117	0.033817422		640	1550	0.055543	0.035833871		640	5286	0.21352	0.040393681
641	947	0.031011	0.032746674		641	1589	0.054588	0.034353682		641	5215	0.21443	0.041118313
642	957	0.028666	0.029954023		642	1828	0.05693	0.03114349		642	5543	0.22495	0.040582717
643	916	0.029626	0.032342358		643	1602	0.054817	0.034217665		643	5335	0.21495	0.040289597
644	916	0.028119	0.030697489		644	1720	0.057024	0.033153198		644	5627	0.22132	0.03933126
645	923	0.029785	0.032269231		645	1798	0.055209	0.030705784		645	5839	0.22228	0.038067477
646	896	0.028884	0.032236719		646	1691	0.062987	0.037248374		646	5296	0.21471	0.040542107
647	911	0.028129	0.030876839		647	1513	0.052911	0.034970654		647	5349	0.2171	0.040587026
648	986	0.032243	0.032700304		648	1740	0.057655	0.033134943		648	5726	0.21355	0.037294446
649	992	0.028885	0.02911744		649	1715	0.055967	0.032633644		649	5225	0.21701	0.041532823
650	990	0.029863	0.030164343		650	1694	0.055162	0.0325634		650	5260	0.21514	0.040901901
651	1022	0.031387	0.030711546		651	1615	0.057295	0.035476718		651	6266	0.22559	0.036001915
652	1021	0.029365	0.028761312		652	1628	0.055158	0.033880651		652	5201	0.21981	0.042263603
653	1046	0.031066	0.029699809		653	1792	0.055778	0.03112606		653	5451	0.21819	0.040027151
654	1126	0.031905	0.028334902		654	1670	0.055899	0.033472335		654	5719	0.21295	0.037235181
655	1126	0.029017	0.02577016		655	1665	0.055211	0.03315976		655	5273	0.21617	0.040995828
656	1090	0.032052	0.029405229		656	1721	0.056242	0.032679895		656	5364	0.21387	0.039871365
657	1119	0.033594	0.030021805		657	1742	0.058098	0.033351378		657	5628	0.22539	0.040047264
658	1109	0.031578	0.028474301		658	1702	0.055104	0.032375969		658	5445	0.21442	0.039379247
659	1121	0.03321	0.029625335		659	1750	0.053163	0.030379086		659	5601	0.22893	0.040873058
660	1201	0.033439	0.027842465		660	1680	0.056741	0.033774464		660	5704	0.21999	0.038567146
661	1169	0.032602	0.027889136		661	1789	0.063624	0.035564114		661	5480	0.21866	0.039900547
662	1083	0.032704	0.030197969		662	1549	0.050402	0.032538347		662	5510	0.21877	0.039703811
663	1081	0.029445	0.02723839		663	1624	0.052894	0.032569951		663	5758	0.21775	0.037816777
664	1105	0.032422	0.029341448		664	1564	0.051067	0.032651407		664	5422	0.21344	0.039366101
665	1104	0.029807	0.026998732		665	1442	0.050697	0.035157143		665	5581	0.21886	0.039215732
666	1229	0.033792	0.027495525		666	1456	0.048643	0.033408723		666	6231	0.232	0.037232868
667	1142	0.031866	0.027903327		667	1462	0.048627	0.03326026		667	5625	0.2154	0.038293333
668	1139	0.029744	0.02611396		668	1461	0.049591	0.033942847		668	5589	0.22586	0.04041188
669	1150	0.031905	0.027743565		669	1651	0.050642	0.030673349		669	5961	0.26195	0.043944305
670	1131	0.033075	0.02924359		670	1548	0.050132	0.032385078		670	5620	0.21853	0.038885053
671	1137	0.030268	0.02662058		671	1545	0.049472	0.032020388		671	5704	0.22119	0.038778226
672	1122	0.032913	0.029333868		672	1604	0.054001	0.033666147		672	6255	0.23636	0.03778705
673	1155	0.033241	0.028779913		673	1702	0.056256	0.033053055		673	5749	0.21622	0.037609149
674	1189	0.030303	0.025485955		674	1734	0.055726	0.03213737		674	5847	0.21691	0.037096973
675	1154	0.033427	0.028966118		675	1677	0.054225	0.032334466		675	6088	0.21394	0.035141919
676	1159	0.031116	0.02684711		676	1624	0.051556	0.03174649		676	5738	0.20257	0.035303939
677	1148	0.033406	0.029098955		677	1707	0.052527	0.030771705		677	5740	0.20739	0.036130139
678	1207	0.03302	0.027357084		678	1654	0.050951	0.030804958		678	6080	0.20959	0.034471875
679	1200	0.031742	0.026452		679	1811	0.054993	0.030366041		679	5816	0.20958	0.036034388
680	1204	0.031701	0.026329983		680	1667	0.051413	0.030841572		680	5705	0.20542	0.036006135
681	1175	0.032451	0.027617957		681	1797	0.051738	0.028791096		681	6004	0.21404	0.035650067
682	1216	0.033589	0.027622451		682	1518	0.048977	0.032264361		682	5676	0.20253	0.035682523
683	1236	0.032203	0.026054531		683	1683	0.051727	0.030735175		683	5623	0.19662	0.034967633
684	1240	0.031099	0.025079677		684	1757	0.054573	0.03106033		684	5967	0.20538	0.034418468
685	1228	0.033442	0.027233143		685	1762	0.052295	0.029679569		685	5693	0.20043	0.035206218
686	1223	0.029678	0.024266394		686	1743	0.054204	0.031098107		686	5572	0.19944	0.035793252
687	1248	0.033566	0.026895833		687	1713	0.050669	0.029578867		687	6059	0.20118	0.033204159
688	1221	0.032766	0.026835053		688	1728	0.050117	0.02900272		688	5671	0.2002	0.035302592
689	1167	0.030933	0.026506341		689	1704	0.05068	0.029742019		689	5715	0.20395	0.035686439
690	1212	0.032489	0.026806436		690	1699	0.050074	0.029472866		690	6099	0.20295	0.033275947

				Avg time per obj over all frames:			0.029803515	Standard Deviation:	0.00606977		
Helicopter Video (640 x 360)				Dashboard Video (848 x 480)				Drone Video (1920 x 1080)			
Frame No	Num Obj	Time in S	Time per obj (ms)	Frame No	Num Obj	Time in S	Time per obj (ms)	Frame No	Num Obj	Time in S	Time per obj (ms)
691	1178	0.033411	0.028362224	691	1711	0.050586	0.0295654	691	5800	0.20074	0.034611034
692	1177	0.028783	0.024454206	692	1846	0.0511	0.027681257	692	5996	0.19893	0.033176451
693	1270	0.030816	0.024264567	693	1737	0.057229	0.032946805	693	6226	0.20464	0.032868615
694	1272	0.033418	0.02627217	694	1762	0.052071	0.029552043	694	5770	0.19591	0.033953206
695	1258	0.031374	0.024939428	695	1888	0.052452	0.027781727	695	5914	0.24692	0.041751945
696	1187	0.033033	0.027828981	696	1827	0.053211	0.029124685	696	6356	0.20622	0.032444934
697	1247	0.033024	0.026482438	697	1634	0.05009	0.030654712	697	5879	0.2056	0.034972614
698	1239	0.033154	0.026758918	698	1787	0.053009	0.029663402	698	6093	0.21217	0.034821763
699	1224	0.034402	0.028105801	699	1652	0.049937	0.030228148	699	6393	0.20869	0.032643047
700	1260	0.031405	0.024924286	700	1887	0.051125	0.027093005	700	6131	0.21058	0.034346436
701	1213	0.033342	0.027487552	701	1584	0.049323	0.031138258	701	6096	0.2095	0.034366306
702	1272	0.031714	0.024931997	702	1609	0.05055	0.031417091	702	6423	0.21098	0.03284789
703	1264	0.032656	0.025835601	703	1696	0.051541	0.030389446	703	6077	0.21544	0.035451703
704	1201	0.034196	0.028472689	704	1838	0.050832	0.027656202	704	6056	0.23804	0.039307133
705	1203	0.03208	0.0266665	705	1670	0.050229	0.030076946	705	6419	0.22901	0.035677053
706	1209	0.033295	0.02753962	706	1810	0.05161	0.028513812	706	6103	0.2037	0.033376372
707	1240	0.033867	0.027312419	707	1731	0.050176	0.028986655	707	6173	0.20866	0.033802527
708	1239	0.030906	0.024944633	708	2024	0.062915	0.031084338	708	6816	0.21715	0.031858421
709	1215	0.032781	0.026979918	709	1744	0.051655	0.029618693	709	6311	0.20641	0.032705593
710	1230	0.03342	0.027170894	710	1650	0.051589	0.031266121	710	6418	0.20575	0.032057806
711	1230	0.030087	0.024461057	711	1715	0.049894	0.029092653	711	6731	0.20735	0.030804932
712	1213	0.03352	0.0276338	712	1652	0.050489	0.030562228	712	6491	0.20842	0.032108766
713	1257	0.033039	0.026284328	713	1628	0.050475	0.031004054	713	6443	0.21187	0.03288375
714	1251	0.030605	0.024464428	714	1709	0.052865	0.030933353	714	6918	0.20645	0.029842874
715	1279	0.03417	0.026716185	715	1642	0.051137	0.031143179	715	6623	0.20641	0.03116488
716	1209	0.033771	0.027933002	716	1610	0.051429	0.031943292	716	6400	0.21262	0.033221406
717	1221	0.03334	0.027305815	717	1745	0.051814	0.029693009	717	7102	0.2465	0.034709096
718	1202	0.033363	0.027756156	718	1638	0.051315	0.031327778	718	6568	0.20986	0.03195204
719	1205	0.031357	0.026022241	719	1636	0.050882	0.03110165	719	6750	0.21	0.031111556
720	1200	0.032925	0.027437333	720	1785	0.054386	0.030468179	720	6940	0.21426	0.030872478
721	1244	0.033898	0.027249035	721	1766	0.054227	0.030706059	721	6744	0.20339	0.030159104
722	1237	0.031929	0.025811803	722	1758	0.053105	0.030207565	722	6644	0.20023	0.030136213
723	1238	0.033453	0.027021809	723	1845	0.052181	0.028282547	723	6857	0.2072	0.030216713
724	1240	0.030919	0.024934758	724	1855	0.060112	0.032405445	724	6587	0.20833	0.031627144
725	1126	0.031544	0.028013943	725	1660	0.051118	0.030793855	725	6862	0.20889	0.030441416
726	1223	0.034128	0.027904988	726	1782	0.051804	0.029070483	726	6976	0.20992	0.0300916
727	1223	0.030933	0.025292968	727	1818	0.050206	0.027615842	727	6745	0.2114	0.031341142
728	1219	0.033593	0.027557998	728	1811	0.052124	0.028781888	728	6650	0.21539	0.032389323
729	1089	0.031924	0.02931506	729	1757	0.051414	0.029262208	729	6934	0.21144	0.030493222
730	1089	0.029865	0.027424334	730	1791	0.051869	0.028961027	730	6535	0.20673	0.031633818
731	1107	0.032725	0.029561789	731	1659	0.050665	0.030539482	731	6606	0.20413	0.030900999
732	1133	0.03305	0.029170344	732	1854	0.053545	0.028880798	732	6770	0.20733	0.030625406
733	1108	0.032333	0.029181047	733	1874	0.052451	0.027988634	733	6571	0.20159	0.030679196
734	1103	0.031201	0.028287217	734	1930	0.052852	0.027384508	734	6548	0.21036	0.032125687
735	1094	0.031535	0.028825046	735	1834	0.052058	0.028385169	735	6833	0.20404	0.029860822
736	1161	0.03296	0.02838932	736	1897	0.052211	0.027522667	736	6521	0.20095	0.030816132
737	1160	0.033318	0.028722328	737	1895	0.052815	0.027870501	737	6485	0.21796	0.033609869
738	1161	0.030001	0.02584031	738	1731	0.051226	0.02959353	738	6870	0.20935	0.030473071
739	1183	0.034973	0.029563145	739	1729	0.04921	0.028461307	739	6611	0.21341	0.03228029
740	1184	0.030799	0.026012669	740	1827	0.051854	0.028381938	740	6616	0.20949	0.031663845
741	1096	0.031087	0.028363686	741	1950	0.050994	0.026150615	741	7191	0.21493	0.029889306
742	1163	0.033148	0.028502494	742	1860	0.05211	0.028016237	742	6749	0.228	0.033782042
743	1159	0.028863	0.024903192	743	1788	0.050411	0.028194295	743	6751	0.21063	0.03119923
744	1285	0.032462	0.025262412	744	2003	0.055135	0.027526261	744	6948	0.21132	0.030414076
745	1265	0.035373	0.027962846	745	1880	0.052386	0.027865053	745	6507	0.20586	0.031636392
746	1265	0.030288	0.023942925	746	2041	0.051543	0.025253552	746	7194	0.20904	0.029057826
747	1196	0.032376	0.027070318	747	1984	0.051738	0.026077419	747	6626	0.21309	0.032159523
748	1103	0.03213	0.029129193	748	1918	0.050955	0.026566528	748	6930	0.21234	0.030640548
749	1103	0.029541	0.02678214	749	1894	0.052229	0.027575871	749	7155	0.21575	0.030153739
750	1211	0.032399	0.026754253	750	1871	0.050996	0.027255852	750	6804	0.20828	0.030610964
751	1279	0.034189	0.02673104	751	1649	0.049818	0.030211037	751	6925	0.21068	0.030422527
752	1256	0.033141	0.026386385	752	1706	0.050848	0.029805334	752	7094	0.21303	0.030029743
753	1219	0.033819	0.027743068	753	1756	0.051246	0.029183257	753	6632	0.20578	0.031028649
754	1198	0.032951	0.027504591	754	1731	0.053136	0.030696765	754	6629	0.21222	0.032013124
755	1261	0.031903	0.025299921	755	1719	0.050202	0.029204421	755	7013	0.21175	0.030193783
756	1195	0.031791	0.026603013	756	1991	0.05975	0.030009995	756	7153	0.20936	0.029268559
757	1193	0.030283	0.025384241	757	1982	0.053935	0.027212563	757	7120	0.21544	0.030258006
758	1117	0.033736	0.030202596	758	1916	0.052852	0.027584603	758	6972	0.21003	0.030124355
759	1117	0.028872	0.025847359	759	1886	0.053766	0.028508112	759	7329	0.21508	0.02934575
760	1156	0.03113	0.02692872	760	1749	0.052061	0.029766038	760	6970	0.22388	0.032119943
761	962	0.031172	0.032403222	761	1796	0.053698	0.029898608	761	7194	0.24611	0.03421087
762	958	0.029186	0.030465449	762	1821	0.052216	0.028674355	762	7574	0.22276	0.029411672
763	1104	0.031519	0.02855	763	1785	0.052972	0.029676303	763	7089	0.22031	0.031077444
764	1203	0.032433	0.026959767	764	1732	0.051375	0.029662413	764	7072	0.22217	0.031415158
765	1202	0.029859	0.024841181	765	1735	0.052866	0.030470029	765	7194	0.21332	0.029652488
766	1107	0.032849	0.029673532	766	1673	0.050963	0.030462224	766	7078	0.21049	0.029738627
767	1027	0.029905	0.029119085	767	1671	0.049867	0.029842849	767	6964	0.21242	0.030502298
768	1031	0.02868	0.027817168	768	1717	0.053432	0.031119394	768	7310	0.22479	0.030750342
769	999	0.029786	0.029816216	769	1698	0.050606	0.02980318	769	6932	0.23896	0.034471725
770	1076	0.031115	0.028916914	770	1713	0.051431	0.030024168	770	6923	0.2365	0.034161924
771	1042	0.029594	0.028401152	771	1729	0.051582	0.02983343	771	7145	0.23739	0.033225192
772	1035	0.029237	0.028248599	772	1701	0.052679	0.03096943	772	7063	0.24069	0.034077021
773	1033	0.02858	0.02766728	773	1660	0.052288	0.031498976	773	7004	0.2363	0.033737579
774	1059	0.031925	0.030145892	774	1696	0.05007	0.029522465	774	7205	0.23367	0.032431922
775	1064	0.030726	0.028878102	775	1768	0.051525	0.02914293	775	7004	0.2273	0.032452313
776	1064	0.028373	0.026665883	776	1810	0.053355	0.029477901	776	6816	0.22005	0.032283597
777	1159	0.030781	0.026558412	777	1745	0.05257	0.030125845	777	7683	0.21297	0.027719901
778	1160	0.030377	0.02618681	778	1844	0.051723	0.028049349	778	6991	0.20926	0.029932485
779	1106	0.035383	0.031991772	779	1886	0.052744	0.027966172	779	7083	0.2195	0.030989411
780	1133	0.032009	0.028251898	780	2068	0.057954	0.028023936	780	7248	0.21621	0.02983016
781	1133	0.028764	0.02538782	781	1888	0.053784	0.028487076	781	6881	0.20899	0.030372475
782	1075	0.031969	0.029738698	782	1957	0.053695	0.027437455	782	6874	0.21013	0.03056881
783	1097	0.031058	0.028311851	783	1828	0.052862	0.028917724			**MIN**	**0.020356222**
784	1097	0.029394	0.026794622	784	1982	0.054821	0.027659284			**MAX**	**0.186074627**
785	1124	0.033551	0.02985	785	1714	0.052913	0.03087077			**AVG**	**0.027673166**
786	1169	0.031656	0.027079812	786	1688	0.052235	0.030945142			**STD DEV**	**0.008641468**
787	986	0.029255	0.029670487	787	1418	0.049416	0.034849224				
788	1026	0.030394	0.029624172	788	1577	0.051868	0.032890552				
789	1092	0.030564	0.027988919	789	1858	0.055371	0.029801399				
790	1025	0.029536	0.028816	790	1793	0.052923	0.029516341				
791	1005	0.029958	0.029808856	791	1757	0.052021	0.029608025				
792	1009	0.028594	0.028338652	792	1934	0.055407	0.028648862				
793	991	0.030978	0.031259637	793	1996	0.054502	0.027305361				
794	992	0.030175	0.030418548	794	1965	0.057094	0.02905542				
795	1033	0.029551	0.02860697	795	1882	0.0549	0.029171148				
796	1028	0.031348	0.030494261	796	2023	0.056097	0.027729461				
797	1022	0.029493	0.028858415	797	1862	0.053813	0.028900806				
798	1079	0.031692	0.029371826	798	1778	0.055063	0.030968785				
799	1104	0.033237	0.03010625	799	1729	0.053403	0.030886755				
800	1108	0.029821	0.026914079	800	1657	0.052896	0.031922692				
801	1101	0.031314	0.028441144	801	1590	0.050656	0.031859308				
802	1057	0.032357	0.030611826	802	1586	0.050651	0.031936192				
803	1067	0.030409	0.028499906	803	1787	0.054776	0.030652266				
804	1050	0.032714	0.031156381	804	1937	0.058248	0.030071141				
805	1058	0.031071	0.029367864	805	1917	0.053647	0.027985029				

				Avg time per obj over all frames:			0.029803515	Standard Deviation:	0.00606977		
Helicopter Video (640 x 360)				Dashboard Video (848 x 480)				Drone Video (1920 x 1080)			
Frame No	Num Obj	Time in S	Time per obj (ms)	Frame No	Num Obj	Time in S	Time per obj (ms)	Frame No	Num Obj	Time in S	Time per obj (ms)
806	1046	0.029933	0.028616635	806	1907	0.054067	0.028351809				
807	1025	0.030017	0.029284683	807	2141	0.060854	0.028423027				
808	1131	0.032572	0.028799293	808	1775	0.056228	0.03167769				
809	1055	0.030772	0.029167867	809	1834	0.053374	0.02910229				
810	1107	0.031544	0.028495122	810	1980	0.056058	0.028311919				
811	1141	0.029904	0.026208414	811	1955	0.056204	0.028748747				
812	1133	0.033266	0.029360724	812	2058	0.057111	0.027750826				
813	1131	0.030537	0.027000088	813	2013	0.057025	0.028328217				
814	1099	0.03153	0.0286899	814	2090	0.059145	0.028299234				
815	1100	0.033134	0.030122	815	1918	0.055592	0.028984463				
816	1101	0.03013	0.027365758	816	2006	0.057781	0.028804237				
817	1103	0.032828	0.029762103	817	1868	0.054989	0.02943758				
818	1081	0.033281	0.030787327	818	1874	0.054419	0.029038687				
819	1074	0.029255	0.027239013	819	1974	0.056059	0.028398784				
820	1051	0.031785	0.030242626	820	1876	0.055343	0.02950032				
821	1059	0.033561	0.031691407	821	1735	0.053988	0.031116715				
822	1052	0.029455	0.02799943	822	2032	0.058492	0.028785433				
823	1096	0.032747	0.029879015	823	1828	0.05771	0.031569912				
824	1027	0.031103	0.030285492	824	1754	0.05287	0.030142702				
825	1022	0.03092	0.030253914	825	1798	0.052898	0.029420634				
826	1022	0.031108	0.03043865	826	1596	0.050735	0.031788596				
827	1022	0.031436	0.030759491	827	1561	0.051085	0.032725561				
828	1030	0.031542	0.03062301	828	1694	0.053791	0.031753719				
829	1051	0.031798	0.03025509	829	1600	0.05451	0.034068625				
830	1051	0.030172	0.028708278	830	1758	0.054138	0.030795222				
831	1014	0.032444	0.03199645	831	1779	0.053439	0.030038842				
832	1014	0.030475	0.030053846	832	1813	0.055571	0.030651351				
833	980	0.031001	0.031633878	833	1810	0.053945	0.029803757				
834	1002	0.031348	0.03128513	834	1849	0.053448	0.028906544				
835	1002	0.02948	0.029421058	835	1769	0.054195	0.030635896				
836	1084	0.03132	0.028893081	836	1715	0.053473	0.031179417				
837	1043	0.031006	0.029727709	837	1723	0.05043	0.029268485				
838	1043	0.029179	0.027975839	838	1656	0.050019	0.030204771				
839	986	0.031001	0.031441379	839	1671	0.048958	0.029298863				
840	998	0.031281	0.031343287	840	1935	0.052814	0.027293953				
841	990	0.031836	0.032157172	841	2159	0.057111	0.026452617				
842	976	0.030496	0.031245902	842	2153	0.057559	0.026734231				
843	1009	0.029762	0.029496036	843	2092	0.0559	0.02672065				
844	970	0.029312	0.030218557	844	1864	0.052947	0.028405204				
845	939	0.030573	0.032559531	845	2118	0.057762	0.027271719				
846	935	0.029762	0.031830909	846	2286	0.058365	0.025531584				
847	971	0.030383	0.031290834	847	1954	0.059749	0.03057784				
848	983	0.030725	0.031256053	848	1827	0.054303	0.029722551				
849	985	0.028604	0.029039188	849	1855	0.054377	0.029313962				
850	946	0.032172	0.034008245	850	1921	0.052967	0.027572827				
851	946	0.028557	0.030186681	851	1924	0.054038	0.028086331				
852	909	0.032958	0.036256986	852	2149	0.058469	0.027207585				
853	884	0.031319	0.035429072	853	2003	0.053568	0.026744034				
854	883	0.027907	0.03160487	854	2049	0.05466	0.026676574				
855	932	0.030488	0.032712124	855	1916	0.054289	0.028334708				
856	932	0.030837	0.033086803	856	1761	0.055081	0.031278251				
857	932	0.032685	0.035069313	857	1623	0.0501	0.030868823				
858	917	0.031834	0.034715812	858	1651	0.049502	0.029983283				
859	961	0.031235	0.032502706	859	1847	0.052712	0.028539145				
860	961	0.032122	0.033425598	860	1773	0.052141	0.029408347				
861	947	0.031627	0.033397466	861	1938	0.053362	0.027534623				
862	952	0.033694	0.035392857	862	2032	0.058021	0.028553691				
863	919	0.028873	0.031417954	863	1973	0.053739	0.027237405				
864	916	0.030057	0.032813428	864	1991	0.058408	0.029335912				
865	916	0.028899	0.031548799	865	2015	0.054366	0.026980645				
866	1000	0.030751	0.0307508	866	2108	0.056456	0.026781879				
867	995	0.02957	0.029718894	867	2133	0.056608	0.0265391				
868	1061	0.031343	0.029540999	868	2155	0.055288	0.02565587				
869	1026	0.030792	0.030011209	869	1918	0.053351	0.027815954				
870	1021	0.029414	0.028808913	870	1782	0.054026	0.030317733				
871	1065	0.031026	0.029132113	871	1729	0.050848	0.029408618				
872	1058	0.03159	0.029858129	872	1630	0.049236	0.030206196				
873	1058	0.030509	0.028836484	873	1951	0.052487	0.026902819				
874	1009	0.030266	0.029996036	874	1771	0.051618	0.029145963				
875	1024	0.030975	0.030249219	875	1842	0.05543	0.030092237				
876	1024	0.029409	0.028719922	876	1652	0.051548	0.031203632				
877	998	0.031303	0.031365832	877	1605	0.055286	0.034446168				
878	1011	0.030225	0.029896439	878	1882	0.054423	0.028917747				
879	972	0.030663	0.031545988	879	1863	0.05315	0.028529254				
880	997	0.030638	0.030730391	880	1751	0.054411	0.031074243				
881	1006	0.028827	0.028654871	881	1753	0.052107	0.029724701				
882	979	0.031729	0.032409602	882	1953	0.058516	0.029961956				
883	965	0.029066	0.03012	883	1985	0.05411	0.027259647				
884	961	0.027896	0.029027784	884	1822	0.052887	0.029026619				
885	926	0.029157	0.031487149	885	1690	0.053005	0.031363905				
886	925	0.029199	0.031566919	886	1815	0.051285	0.028256143				
887	889	0.030316	0.034101012	887	1894	0.051705	0.027299261				
888	914	0.030523	0.033395186	888	2078	0.053804	0.025892108				
889	913	0.026861	0.029420591	889	2006	0.053007	0.026424177				
890	937	0.029777	0.031779402	890	1932	0.053107	0.027487836				
891	936	0.031177	0.03330844	891	1914	0.058229	0.030422414				
892	936	0.028388	0.030328526	892	1866	0.056495	0.030275938				
893	920	0.031383	0.034112283	893	2019	0.059902	0.029669143				
894	1026	0.029364	0.028619981	894	1939	0.05601	0.028885921				
895	911	0.029724	0.032627442	895	1961	0.056601	0.028863233				
896	1030	0.030372	0.029486893	896	1833	0.054747	0.02986754				
897	1027	0.030979	0.030164265	897	1813	0.054791	0.030221401				
898	1019	0.029812	0.029256232	898	1784	0.056211	0.031508128				
899	976	0.029813	0.030545594	899	1861	0.056873	0.030560451				
900	975	0.028508	0.029238974	900	2152	0.059352	0.027579926				
901	993	0.030412	0.030626183	901	2196	0.05734	0.026110929				
902	966	0.029511	0.030549379	902	2267	0.055908	0.024661623				
903	964	0.028584	0.029651349	903	2312	0.056958	0.024635813				
904	954	0.031119	0.032619078	904	2196	0.056574	0.025762113				
905	954	0.029764	0.031198637	905	2197	0.058447	0.02660305				
906	865	0.028476	0.032920694	906	2199	0.05806	0.026402819				
907	917	0.029309	0.031961614	907	2190	0.064951	0.0296579				
908	917	0.028153	0.030700654	908	2104	0.057722	0.027434221				
909	909	0.030107	0.033120792	909	2347	0.061254	0.026098935				
910	867	0.031127	0.035901384	910	2182	0.057304	0.026262145				
911	873	0.027311	0.031284536	911	2161	0.057604	0.026655946				
912	999	0.031228	0.031258859	912	2168	0.059948	0.027651199				
913	918	0.030993	0.033761438	913	2154	0.057742	0.026806778				
914	920	0.030033	0.032644565	914	1973	0.05648	0.028626609				
915	880	0.030048	0.034145455	915	2068	0.059183	0.02861823				
916	872	0.030408	0.034871904	916	1996	0.056884	0.028499048				
917	847	0.028448	0.033587249	917	2010	0.059365	0.029534876				
918	924	0.030506	0.033014935	918	2150	0.059913	0.027866512				
919	924	0.02919	0.031591234	919	2159	0.058357	0.027029782				
920	901	0.032869	0.036480688	920	2085	0.055356	0.026549832				

Avg time per obj over all frames:	0.029803515	Standard Deviation:	0.00606977

Helicopter Video (640 x 360)

Frame No	Num Obj	Time in S	Time per obj (ms)
921	899	0.028554	0.03176218
922	927	0.029738	0.032079612
923	900	0.030349	0.033721556
924	915	0.029272	0.031990929
925	935	0.029522	0.031574011
926	908	0.031261	0.034428194
927	903	0.027732	0.030711074
928	897	0.030545	0.034052397
929	891	0.031326	0.035158249
930	891	0.02905	0.032603816
931	1095	0.033027	0.03016137
932	965	0.031262	0.032396269
933	939	0.030981	0.032993291
934	954	0.030868	0.032356499
935	970	0.029644	0.030561031
936	1051	0.031773	0.030231018
937	936	0.030837	0.032945406
938	935	0.029491	0.031540642
939	1095	0.032278	0.029477352
940	1105	0.030102	0.027241719
941	917	0.030165	0.032895202
942	925	0.031681	0.034249514
943	925	0.028548	0.030863027
944	935	0.030219	0.032319786
945	996	0.031794	0.031921787
946	990	0.029528	0.02982596
947	957	0.033049	0.034533438
948	919	0.030634	0.033334276
949	959	0.030716	0.032029197
950	893	0.030122	0.033731467
951	953	0.031059	0.032590661
952	942	0.030011	0.031859023
953	968	0.032319	0.033387603
954	964	0.029993	0.031113174
955	943	0.033227	0.035235419
956	1102	0.033499	0.030398185
957	1094	0.029819	0.027256673
958	1069	0.031684	0.029638821
959	1069	0.030147	0.028201216
960	1124	0.032379	0.028806762
961	1116	0.03398	0.03044767
962	1116	0.029708	0.026620251
963	1106	0.031891	0.02883481
964	1128	0.032884	0.029152216
965	1128	0.031127	0.027595124
966	1106	0.032808	0.029663562
967	1104	0.031598	0.028621196
968	1081	0.031443	0.029087234
969	1065	0.031278	0.029368826
970	1072	0.034093	0.031803545
971	1101	0.032721	0.029719619
972	1071	0.030977	0.028923249
973	1071	0.02801	0.026153315
974	1106	0.033048	0.029880741
975	1106	0.029961	0.027089873
976	1067	0.031469	0.029492596
977	1093	0.031241	0.028582525
978	1095	0.029615	0.027046027
979	1035	0.032162	0.031073913
980	980	0.031072	0.031705714
981	981	0.028237	0.028783894
982	984	0.031187	0.031693598
983	1041	0.033005	0.031705091
984	1057	0.028146	0.026627909
985	1013	0.030509	0.030117868
986	1027	0.031372	0.03054742
987	992	0.0307	0.030947379
988	999	0.030773	0.030803303
989	999	0.030183	0.030213313
990	905	0.0301	0.033259337
991	998	0.030871	0.030932766
992	1000	0.029651	0.0296513
993	986	0.030586	0.03102069
994	986	0.029503	0.029921704
995	956	0.031262	0.032701255
996	1090	0.033109	0.030374771
997	1091	0.030464	0.027923373
998	1167	0.031334	0.026849614
999	1087	0.029455	0.027097516
1000	1089	0.029945	0.027497245
		MIN	**0.020739069**
		MAX	**0.049430935**
		AVG	**0.031405005**
		STD DEV	**0.005491394**

Dashboard Video (848 x 480)

Frame No	Num Obj	Time in S	Time per obj (ms)
921	1962	0.061413	0.031301274
922	1930	0.053936	0.027946166
923	1964	0.056404	0.028718992
924	2120	0.057841	0.027283255
925	2308	0.057126	0.024751256
926	2215	0.05642	0.025471919
927	2101	0.060855	0.028964683
928	2211	0.058715	0.026555767
929	2053	0.057906	0.028205407
930	2058	0.059763	0.029039116
931	1995	0.055671	0.027905363
932	2136	0.057813	0.027066152
933	2039	0.058419	0.02865076
934	2065	0.055448	0.026851138
935	2051	0.057439	0.028005412
936	2027	0.059699	0.02945185
937	2036	0.056717	0.027857269
		MIN	**0.024304688**
		MAX	**0.042673413**
		AVG	**0.02987229**
		STD DEV	**0.002423722**

Drone Video (1920 x 1080)

Frame No	Num Obj	Time in S	Time per obj (ms)

Appendix C

WISE C++ Code

This section contains a summary of the C++ code used to implement WISE. The base classes for the algorithms are included, intermediary and utility code is not included (the length of code would be to great to print)

```
/*********************************************************************************************

    This file has been produced at the Intelligent Systems Research Laboratory at
    InfoLab21, Lancaster University under the supervision of Professor Plamen Angelov.
    Reproduction of the code is permitted for academic and research purposes only without
    the express permission of the author. Use for any  other purpose is not permitted without
    prior authorisation or permission. All code taken from this document must have this
    header at the top of the new source file.

    Filename:       EdgeFlow.cpp
    Author:         Gruffydd Morris
    Date Created:   13/07/2014
    Description:    This file contains code implementation relating to the new edge flow
                    technique. This
    ___________
    Amendments
    ****************************************************************************
    *   Date    |   Description of change                       |   Initials    *
    *           |                                               |               *
    ****************************************************************************


*********************************************************************************************/

#include <math.h>
#include <EdgeFlow.h>
#include <EdgeFlow_GUI.h>
#include <Utils.h>
//#include <CEDAS.h>
#include <opencv2\opencv.hpp>

// Function to calulate a colour matrix value for a given input.
// This is used in gradient definition of an image (and possibly in
// other cases)

void MouseCallBack(int event, int x, int y, int flags, void* userdata)
{
    if (event == cv::EVENT_LBUTTONDOWN)
    {
        static_cast<EdgeSobel*>(userdata)->xyFinder(x, y);
    }
}

bool uniqueClusters(cv::Point2f first, cv::Point2f second)
{
    return (first.x == second.x && first.y == second.y);
}

ColourBGR& colourMap(double max, double min, double input)
{
    static ColourMap cM;

    double stepNum = ((input - min) / (std::ceil(max - min))) * NUM_COLOUR_STEPS;
    unsigned int index = static_cast<unsigned int>(std::floor(stepNum));

    //if (input > -0.05 && input < 0.05)
    //{
        //return ColourBGR(0, 0, 0);
    //}
    //else
    {
        return cM.getColour(index);
    }

}

EdgeSobel::EdgeSobel(cv::Mat& frame, EdgeFlow& eG)
{
    showObj = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC3);                  // Coloured matrix for   ↙
```

```
    outputting object detection rectangles
    showObjX = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC3);                 // Coloured matrix for          ↙
    outputting object detection rectangles
    showObjY = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC3);                 // Coloured matrix for          ↙
    outputting object detection rectangles
    showObjClust = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC3);                 // Coloured matrix for      ↙
    outputting object detection rectangles

    gray = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC1);                 // Coloured matrix for outputting   ↙
    object detection rectangles
    prevGray = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC1);                 // Coloured matrix for          ↙
    outputting object detection rectangles

    data = cv::Mat::zeros(frame.rows, frame.cols, CV_64FC1);                    // Coloured matrix for          ↙
    outputting object detection rectangles
    data2 = cv::Mat::zeros(frame.rows, frame.cols, CV_64FC1);                   // Coloured matrix for          ↙
    outputting object detection rectangles

    dNormX = cv::Mat::zeros(frame.rows, frame.cols, CV_64FC1);                  // Coloured matrix for          ↙
    outputting object detection rectangles
    dNormY = cv::Mat::zeros(frame.rows, frame.cols, CV_64FC1);                  // Coloured matrix for          ↙
    outputting object detection rectangles
    dNormC = cv::Mat::zeros(frame.rows, frame.cols, CV_64FC1);                  // Coloured matrix for          ↙
    outputting object detection rectangles

    clusterFlag[0] = cv::Mat::zeros(frame.rows, frame.cols, CV_16UC1);               // Cluster membership      ↙
    tracking flag for first sobel image
    clusterFlag[1] = cv::Mat::zeros(frame.rows, frame.cols, CV_16UC1);               // Cluster membership      ↙
    tracking flag for second sobel image

    roiClick = cv::Mat::zeros(frame.rows, frame.cols, CV_16UC1);                   // Frame storing the region ↙
    value

    vidOutput = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC1);                   // Video output matrix used ↙
    when the output is normally greyscale or floating point

    blobs = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC3);
    theClusters = cv::Mat::zeros(frame.rows, frame.cols, CV_64FC1);
    theClustersOut = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC3);

    ofWork = CreateEvent(NULL, true, false, NULL);



    numPoints = 0;
    maxNumClust = 0;
    totalClusters = 0;

    // Initialise the max / min gradient variables used to assign the correct range to colour assignment
    locX = 0;
    locY = 0;

    clickPoint.dir = -1;
    clickPoint.mag = 0;
    clickPoint.region = 0;



    addRemovePt = false;
    segmentOn = false;
    clickedFlag = false;
    ofFlag = false;

    eGUI = &eG;

    // **************************** CEDAS INIT ***************************************//
    this->fnCEDAS = 0;
    initRad = 0.03;
    minClustSize = 2;
```

```
    decay = 10;

#ifdef CEDASCLUSTER
    cedas = new CEDAS(0.03, 2, 10);
#endif

   cv::Size S = cv::Size((int) X_RES, (int) Y_RES);
   demoOut.open(cv::String("D:\\SampleVideos\\Out\\DemoOut.wmv"), CV_FOURCC('W','M','V','1'), /*cap.get
    (CV_CAP_PROP_FPS)*/25, S, true);

#ifdef DATASET_OUTPUT
    char filename[30];

    sprintf_s(filename, 30, "D:\\VideoDataset\\FramesDataset");
    dataset.open(filename, std::ios::trunc);
#endif
}

EdgeSobel::~EdgeSobel()
{
#ifdef DATASET_OUTPUT
     dataset.close();
#endif

    this->vecOut.close();
    demoOut.release();
}

cv::Mat& EdgeSobel::sobelRun(cv::Mat& rde, cv::Mat& rdeY, cv::Mat& frame, cv::Mat& oldFrame, cv::VideoWriter&
    vw)
{
    fnCEDAS++;
    static OFCluster ocl(this, ofWork);
    static int frameNumber = 0;
    double min;
    double max;
    double min2;
    double max2;

    frameNumber++;

    if (frameNumber == 1)
        pVW = &vw;

    cv::Sobel(rde, data, CV_64F, 1, 0, 7, 1.0, 0.0, cv::BORDER_REPLICATE);
    cv::Sobel(rdeY, data2, CV_64F, 0, 1, 7, 1.0, 0.0, cv::BORDER_REPLICATE);


    cv::minMaxLoc(data, &min, &max);
    cv::minMaxLoc(data2, &min2, &max2);

    min = min < min2 ? min : min2;
    max = max > max2 ? max : max2;

    cv::normalize(data, dNormX, min, max, cv::NORM_L2);
    cv::normalize(data2, dNormY, min, max, cv::NORM_L2);

    dNormC = dNormX + dNormY;


    this->colourGradients(dNormC, showObj);
    this->colourGradients(dNormX, showObjX);
    this->colourGradients(dNormY, showObjY);

    //vw << showObj;

#ifdef VISUALS
    cv::imshow("CombinedColour", showObj);
#endif
```

```
#ifdef FULLVISUALS
    cv::imshow("Combined", dNormC);

    cv::imshow("CombinedColourX", showObjX);
    cv::imshow("CombinedColourY", showObjY);
#endif

#ifdef CEDASCLUSTER
    if (frameNumber % 5 == 0)
    {
        oldFrame.copyTo(next);
        for (unsigned int y = 0; y < dNormC.rows; y++)
        {
            for (unsigned int x = 0; x < dNormC.cols; x++)
            {
                if (dNormC.at<double>(y, x) >= eGUI->getCValue() || dNormC.at<double>(y, x) <= 0 - eGUI->
    getCValue())
                    ofPoints.push_back(cv::Point2f(x, y));
            }
        }

        std::cout << "Optical flow points " << ofPoints.size() << std::endl;
        if (ofPoints.size() > 0)
            this->edgeOpticalFlow(frame, next, ofPoints);

#ifdef FULLVISUALS
        cv::imshow("Optical flow", next);
#endif

#ifdef OFCLUSTERVISUALS
        cv::imshow("Optical flow", next);
#endif
    }
#endif CEDASCLUSTER



#ifdef FULLVISUALS
    drawImages(SOBEL);
#endif

#ifdef EDGECLUSTER
    this->edgeCluster(frame, oldFrame, vw, rde, rdeY);
#endif

    ofPoints.clear();
    return cv::Mat();

}



void EdgeSobel::edgeCluster(cv::Mat& frame, cv::Mat& oldFrame, cv::VideoWriter& vw, cv::Mat& rde, cv::Mat& rdeY)
{
    static int fNum = 0;
    fNum++;



    cNumber = 0;
    clusterN = 0;

    oldFrame.copyTo(next);


    clusterFlag[0] = cv::Mat::zeros(data.rows, data.cols, CV_16UC1);               // Cluster membership
    tracking flag for first sobel image
    clusterFlag[1] = cv::Mat::zeros(data.rows, data.cols, CV_16UC1);               // Cluster membership
```

```
    tracking flag for second sobel image

    for (int y = 0; y < data.rows; y++)
    {
        for (int x = 0; x < data.cols; x++)
        {

#ifdef TWOCLUSTERIMAGES
            density = data.ptr<double>(y)[x];
            density2 = data2.ptr<double>(y)[x];
#else
            density = dNormC.ptr<double>(y)[x];
#endif

            if (density <= this->eGUI->getCValue()/*MAX_CLUSTERS*/ && density >= (0 - this->eGUI->getCValue())/* ↙
    MIN_CLUSTERS*/)       // Old density filter
            {
                // do nothing
            }
            else
            {
                clusterN = clusterFlag[0].ptr<unsigned short>(y)[x];
                // Check if the cluster flag is set, if so - just add to existing cluster and check local region
                if (clusterN != 0)
                {
                    // If within proximity of existing points, add the point to the regions list, and update
                    // the regions extremities. Extremities only used to find cluster centre.
                    c[clusterN - 1].addPoint(cv::Point2i(x, y));

                    c[clusterN - 1].expandSize(x, y, data);

                    // Update the mean density of the region
                    c[clusterN - 1].setMeanDensity(((c[clusterN - 1].getPoints().size() - 1) / (c[clusterN - 1]. ↙
    getPoints().size() * c[clusterN - 1].getMeanDensity()))
                                                            + ((1 / c[clusterN - 1].getPoints().size()) * density));

                    if (y < (data.rows - 1) && x < (data.cols - 1))
                    {
                        checkProximityGradients(x, y);
                    }
                }
                else
                {
                    newCluster = new FeatureExtraction(cv::Point2i(x, y));
                    newCluster->setSize(x, y);

                    meanD = density;
                    newCluster->setMeanDensity(meanD);

                    c.push_back(*newCluster);

                    cNumber++;
                    clusterFlag[0].ptr<unsigned short>(y)[x] = clusterN = cNumber;

                    if (y < (data.rows - 1) && x < (data.cols - 1))
                    {
                       checkProximityGradients(x, y);
                    }

                }      // else flag check
            }           // If do nothing

#ifdef TWOCLUSTERIMAGES
            if (density2 <= this->eGUI->getCValue()/*MAX_CLUSTERS*/ && density2 >= (0 - this->eGUI->getCValue()) ↙
    /*MIN_CLUSTERS*/)
                //if (density2 >= this->eGUI->getCValue()/*MAX_CLUSTERS*/ || density2 <= (0 - this->eGUI->        ↙
    getCValue())/*MIN_CLUSTERS*/)
            {
                // do nothing
```

```
            }
            else
            {
                clusterN = clusterFlag[1].ptr<unsigned short>(y)[x];
                // Check if the cluster flag is set, if so - just add to existing cluster and check local region
                if (clusterN != 0)
                {
                     c[clusterN - 1].addPoint(cv::Point2i(x, y));

                    c[clusterN - 1].expandSize(x, y, data2);

                    // Update the mean density of the region
                    c[clusterN - 1].setMeanDensity(((c[clusterN - 1].getPoints().size() - 1) / (c[clusterN - 1].↙
    getPoints().size() * c[clusterN - 1].getMeanDensity()))
                                                                    + ((1 / c[clusterN - 1].getPoints().size()) * density2))↙
    ;

                    if (y < (data.rows - 1) && x < (data.cols - 1))
                    {
                        checkProximityGradients(x,y);
                    }
                }
                else
                {
                    // Add the x y coords to the points list

                    newCluster = new FeatureExtraction(cv::Point2i(x, y));
                    newCluster->setSize(x, y);

                    meanD = density2;
                    newCluster->setMeanDensity(meanD);

                    c.push_back(*newCluster);

                    cNumber++;
                    clusterFlag[1].ptr<unsigned short>(y)[x] = clusterN = cNumber;

                    // Check in the other sobel framee as well
                    if (y < (data.rows - 1) && x < (data.cols - 1))
                    {
                       checkProximityGradients(x, y);
                    }
                }      // else flag check
            }           // If do nothing
#endif
        }               // For x
    }                   // For y

    frame.copyTo(showObjClust);

#ifndef DRAWBLOBS
    frame.copyTo(blobs);
    theClusters = cv::Mat::zeros(frame.rows, frame.cols, CV_64FC1);
    theClustersOut = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC3);
#endif

    cv::Point2f centrePoint;

    segmentOn = false;

    if (this->eGUI->drawsegmentation())
    {
        segmentOn = true;
        mask = cv::Mat::zeros(frame.size(), CV_8UC1);
        frameROI = cv::Mat::zeros(frame.size(), CV_8UC3);
    }

    //for (unsigned int reg = 0; reg < regions.size(); reg++)
    for (unsigned int reg = 0; reg < c.size(); reg++)
```

```
    {
        //if (regions[reg][0].x != 9999)
        if (c[reg].getPoints()[0].x != 9999)
        {


            //cv::Rect roi = cv::boundingRect(regions[reg]);
            cv::Rect roi = cv::boundingRect(c[reg].getPoints());

#ifndef DRAWBLOBS
            for (int i = 0; i < c[reg].getPoints().size(); i++)
            {
                theClusters.at<double>(c[reg].getPoints()[i].y, c[reg].getPoints()[i].x) = (double)(reg + 1);
                theClustersOut.at<uchar>(c[reg].getPoints()[i].y, c[reg].getPoints()[i].x) = (reg + 1);
            }
#endif
            if((roi.size().height > this->eGUI->getObjSize() && roi.size().width > this->eGUI->getObjSize()))
            {
                //centrePoint.x = static_cast<float>(cLimits[reg].minX + ((cLimits[reg].maxX - cLimits[reg].
    minX) / 2));
                //centrePoint.y = static_cast<float>(cLimits[reg].minY + ((cLimits[reg].maxY - cLimits[reg].
    minY) / 2));

                centrePoint.x = c[reg].getPoints()[static_cast<int>(c[reg].getPoints().size() / 2)].x; //
    static_cast<float>(c[reg].getSize().minX + ((c[reg].getSize().maxX - c[reg].getSize().minX) / 2));
                centrePoint.y = c[reg].getPoints()[static_cast<int>(c[reg].getPoints().size() / 2)].y;//
    static_cast<float>(c[reg].getSize().minY + ((c[reg].getSize().maxY - c[reg].getSize().minY) / 2));

                ofPoints.push_back(centrePoint);

                c[reg].run(frame, dNormC, cv::Mat());

                cv::rectangle(showObjClust, roi, cv::Scalar(0,0,255), 2, 8, 0);

                cv::Mat roiTemp(this->roiClick, roi);
                roiTemp = cv::Scalar(reg + 1);      // Reg + 1 because the roiClick matrix is initialised to
    zero. As regions is a
                                                    // zero based array, clicking the first region would yield
    the entire image therefore
                                                    // an offset of 1 is required.
                //if (this->eGUI->drawsegmentation() && segmentOn == true)
                    //cv::drawContours(mask, regions, reg, cv::Scalar(255), CV_FILLED);
            }
        }
    }



#ifndef DRAWBLOBS
    this->colourGradients(theClusters, theClustersOut);
    theClustersOut.copyTo(blobs, theClustersOut);
    cv::imshow("Blobs", blobs);
     //vw << blobs;
    cv::imshow("TheClusters", theClustersOut);
#endif

    this->edgeOpticalFlow(next, frame, ofPoints);


#ifdef DATASET_OUTPUT
     /*for (int i = 0; i < c.size(); i++)
     {
         c[i].writeFrameDataset(fNum, dataset);
     }*/
    dataset << fNum << "," << c.size() << std::endl;

#endif

    if (this->eGUI->drawsegmentation() && segmentOn == true)
```

```
        oldFrame.copyTo(frameROI, mask);

    if (this->eGUI->drawsegmentation() && segmentOn == true)
        cv::imshow("Contoured", frameROI);

#ifdef OFCLUSTERVISUALS
    //vw << showObjClust;
    cv::imshow("Clusters", showObjClust);
    //vw << showObjClust;
#else
    cv::imshow("Clusters", showObjClust);
#endif


    delete newCluster;
    theClusters.release();
    theClustersOut.release();
    c.clear();
    vOF.clear();
    ofPoints.clear();
}                       // Function end

void EdgeSobel::edgeOpticalFlow(cv::Mat& prev, cv::Mat& next, std::vector<cv::Point2f>& points)
{
    std::vector<cv::Point2f> outPoints;
    cv::Mat image;

    double mag = 0, dir = 0, xdist = 0, ydist = 0;

    cvtColor(next, gray, cv::COLOR_BGR2GRAY);
    cvtColor(prev, prevGray, cv::COLOR_BGR2GRAY);

    addRemovePt = false;

    if( !points.empty() )
    {
        std::vector<uchar> status;
        std::vector<float> err;
        if(prevGray.empty())
            gray.copyTo(prevGray);
        calcOpticalFlowPyrLK(prevGray, gray, points, outPoints, status, err);

        if (points.size() != outPoints.size())
            std::cout << "Points : " << points.size() << " outPoints : " << outPoints.size() << std::endl;

        size_t i, k;
        for( i = k = 0; i < outPoints.size(); i++ )
        {

            if( addRemovePt )
            {
                if( cv::norm(point - outPoints[i]) <= 5 )
                {
                    addRemovePt = false;
                    continue;
                }
            }

            if( !status[i] )
                continue;

            outPoints[k++] = outPoints[i];


            // Optical flow vector caluclation //

            mag = cv::sqrt(cv::pow(points[i].x - outPoints[i].x, 2) + cv::pow(points[i].y - outPoints[i].y, 2));
          // Calculate the magnitude of the optical flow
            xdist = outPoints[i].x - points[i].x;
```

```
            ydist = outPoints[i].y - points[i].y;

            dir = std::atan2(ydist, xdist);
            dir += 1.7;
            if (dir > PI)
                dir -= 2 * PI;

            this->c[i].twoframeOF(mag, dir, (unsigned int)outPoints[i].x, (unsigned int)outPoints[i].y);

           // Draw all optical flow regions
#ifndef CEDASCLUSTER
            line(showObjClust, points[i], outPoints[i], cv::Scalar(0,255,0), 1, 8);
#else
            line(next, points[i], outPoints[i], cv::Scalar(0,255,0), 1, 8);
#endif

#ifdef CEDASCLUSTER
          /*  if (mag > 20.0)
            {
            }
            else if (mag > 1)*/
            {
                dir = (dir - (-PI)) / (PI - (-PI));
                mag = (mag / 20.0);

                vOF.push_back(cv::Point2f((float)dir, (float)mag));

            }
           /* else
            {

            }*/
#endif

        }
        outPoints.resize(k);
    }

#ifdef CEDASCLUSTER
    cedas->newSamples(vOF, fnCEDAS);

    cv::Mat clusters = cv::Mat::zeros(640, 640, CV_8UC3);

    double numClusters = cedas->getClusters().size();//outArray.Get("Centre", 1, 1).GetDimensions()(1,1);
    double xLoc;
    double yLoc;
    double radius;

    for (unsigned int i = 0; i < static_cast<unsigned int>(numClusters); i++)
    {
        xLoc =  cedas->getClusters()[i].centre.x;//outArray.Get("Centre", 1, 1)(i, 1);
        xLoc *= 640;
        yLoc = cedas->getClusters()[i].centre.y;//outArray.Get("Centre", 1,1)(i, 2);
        yLoc *= 640;

        xLoc = std::floor(xLoc);
        yLoc = std::floor(yLoc);
        radius = cedas->getClusters()[i].radius;//outArray.Get("Radius", 1, 1)(i, 1);
        radius *= 100;

        cv::circle(clusters, cv::Point(xLoc, yLoc), radius, cv::Scalar(0,255,0));
    }

    imshow("Cluster_CEDAS", clusters);
//#ifdef VIDEO_OUTPUT
//    *pVW << next;
//#endif
    cv::waitKey(1);
```

```
#endif

    vOF.clear();
    magV.clear();
    dirV.clear();
    outPoints.clear();

#ifdef CEDASCLUSTER
    cedas->cleanUp();
#endif
}

void EdgeSobel::emptyFrames()
{

}


// This function encapsulates the drawing of images for the
// edge flow technique. It simplifies the process of turning on or off
// the images to show.
void EdgeSobel::drawImages(unsigned int imgs)
{
    switch (imgs)
    {

        case ALL:
            cv::imshow("Obj X Sobel", data);
            cv::imshow("Obj Y Sobel", data2);
            break;

        case SOBEL:
            cv::imshow("Obj X Sobel", data);
            cv::imshow("Obj Y Sobel", data2);
#ifdef COMBINED_SOBEL // Only run command if we've combined the two Sobel images
            cv::imshow("Obj Grad Sobel", grad);
#endif
            break;

        case COLOURS:

            break;

        case COMBINED:

            break;

        case NO_OUTPUTS:
            break;

        default:
            cv::imshow("Obj X Sobel", data);
            cv::imshow("Obj Y Sobel", data2);

            break;
    }

    // Mandatory wait to enable drawing to the screen
    cv::waitKey(1);
}

void EdgeSobel::colourGradients(cv::Mat& inData, cv::Mat& outData)
{
    // Find the max and min values from the input matrix
    cv::minMaxLoc(inData, &minG, &maxG);
    //// Loop through the x and y of each sobel output to assign colour gradient
    for (int y = 0; y < inData.rows; y++)
    {
        for (int x = 0; x < inData.cols; x++)
```

```
        {
            // Assign colours to gradients from the Sobel X filter based on it's range of values. This also 
    generates a unique colour map.
            /*if (inData.at<double>(y, x) == 0.0)
            {
                outData.data[(y * inData.cols * 3) + x * 3] = 0;
                outData.data[(y * inData.cols * 3) + (x * 3) + 1] = 0;
                outData.data[(y * inData.cols * 3) + (x * 3) + 2]= 0;
            }*/
            //else
            {
                myColours = colourMap(maxG, minG, inData.at<double>(y, x));

            outData.data[(y * inData.cols * 3) + x * 3] = static_cast<uchar>(myColours.b);
            outData.data[(y * inData.cols * 3) + (x * 3) + 1] = static_cast<uchar>(myColours.g);
            outData.data[(y * inData.cols * 3) + (x * 3) + 2]= static_cast<uchar>(myColours.r);
            }
        }
    }
}

void EdgeSobel::checkProximityGradients(unsigned int x, unsigned int y)
{
     // Check proximity of other points, set a flag if within range
    if (data.ptr<double>(y)[x + 1] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data.ptr 
    <double>(y)[x + 1]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
    {
        if (clusterFlag[0].ptr<unsigned short>(y)[x + 1] == 0)
        {
            clusterFlag[0].ptr<unsigned short>(y)[x + 1] = clusterN;
        }
        else if (clusterFlag[0].ptr<unsigned short>(y)[x + 1] > clusterN)
        {
            clustMerge.x = clusterN;
            clustMerge.y = clusterFlag[0].ptr<unsigned short>(y)[x + 1];
            clusterFlag[0].ptr<unsigned short>(y)[x + 1] = clusterN;
        }
        else if (clusterFlag[0].ptr<unsigned short>(y)[x + 1] < clusterN)
        {
            clustMerge.x = clusterFlag[0].ptr<unsigned short>(y)[x + 1];
            clustMerge.y = clusterN;
        }
        else
        {
            // Do nothing - already assigned to this cluster
        }
    }

    if (data.ptr<double>(y + 1)[x + 1] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data.ptr 
    <double>(y + 1)[x + 1]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
    {
        if (clusterFlag[0].ptr<unsigned short>(y + 1)[x + 1] == 0)
        {
            clusterFlag[0].ptr<unsigned short>(y + 1)[x + 1] = clusterN;
        }
        else if (clusterFlag[0].ptr<unsigned short>(y + 1)[x + 1] > clusterN)
        {
            clustMerge.x = clusterN;
            clustMerge.y = clusterFlag[0].ptr<unsigned short>(y + 1)[x + 1];
            clusterFlag[0].ptr<unsigned short>(y + 1)[x + 1] = clusterN;
        }
        else if (clusterFlag[0].ptr<unsigned short>(y + 1)[x + 1] < clusterN)
        {
            clustMerge.x = clusterFlag[0].ptr<unsigned short>(y + 1)[x + 1];
            clustMerge.y = clusterN;
        }
        else
        {
            // Do nothing - already assigned to this cluster
```

```
        }
    }

    if (data.ptr<double>(y + 1)[x] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data.ptr
    <double>(y + 1)[x]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
    {
        if (clusterFlag[0].ptr<unsigned short>(y + 1)[x] == 0)
        {
            clusterFlag[0].ptr<unsigned short>(y + 1)[x] = clusterN;
        }
        else if (clusterFlag[0].ptr<unsigned short>(y + 1)[x] > clusterN)
        {
            clustMerge.x = clusterN;
            clustMerge.y = clusterFlag[0].ptr<unsigned short>(y + 1)[x];
            clusterFlag[0].ptr<unsigned short>(y + 1)[x] = clusterN;
        }
        else if (clusterFlag[0].ptr<unsigned short>(y + 1)[x] < clusterN)
        {
            clustMerge.x = clusterFlag[0].ptr<unsigned short>(y + 1)[x];
            clustMerge.y = clusterN;
        }
        else
        {
            // Do nothing - already assigned to this cluster
        }
    }

    if (x > 0)
    {
        if (data.ptr<double>(y + 1)[x - 1] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data.
    ptr<double>(y + 1)[x - 1]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
        {
            if (clusterFlag[0].ptr<unsigned short>(y + 1)[x - 1] == 0)
            {
                clusterFlag[0].ptr<unsigned short>(y + 1)[x - 1] = clusterN;
            }
            else if (clusterFlag[0].ptr<unsigned short>(y + 1)[x - 1] > clusterN)
            {
                clustMerge.x = clusterN;
                clustMerge.y = clusterFlag[0].ptr<unsigned short>(y + 1)[x - 1];
                clusterFlag[0].ptr<unsigned short>(y + 1)[x - 1] = clusterN;
            }
            else if (clusterFlag[0].ptr<unsigned short>(y + 1)[x - 1] < clusterN)
            {
                clustMerge.x = clusterFlag[0].ptr<unsigned short>(y + 1)[x - 1];
                clustMerge.y = clusterN;
            }
            else
            {
                // Do nothing - already assigned to this cluster
            }
        }

        if (data.ptr<double>(y)[x - 1] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data.ptr
    <double>(y)[x - 1]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
        {
            if (clusterFlag[0].ptr<unsigned short>(y)[x - 1] == 0)
            {
                clusterFlag[0].ptr<unsigned short>(y)[x - 1] = clusterN;
            }
            else if (clusterFlag[0].ptr<unsigned short>(y)[x - 1] > clusterN)
            {
                clustMerge.x = clusterN;
                clustMerge.y = clusterFlag[0].ptr<unsigned short>(y)[x - 1];
                clusterFlag[0].ptr<unsigned short>(y)[x - 1] = clusterN;
            }
            else if (clusterFlag[0].ptr<unsigned short>(y)[x - 1] < clusterN)
            {
                clustMerge.x = clusterFlag[0].ptr<unsigned short>(y)[x - 1];
```

```
                clustMerge.y = clusterN;
            }
            else
            {
                // Do nothing - already assigned to this cluster
            }
        }
    }

    if (y > 0)
    {
        if (data.ptr<double>(y - 1)[x + 1] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data.
    ptr<double>(y - 1)[x + 1]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
        {
            if (clusterFlag[0].ptr<unsigned short>(y - 1)[x + 1] == 0)
            {
                clusterFlag[0].ptr<unsigned short>(y - 1)[x + 1] = clusterN;
            }
            else if (clusterFlag[0].ptr<unsigned short>(y - 1)[x + 1] > clusterN)
            {
                clustMerge.x = clusterN;
                clustMerge.y = clusterFlag[0].ptr<unsigned short>(y - 1)[x + 1];
                clusterFlag[0].ptr<unsigned short>(y - 1)[x + 1] = clusterN;
            }
            else if (clusterFlag[0].ptr<unsigned short>(y - 1)[x + 1] < clusterN)
            {
                clustMerge.x = clusterFlag[0].ptr<unsigned short>(y - 1)[x + 1];
                clustMerge.y = clusterN;
            }
            else
            {
                // Do nothing - already assigned to this cluster
            }
        }

        if (data.ptr<double>(y - 1)[x] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data.ptr
    <double>(y - 1)[x]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
        {
            if (clusterFlag[0].ptr<unsigned short>(y - 1)[x] == 0)
            {
                clusterFlag[0].ptr<unsigned short>(y - 1)[x] = clusterN;
            }
            else if (clusterFlag[0].ptr<unsigned short>(y - 1)[x] > clusterN)
            {
                clustMerge.x = clusterN;
                clustMerge.y = clusterFlag[0].ptr<unsigned short>(y - 1)[x];
                clusterFlag[0].ptr<unsigned short>(y - 1)[x] = clusterN;
            }
            else if (clusterFlag[0].ptr<unsigned short>(y - 1)[x] < clusterN)
            {
                clustMerge.x = clusterFlag[0].ptr<unsigned short>(y - 1)[x];
                clustMerge.y = clusterN;
            }
            else
            {
                // Do nothing - already assigned to this cluster
            }
        }
    }


#ifdef TWOCLUSTERIMAGES
    if (data2.ptr<double>(y)[x + 1] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data2.ptr
    <double>(y)[x + 1]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
    {
        if (clusterFlag[1].ptr<unsigned short>(y)[x + 1] == 0)
        {
            clusterFlag[1].ptr<unsigned short>(y)[x + 1] = clusterN;
        }
```

```
        else if (clusterFlag[1].ptr<unsigned short>(y)[x + 1] > clusterN)
        {
            clustMerge.x = clusterN;
            clustMerge.y = clusterFlag[1].ptr<unsigned short>(y)[x + 1];
            clusterFlag[1].ptr<unsigned short>(y)[x + 1] = clusterN;
        }
        else if (clusterFlag[1].ptr<unsigned short>(y)[x + 1] < clusterN)
        {
            clustMerge.x = clusterFlag[1].ptr<unsigned short>(y)[x + 1];
            clustMerge.y = clusterN;
        }
        else
        {
            // Do nothing - already assigned to this cluster
        }
    }

    if (data2.ptr<double>(y + 1)[x + 1] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data2.
    ptr<double>(y + 1)[x + 1]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
    {
        if (clusterFlag[1].ptr<unsigned short>(y + 1)[x + 1] == 0)
        {
            clusterFlag[1].ptr<unsigned short>(y + 1)[x + 1] = clusterN;
        }
        else if (clusterFlag[1].ptr<unsigned short>(y + 1)[x + 1] > clusterN)
        {
            clustMerge.x = clusterN;
            clustMerge.y = clusterFlag[1].ptr<unsigned short>(y + 1)[x + 1];
            clusterFlag[1].ptr<unsigned short>(y + 1)[x + 1] = clusterN;
        }
        else if (clusterFlag[1].ptr<unsigned short>(y + 1)[x + 1] < clusterN)
        {
            clustMerge.x = clusterFlag[1].ptr<unsigned short>(y + 1)[x + 1];
            clustMerge.y = clusterN;
        }
        else
        {
            // Do nothing - already assigned to this cluster
        }
    }

    if (data2.ptr<double>(y + 1)[x] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() && data2.ptr
    <double>(y + 1)[x]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
    {
        if (clusterFlag[1].ptr<unsigned short>(y + 1)[x] == 0)
        {
            clusterFlag[1].ptr<unsigned short>(y + 1)[x] = clusterN;
        }
        else if (clusterFlag[1].ptr<unsigned short>(y + 1)[x] > clusterN)
        {
            clustMerge.x = clusterN;
            clustMerge.y = clusterFlag[1].ptr<unsigned short>(y + 1)[x];
            clusterFlag[1].ptr<unsigned short>(y + 1)[x] = clusterN;
        }
        else if (clusterFlag[1].ptr<unsigned short>(y + 1)[x] < clusterN)
        {
            clustMerge.x = clusterFlag[1].ptr<unsigned short>(y + 1)[x];
            clustMerge.y = clusterN;
        }
        else
        {
            // Do nothing - already assigned to this cluster
        }
    }

    if (x > 0)
    {
        if (data2.ptr<double>(y + 1)[x - 1] < c[clusterN - 1].getMeanDensity() + this->eGUI->getGValue() &&
    data2.ptr<double>(y + 1)[x - 1]  > c[clusterN - 1].getMeanDensity() + (0 - this->eGUI->getGValue()))
```

```
        {
            if (clusterFlag[1].ptr<unsigned short>(y + 1)[x - 1] == 0)
            {
                clusterFlag[1].ptr<unsigned short>(y + 1)[x - 1] = clusterN;
            }
            else if (clusterFlag[1].ptr<unsigned short>(y + 1)[x - 1] > clusterN)
            {
                clustMerge.x = clusterN;
                clustMerge.y = clusterFlag[1].ptr<unsigned short>(y + 1)[x - 1];
                clusterFlag[1].ptr<unsigned short>(y + 1)[x - 1] = clusterN;
            }
            else if (clusterFlag[1].ptr<unsigned short>(y + 1)[x - 1] < clusterN)
            {
                clustMerge.x = clusterFlag[1].ptr<unsigned short>(y + 1)[x - 1];
                clustMerge.y = clusterN;
            }
            else
            {
                // Do nothing - already assigned to this cluster
            }
        }
    }
#endif
}

OFCluster::OFCluster(EdgeSobel* eS, HANDLE sObj) : eSob(eS), Threaded(sObj)
{

}

void OFCluster::setup(cv::Mat& oF, cv::Mat& n, cv::Mat& f, cv::Mat& d, std::vector<cv::Point2f>& o)
{
    oldFrame = oF;
    next = n;
    frame = f;
    dNormC = d;
    ofPoints = o;
}
void OFCluster::work()
{
    oldFrame.copyTo(next);
    for (unsigned int y = 0; y < dNormC.rows; y++)
    {
        for (unsigned int x = 0; x < dNormC.cols; x++)
        {
            if (dNormC.at<double>(y, x) >= eSob->eGUI->getCValue() || dNormC.at<double>(y, x) <= 0 - eSob->eGUI ↙
    ->getCValue())
                ofPoints.push_back(cv::Point2f(x, y));
        }
    }

    std::cout << "Optical flow points " << ofPoints.size() << std::endl;
    if (ofPoints.size() > 0)
        eSob->edgeOpticalFlow(frame, next, ofPoints);
}
```

```
/*******************************************************************************************

    This file has been produced at the Intelligent Systems Research Laboratory at
    InfoLab21, Lancaster University under the supervision of Professor Plamen Angelov.
    Reproduction of the code is permitted for academic and research purposes only without
    the express permission of the author. All code taken from this document must have this
    header at the top of the new source file.

    Filename:       RDE_class.cpp
    Author:         Gruffydd Morris
    Date Created:   27/01/2014
    Description:    This file contains the implementation of the WIDE class functions. The
                    algorithm calculations occur in this file. This class does not have
                    thresholding incorporated. The output is a density gradient in grayscale.

    ___________
    Amendments
    ****************************************************************************
    *   Date    |   Description of change                       |   Initials    *
    *           |                                               |               *
    * 12-02-14  |   Addition of WIDE implementation that    |       GM      *
    *           |   can be called from the thread class         |               *
    *           |   Addition of overloaded init_RDE function    |               *
    * 14-02-14  |   Updated result to the MAT_SIZE type and     |       GM      *
    *           |   the floating point declarations to          |               *
    *           |   RDE_PRECISION defined in DataSizes.h. This  |               *
    *           |   allows for easy changing of the data types  |               *
    *           |   stored in the Mat array, and precision of   |               *
    *           |   the RDE output without having to            |               *
    *           |   change the entire program                   |               *
    ****************************************************************************

*******************************************************************************************/

#include <stdio.h>
#include <iostream>
#include <math.h>
#include <process.h>

#include "RDE_class.h"
#include <Utils.h>

RDE::RDE()
{
    this->currPtr = 1;
}

RDE::RDE(cv::Mat& frameIn, cv::Mat& oFrame, unsigned int dim, unsigned int rowS, unsigned int reset, HANDLE
    sObj) : Threaded(sObj)
{
    this->currPtr = 1;
    this->initFlag = false;
    this->rst = reset;
    this->rowSz = rowS;
    this->frameNumber = 1;


    if (dim == HORZ)
    {
        this->in = frameIn;
        this->out = oFrame;
        this->dimension = dim;
        this->set_trails_output(oFrame.col(0).data);
        this->init_RDE_reset(frameIn.rows, frameIn.channels(), frameIn.col(0).data, in.cols);
    }
    else if (dim == VERT)
    {
        this->in = frameIn;
        this->out = oFrame;
        this->dimension = dim;
```

```
        this->set_trails_output(oFrame.row(0).data);
        this->init_RDE_reset(frameIn.cols, frameIn.channels(), frameIn.row(0).data);
    }
    else
    {
        this->in = frameIn;
        this->out = oFrame;
        this->dimension = dim;
        this->set_trails_output(oFrame.data);
        this->init_RDE_reset(frameIn.rows * frameIn.cols, frameIn.channels(), frameIn.data);
    }
}

void RDE::work()
{
    if (dimension != TIME_DOMAIN)
    {
        if (dimension == HORZ)
        {
            for (unsigned int x = 1; x < this->in.cols; x++)
            {
                this->set_trails_output(this->out.col(x).data);
                this->update_RDE_reset(this->in.col(x).data, x, in.cols);
            }
        }
        else
        {
            for (unsigned int y = 1; y < this->in.rows; y++)
            {
                this->set_trails_output(this->out.row(y).data);
                this->update_RDE_reset(this->in.row(y).data, y);
            }
        }
    }
    else
    {
        if (frameNumber == 1)
            this->frameNumber++;
        else
        {
            this->update_RDE_reset(this->in.data, this->frameNumber);
            frameNumber++;
        }
    }
}
/* This is run at the start to create a new RDE frame, or when a new "window" begins */
int RDE::init_RDE(int num_data_points, int num_features, unsigned char* data)
{
    data_points = num_data_points;
    features = num_features;

    if (initFlag == false)
    {
        RDE_storage = (RDE_PRECISION*)calloc(data_points * (features + 1), sizeof(RDE_PRECISION));
        initFlag = true;
    }
    else
    {
        memset(RDE_storage, 0, sizeof(RDE_PRECISION) * data_points * (features + 1));
    }

    /* Creates int variables for loops and offsets */
    int i, j, originalOffset, rdeOffset;

    /* Creates double variables to store intermediate data */
    RDE_PRECISION x_eucl_norm = 0, xMu_eucl_dist = 0, mu_eucl_norm = 0, data_value = 0, result = 0;

    /* Loops through each row of pixels in the video stream */
    for(i = 0; i < data_points; i++)
```

```
    {
        /* Creates offsets due to different number of channels */
        originalOffset = i * features;
        rdeOffset = i * (features + 1);

        /* Resets intermediate values for each pixel */
        x_eucl_norm = 0;
        xMu_eucl_dist = 0;
        mu_eucl_norm = 0;

        /* For each channel per pixel column */
        for (j = 0; j < features; j++)
        {
            /* Normalizes the pixel value */
            data_value = (static_cast<RDE_PRECISION>(data[originalOffset + j])-MIN_DATA_VALUE) / (MAX_DATA_VALUE
    -MIN_DATA_VALUE);

            /* Set the mean to equal the current value of the pixel */
            RDE_storage[rdeOffset + j] = data_value;

            /* Update the scalar product which is the Standard Euclidean Norm of the original pixel values */
            RDE_storage[rdeOffset + features] += (data_value * data_value);

            /* Calculates standard Euclidean distance between x and Mu for density (should be zero) */
            xMu_eucl_dist += (data_value - RDE_storage[rdeOffset + j]) * (data_value - RDE_storage[rdeOffset +
    j]);

            /* Need to calculate standard euclidea norm of mu for density */
            mu_eucl_norm += (RDE_storage[rdeOffset + j] * RDE_storage[rdeOffset + j]);
        }

        /* Calculates density, should all equal 1 on the first run through */
        result = (1 / (1 + xMu_eucl_dist + RDE_storage[rdeOffset + features] - mu_eucl_norm));
        result *= 255;

        RDE_result[i] = static_cast<MAT_SIZE>(result);
    }
    return 0;
}

int RDE::init_RDE_reset(int num_data_points, int num_features, unsigned char* data)
{
    data_points = num_data_points;
    features = num_features;

    if (initFlag == false)
    {
        RDE_storage = (RDE_PRECISION*)calloc(data_points * (features + 1), sizeof(RDE_PRECISION));
        initFlag = true;
    }
    else
    {
        memset(RDE_storage, 0, sizeof(RDE_PRECISION) * data_points * (features + 1));
    }

    /* Allocate sufficient memory to the history - to store all pixel values up to the frame number we want
    (rst) */
    histMem = (RDE_PRECISION*)calloc(data_points * features * rst, sizeof(RDE_PRECISION));

    /* Creates int variables for loops and offsets */
    int i, j, originalOffset, rdeOffset;

    /* Creates double variables to store intermediate data */
    RDE_PRECISION x_eucl_norm = 0, xMu_eucl_dist = 0, mu_eucl_norm = 0, data_value = 0, result = 0;

    /* Loops through each row of pixels in the video stream */
    for(i = 0; i < data_points; i++)
    {
        /* Creates offsets due to different number of channels */
```

```
        originalOffset = i * features;
        rdeOffset = i * (features + 1);

        /* Resets intermediate values for each pixel */
        x_eucl_norm = 0;
        xMu_eucl_dist = 0;
        mu_eucl_norm = 0;

        /* For each channel per pixel column */
        for (j = 0; j < features; j++)
        {
            /* Normalizes the pixel value */
            data_value = (static_cast<RDE_PRECISION>(data[originalOffset + j])-MIN_DATA_VALUE) / (MAX_DATA_VALUE ↙
    -MIN_DATA_VALUE);

            /* Assign the initial value to the first history block of the pixel history memory */
            histMem[originalOffset + j] = data_value;

            /* Set the mean to equal the current value of the pixel */
            RDE_storage[rdeOffset + j] = data_value;

            /* Update the scalar product which is the Standard Euclidean Norm of the original pixel values */
            RDE_storage[rdeOffset + features] += (data_value * data_value);

            /* Calculates standard Euclidean distance between x and Mu for density (should be zero) */
            xMu_eucl_dist += (data_value - RDE_storage[rdeOffset + j]) * (data_value - RDE_storage[rdeOffset +  ↙
    j]);

            /* Need to calculate standard euclidea norm of mu for density */
            mu_eucl_norm += (RDE_storage[rdeOffset + j] * RDE_storage[rdeOffset + j]);
        }

        /* Calculates density, should all equal 1 on the first run through */
        result = (1 / (1 + xMu_eucl_dist + RDE_storage[rdeOffset + features] - mu_eucl_norm));

        RDE_result[i] = static_cast<MAT_SIZE>(result);
    }
    return 0;
}

int RDE::init_RDE_reset(int num_data_points, int num_features, unsigned char* data, unsigned int orientation)
{
    data_points = num_data_points;
    features = num_features;

    if (initFlag == false)
    {
        RDE_storage = (RDE_PRECISION*)calloc(data_points * (features + 1), sizeof(RDE_PRECISION));
        initFlag = true;
    }
    else
    {
        memset(RDE_storage, 0, sizeof(RDE_PRECISION) * data_points * (features + 1));
    }

    /* Allocate sufficient memory to the history - to store all pixel values up to the frame number we want    ↙
    (rst) */
    histMem = (RDE_PRECISION*)calloc(data_points * features * rst, sizeof(RDE_PRECISION));

    /* Creates int variables for loops and offsets */
    int i, j, originalOffset, rdeOffset, memOffset;

    /* Creates double variables to store intermediate data */
    RDE_PRECISION x_eucl_norm = 0, xMu_eucl_dist = 0, mu_eucl_norm = 0, data_value = 0, result = 0;

    /* Loops through each row of pixels in the video stream */
    for(i = 0; i < data_points; i++)
    {
        /* Creates offsets due to different number of channels */
```

```
        originalOffset = (orientation * i * features);
        memOffset = i * features;
        rdeOffset = i * (features + 1);

        /* Resets intermediate values for each pixel */
        x_eucl_norm = 0;
        xMu_eucl_dist = 0;
        mu_eucl_norm = 0;

        /* For each channel per pixel column */
        for (j = 0; j < features; j++)
        {
            /* Normalizes the pixel value */
            data_value = (static_cast<RDE_PRECISION>(data[originalOffset + j])-MIN_DATA_VALUE) / (MAX_DATA_VALUE ↙
    -MIN_DATA_VALUE);

            /* Assign the initial value to the first history block of the pixel history memory */
            histMem[memOffset + j] = data_value;

            /* Set the mean to equal the current value of the pixel */
            RDE_storage[rdeOffset + j] = data_value;

            /* Update the scalar product which is the Standard Euclidean Norm of the original pixel values */
            RDE_storage[rdeOffset + features] += (data_value * data_value);

            /* Calculates standard Euclidean distance between x and Mu for density (should be zero) */
            xMu_eucl_dist += (data_value - RDE_storage[rdeOffset + j]) * (data_value - RDE_storage[rdeOffset +  ↙
    j]);

            /* Need to calculate standard euclidea norm of mu for density */
            mu_eucl_norm += (RDE_storage[rdeOffset + j] * RDE_storage[rdeOffset + j]);
        }

        /* Calculates density, should all equal 1 on the first run through */
        result = (1 / (1 + xMu_eucl_dist + RDE_storage[rdeOffset + features] - mu_eucl_norm));

        RDE_result[i] = static_cast<MAT_SIZE>(result);
    }
    return 0;
}

/* Frees the calloc'ed memory */
int RDE::close_RDE()
{
    free(RDE_storage);

    return 0;
}

int RDE::update_RDE(unsigned char* data, int iteration)
{
    /* Creates loop counters and variables for offsets */
    int i, j, originalOffset, rdeOffset;

    /* Creates variables for storing intermediate results */
    RDE_PRECISION x_eucl_norm = 0, xMu_eucl_dist = 0, mu_eucl_norm = 0, data_value = 0, result = 0;

    /* For each row of pixels */
    for(i = 0; i < data_points; i++)
    {
        /* Creates offsets due to different number of channels */
        originalOffset = i * features;
        rdeOffset = i * (features + 1);

        /* Resets intermediate values for each pixel */
        x_eucl_norm = 0;
        xMu_eucl_dist = 0;
        mu_eucl_norm = 0;
```

```
        /* Loops through each channel */
        for (j = 0; j < features; j++)
        {
            /* Normalizes the pixel value */
            data_value = (static_cast<RDE_PRECISION>(data[originalOffset + j])-MIN_DATA_VALUE) / (MAX_DATA_VALUE ↙
    -MIN_DATA_VALUE);

            /* Update the mean for each colour channel */
            RDE_storage[rdeOffset + j] = update_mean(data_value, RDE_storage[rdeOffset + j], iteration);

            /* Calculate the Standard Euclidean Norm for each pixel value for calculation of Scalar Product */
            x_eucl_norm += (data_value * data_value);

            /* Calculates Standard Euclidean Distance between x and Mu for Density */
            xMu_eucl_dist += (data_value - RDE_storage[rdeOffset + j]) * (data_value - RDE_storage[rdeOffset +  ↙
    j]);

            /* Need to calculate Standard Euclidean Norm of the means for Density */
            mu_eucl_norm += (RDE_storage[rdeOffset + j] * RDE_storage[rdeOffset + j]);
        }
        /* Updates the scalar product */
        RDE_storage[rdeOffset + features] = update_scalar_product(x_eucl_norm, RDE_storage[rdeOffset + features]↙
    ,iteration);

        /* Calculates the new RDE */
        result = (1 / (1 + xMu_eucl_dist + RDE_storage[rdeOffset + features] - mu_eucl_norm));

#ifdef MAT_NOFP
        result *= 255;
#endif

        if (result > 170)
            result = 255;
        else
            result = 0;

        RDE_result[i] = static_cast<MAT_SIZE>(result);
    }

    return 0;
}

int RDE::update_RDE_reset(unsigned char* data, int iteration)
{
    /* Creates loop counters and variables for offsets */
    int i, j, originalOffset, rdeOffset;

    /* Creates variables for storing intermediate results */
    RDE_PRECISION x_eucl_norm = 0, xMu_eucl_dist = 0, mu_eucl_norm = 0, data_value = 0, result = 0,           ↙
    old_eucl_norm = 0;

    /* For each row of pixels */
    for(i = 0; i < data_points; i++)
    {
        /* Creates offsets due to different number of channels */
        originalOffset = i * features;
        rdeOffset = i * (features + 1);

        /* Resets intermediate values for each pixel */
        x_eucl_norm = 0;
        xMu_eucl_dist = 0;
        mu_eucl_norm = 0;
        old_eucl_norm = 0;

        /* Loops through each channel */
        for (j = 0; j < features; j++)
        {
            /* Normalizes the pixel value */
            data_value = (static_cast<RDE_PRECISION>(data[originalOffset + j])-MIN_DATA_VALUE) / (MAX_DATA_VALUE ↙
```

```
    -MIN_DATA_VALUE);

            if (iteration > rst)
            {
                /* Update the mean for each colour channel */
                RDE_storage[rdeOffset + j] = update_mean_reset(data_value, RDE_storage[rdeOffset + j], histMem ↙
    [originalOffset + (currPtr * data_points * features) + j]);

                /* Calculate the old euclidean norm that we want to remove during reset, must do it here because↙
     we're going to overwrite
                   the old data feature with the new one in the next line */
                old_eucl_norm += histMem[originalOffset + (currPtr * data_points * features) + j] * histMem     ↙
    [originalOffset + (currPtr * data_points * features) + j];
            }
            else
            {
                /* Update the mean for each colour channel */
                RDE_storage[rdeOffset + j] = update_mean(data_value, RDE_storage[rdeOffset + j], iteration);
            }

            /* Assign the current value to the history block in accordance to where the pointer is for the      ↙
    history block*/
            histMem[originalOffset + (currPtr * data_points * features) + j] = data_value;

            /* Calculate the Standard Euclidean Norm for each pixel value for calculation of Scalar Product */
            x_eucl_norm += (data_value * data_value);

            /* Calculates Standard Euclidean Distance between x and Mu for Density */
            xMu_eucl_dist += (data_value - RDE_storage[rdeOffset + j]) * (data_value - RDE_storage[rdeOffset +  ↙
    j]);

            /* Need to calculate Standard Euclidean Norm of the means for Density */
            mu_eucl_norm += (RDE_storage[rdeOffset + j] * RDE_storage[rdeOffset + j]);
        }
        /* Updates the scalar product */
        if (iteration > rst)
            RDE_storage[rdeOffset + features] = update_scalar_product_reset(x_eucl_norm, RDE_storage[rdeOffset +↙
     features], old_eucl_norm);
        else
            RDE_storage[rdeOffset + features] = update_scalar_product(x_eucl_norm, RDE_storage[rdeOffset +      ↙
    features], iteration);

        /* Calculates the new RDE */
        result = (1 / (1 + xMu_eucl_dist + RDE_storage[rdeOffset + features] - mu_eucl_norm));

#ifdef MAT_NOFP
        result *= 255;
#endif

        //if (result < 200)
            RDE_result[i] = static_cast<MAT_SIZE>(result);
        //else
            //RDE_result[i] = 0;
    }

    // Add one to the history block pointer to point to the next block to store the next frame pixel values
    currPtr++;

    // If the history block pointer is the same size as the history, reset it to the start of the array
    if (currPtr == static_cast<unsigned>(rst))
        currPtr = 0;

    return 0;
}

int RDE::update_RDE_reset(unsigned char* data, int iteration, unsigned int orientation)
{
    /* Creates loop counters and variables for offsets */
    int i, j, originalOffset, rdeOffset, memOffset;
```

```
    /* Creates variables for storing intermediate results */
    RDE_PRECISION x_eucl_norm = 0, xMu_eucl_dist = 0, mu_eucl_norm = 0, data_value = 0, result = 0,
    old_eucl_norm = 0;

    /* For each row of pixels */
    for(i = 0; i < data_points; i++)
    {
        /* Creates offsets due to different number of channels */
        originalOffset = (orientation * i * features);
        memOffset = i * features;
        rdeOffset = i * (features + 1);

        /* Resets intermediate values for each pixel */
        x_eucl_norm = 0;
        xMu_eucl_dist = 0;
        mu_eucl_norm = 0;
        old_eucl_norm = 0;

        /* Loops through each channel */
        for (j = 0; j < features; j++)
        {
            /* Normalizes the pixel value */
            data_value = (static_cast<RDE_PRECISION>(data[originalOffset + j])-MIN_DATA_VALUE) / (MAX_DATA_VALUE
    -MIN_DATA_VALUE);

            if (iteration > rst)
            {
                /* Update the mean for each colour channel */
                RDE_storage[rdeOffset + j] = update_mean_reset(data_value, RDE_storage[rdeOffset + j], histMem
    [memOffset + (currPtr * data_points * features) + j]);

                /* Calculate the old euclidean norm that we want to remove during reset, must do it here because
     we're going to overwrite
                   the old data feature with the new one in the next line */
                old_eucl_norm += histMem[memOffset + (currPtr * data_points * features) + j] * histMem[memOffset
     + (currPtr * data_points * features) + j];
            }
            else
            {
                /* Update the mean for each colour channel */
                RDE_storage[rdeOffset + j] = update_mean(data_value, RDE_storage[rdeOffset + j], iteration);
            }

            /* Assign the current value to the history block in accordance to where the pointer is for the
    history block*/
            histMem[memOffset + (currPtr * data_points * features) + j] = data_value;

            /* Calculate the Standard Euclidean Norm for each pixel value for calculation of Scalar Product */
            x_eucl_norm += (data_value * data_value);

            /* Calculates Standard Euclidean Distance between x and Mu for Density */
            xMu_eucl_dist += (data_value - RDE_storage[rdeOffset + j]) * (data_value - RDE_storage[rdeOffset +
    j]);

            /* Need to calculate Standard Euclidean Norm of the means for Density */
            mu_eucl_norm += (RDE_storage[rdeOffset + j] * RDE_storage[rdeOffset + j]);
        }
        /* Updates the scalar product */
        if (iteration > rst)
            RDE_storage[rdeOffset + features] = update_scalar_product_reset(x_eucl_norm, RDE_storage[rdeOffset +
     features], old_eucl_norm);
        else
            RDE_storage[rdeOffset + features] = update_scalar_product(x_eucl_norm, RDE_storage[rdeOffset +
    features], iteration);

        ///* Calculates the new RDE */
        result = (1 / (1 + xMu_eucl_dist + RDE_storage[rdeOffset + features] - mu_eucl_norm));
```

```
#ifdef MAT_NOFP
        result *= 255;
#endif


        //if (result < 200)
            RDE_result[(i * orientation)] = static_cast<MAT_SIZE>(result);
        //else
            //RDE_result[i] = 0;
    }

    // Add one to the history block pointer to point to the next block to store the next frame pixel values
    currPtr++;

    // If the history block pointer is the same size as the history, reset it to the start of the array
    if (currPtr == static_cast<unsigned>(rst))
        currPtr = 0;

    return 0;
}

int RDE::set_trails_output(unsigned char* output)
{
    RDE_result = (MAT_SIZE*)output;
    return 0;
}

RDE_PRECISION RDE::update_mean(RDE_PRECISION data_value, RDE_PRECISION old_mean, int iteration)
{
    RDE_PRECISION result = 0;
    result = ((static_cast<RDE_PRECISION>(iteration) - 1)/static_cast<RDE_PRECISION>(iteration))*old_mean;
    result += (1/static_cast<RDE_PRECISION>(iteration))*data_value;
    return result;
}

RDE_PRECISION RDE::update_scalar_product(RDE_PRECISION euclidean_norm, RDE_PRECISION old_scalar, int iteration)
{
    RDE_PRECISION result = 0;
    result = ((static_cast<RDE_PRECISION>(iteration) - 1)/static_cast<RDE_PRECISION>(iteration)) * old_scalar;
    result += (1/(static_cast<RDE_PRECISION>(iteration))) * euclidean_norm;
    return result;
}

RDE_PRECISION RDE::update_mean_reset(RDE_PRECISION data_value, RDE_PRECISION old_mean, RDE_PRECISION
    oldest_value)
{
    RDE_PRECISION result = 0;
    result = (static_cast<RDE_PRECISION>(rst * old_mean) - oldest_value + data_value) / static_cast
    <RDE_PRECISION>(rst);
    return result;
}

RDE_PRECISION RDE::update_scalar_product_reset(RDE_PRECISION euclidean_norm, RDE_PRECISION old_scalar,
    RDE_PRECISION oldest_value)
{
    RDE_PRECISION result = 0;
    result = (static_cast<RDE_PRECISION>(rst * old_scalar) - oldest_value + euclidean_norm) / static_cast
    <RDE_PRECISION>(rst);
    return result;
}

RDEInstance::RDEInstance(cv::Mat& frame, unsigned int numThreads, unsigned int type, unsigned int reset, HANDLE
    sObj) : numT(numThreads), rdeType(type), resetValue(reset), Threaded(sObj)
{

#ifndef MAT_NOFP
    showRDE = cv::Mat::zeros(frame.rows, frame.cols, CV_64FC1);
#else
    showRDE = cv::Mat::zeros(frame.rows, frame.cols, CV_8UC1);
```

```
#endif

    blockDivider(frame, blocks, numT, rdeType);             // Divide the input frame into blocks
    blockDivider(showRDE, resultBlocks, numT, rdeType);     // Divide the result frame into blocks

    myRDE = (RDE**)std::calloc(blocks.size(), sizeof(RDE*));    // Initialise the RDE array pointer
    pThreadIDs = (HANDLE*)std::calloc(blocks.size(), sizeof(HANDLE));

    for (unsigned int i = 0; i < blocks.size(); i++)
    {
        pThreadIDs[i] = CreateEvent(NULL, true, false, NULL);
        myRDE[i] = new RDE(blocks[i], resultBlocks[i], rdeType, frame.rows, resetValue, pThreadIDs[i]);
    }
}

RDEInstance::~RDEInstance()
{
    showRDE.release();
    blocks.clear();
    resultBlocks.clear();
}

void RDEInstance::work()
{
    for (unsigned int i = 0; i < blocks.size(); i++)
    {
        myRDE[i]->run();
    }

    DWORD dwMulti = WaitForMultipleObjects((DWORD)blocks.size(), pThreadIDs, true, INFINITE);

    switch (dwMulti)
    {
        case WAIT_OBJECT_0:
        {
            for (unsigned int i = 0; i < blocks.size(); i++)            // Loop through each mutex, releasing 
    each one ready for the next frame loop
            {
                ResetEvent(pThreadIDs[i]);
                //std::cout << "Event reset " << i << std::endl;
            }
            break;
        }

        default:
            std::cout << "Error waiting for event " << std::endl;
    }
}

cv::Mat& RDEInstance::getResult()
{
    return this->showRDE;
}
```