# A novel approach to recognition of the detected moving objects in non-stationary background using heuristics and colour measurements

---

A thesis presented in partial fulfilment of the requirement for the degree of

Master of Engineering

At

Massey University,

Albany, New Zealand

Kartikay Lal

2017

# Abstract

Computer vision has become a growing area of research which involves two fundamental steps, object detection and object recognition. These two steps have been implemented in real world scenarios such as video surveillance systems, traffic cameras for counting cars, or more explicit detection such as detecting faces and recognizing facial expressions. Humans have a vision system that provides sophisticated ways to detect and recognize objects. Colour detection, depth of view and our past experience helps us determine the class of objects with respect to object's size, shape and the context of the environment. Detection of moving objects on a non-stationary background and recognizing the class of these detected objects, are tasks that have been approached in many different ways. However, the accuracy and efficiency of current methods for object detection are still quite low, due to high computation time and memory intensive approaches. Similarly, object recognition has been approached in many ways but lacks the perceptive methodology to recognise objects.

This thesis presents an improved algorithm for detection of moving objects on a non-stationary background. It also proposes a new method for object recognition. Detection of moving objects is initiated by detecting SURF features to identify unique keypoints in the first frame. These keypoints are then searched through individually in another frame using cross correlation, resulting in a process called optical flow. Rejection of outliers is performed by using keypoints to compute global shift of pixels due to camera motion, which helps isolate the points that belong to the moving objects. These points are grouped into clusters using the proposed improved clustering algorithm. The clustering function is capable of adapting to the search radius around a feature point by taking the average Euclidean distance between all the feature points into account. The detected object is then processed through colour measurement and heuristics. Heuristics provide context of the surroundings to recognize the class of the object based upon the object's size, shape and the environment it is in. This gives object recognition a perceptive approach.

Results from the proposed method have shown successful detection of moving objects in various scenes with dynamic backgrounds achieving an efficiency for object detection of over 95% for both indoor and outdoor scenes. The average processing time was computed to be around 16.5 seconds which includes the time taken to detect objects, as well as recognize them. On the other hand, Heuristic and colour based object recognition methodology achieved an efficiency of over 97%.

# Acknowledgements

First and foremost, I would like to thank my supervisor Dr. Khalid Arif for his support and valuable insights throughout the duration of the thesis.
I would also like to thank my family for their support and understanding.

# Contents

# List of Figures

# List of Tables



# List of Code Snippets

# Chapter 1 Introduction

## 1.1 Background

Humans constantly perceive the changing world with apparent ease. Our eyes are able to detect an entire spectrum of colours that helps us to perceive our environment in an orderly manner. Our stereo vision helps us determine the depth of view and helps us understand what is closer to us. Our vision system detects objects in a subtle way and combined with colour detection and object recognition, the actions of constantly observing, perceiving, engaging and recognizing objects around us are unconscious. The colour of an object would vary in different contrasts and brightness levels. Additionally, image processing is restricted to recognising the object itself. However, objects are usually defined by their environment. For example, a red vase of flowers on a white table. A red cylindrical object on a white surface may not necessary mean that this object is always going to be a vase on a table. It could very well be a pen holder or a glass. According to the book by Richard Szeliski [1], our vision system and our brain do not only detect the object of interest, but also process the information that we perceive from the environment around the object, which in this case would be a room with a white surface and a red object on it. By combining the context as well as the object's shape and colour, we arrive at a decision that the red cylindrical object is definitely a vase of flowers.

Computer vision is an interesting topic. Despite the fact that intensive research on computer vision has been going on for a few decades, and although there has been a lot of improvement made in algorithms, it can be realized that modern image processing is still quite far from matching the precision and the performance of human vision. We perceive the world in a certain way and we can not only detect objects, but also recognize them by judging the object's shape, size and colour, as well as taking the context of the environment into account, using our intelligence. Our mind provides us with the ability to identify something out of place, or irregular. For example, animals belong in the zoo and, therefore, seeing them out on the open road would be recognised as being odd. Another example would be a picture of a dog in an office. Due to our perception and our awareness of the environment, we can perceive the painting of a dog even though no real dog is present. In computer vision, detection of

objects brings us a step closer to mammalian vision, but the detected objects would become useful if they could be recognized in terms of their class and end up with a description of what they could actually be.

Object detection and recognition have become a growing area of research, attracting attention over the past few years. Detection of objects have been commonly used in areas such as video surveillance systems, traffic cameras for counting cars, traffic speed cameras, pedestrian detection systems, or more explicit detection such as detecting faces and recognizing facial expressions. These applications usually involve images or frames acquired from a camera fixed to a point, which would result in the background being static while the objects of interest move freely in the foreground. However, lately a lot of videos are captured from smartphones and autonomous cars such as GoPro, autonomous robots and even flying UAVs, all which are capable of their own vision systems. This leads to processing video frames when the camera is also moving, which introduces ego-motion to the frames, resulting in a non-static background, and the problem of detecting moving objects stays the same.

## 1.2 Problem Statement

The algorithms that have been developed in the past for object detection on a moving background as well as object recognition, have achieved a certain degree of efficiency. The literature review indicates that the efficiency of detection of moving objects, which includes people and cars, ranges between 70% to 90%. However, the current efficiency rate may not be good enough for critical scenarios like the one encountered by Tesla car in May 2016. The car was driving on a highway when a trailer drove perpendicular to it, while autopilot was engaged. The car's vision system noticed the white side of the trailer against a brightly lit sky. The detected object was processed to be the sky so the brakes were not applied, resulting in the car crashing into the trailer. Here, the problem was caused by the vision system's inability to relate the detected object to its environment. None of the object detection algorithms in the literature review reach an efficiency of 100%. Object recognition algorithms lack the capability of identifying objects in context to their environment, which is yet to be researched.

## 1.3 Proposed solution and novelty

Based on the critical evaluation of prior research work done in the realm of object detection and recognition, it was found that various techniques have been used in the past, obtaining decent results, but at the cost of computation time. Similarly, recognition of detected objects in the scene have used large databases in the past, incorporated with searching algorithms, which are memory intensive and also time consuming. However, there is insufficient work done on context-aware object recognition. Therefore, an algorithm is developed for this research, which comprises of two main components:

1. An improved object detection algorithm using feature based optical flow.
2. A new approach to object recognition which would almost be at par with human perception by incorporating the awareness of surroundings of the detected object.

The algorithm was simulated and tested in Matlab with various sets of video frames taken from Matlab datasets, YouTube as well as BMS datasets [2].

## 1.4 Outline of Thesis

Chapter two presents the literature review focusing on different methods for object detection and object recognition. It includes a discussion of the issues with object detection methods with respect to their accuracy and quality, and also evaluate different methods used for object recognition.

Chapter three includes the algorithm design for object detection and recognition and describes the methodology which is split into two parts. Sections 3.1 – 3.7 are linked to object detection and Sections 3.8 and 3.9 relate to object recognition with references to Matlab code in each section.

Chapter four shows the results of the algorithm which was developed, as well the comparison with other works.

Chapter five concludes the thesis along with recommendation for future work.

# Chapter 2 Literature Review

This chapter presents a review of the literature focusing on two topics: detecting moving objects on a non-stationary background and recognition of detected objects relative to its shape, size and the environment. The purpose of this review is to present the existing state of development and find appropriate methods or algorithms suitable for real-time implementation, which is computationally less expensive.

Shown in Figure 2-1 is a generic and most simplified diagram of detection of moving objects on a non-stationary background. The general process starts with a stream of video frames which pass through a number of steps such as feature extraction, optical flow, clustering, etc., along with various methods for each step that provides a base structure for detecting objects, which has been adopted by a number of researchers. However, there has been a lot of work done in the areas of object detection and recognition with many different variants to achieve both aspects of image processing, which have been studied in this literature.

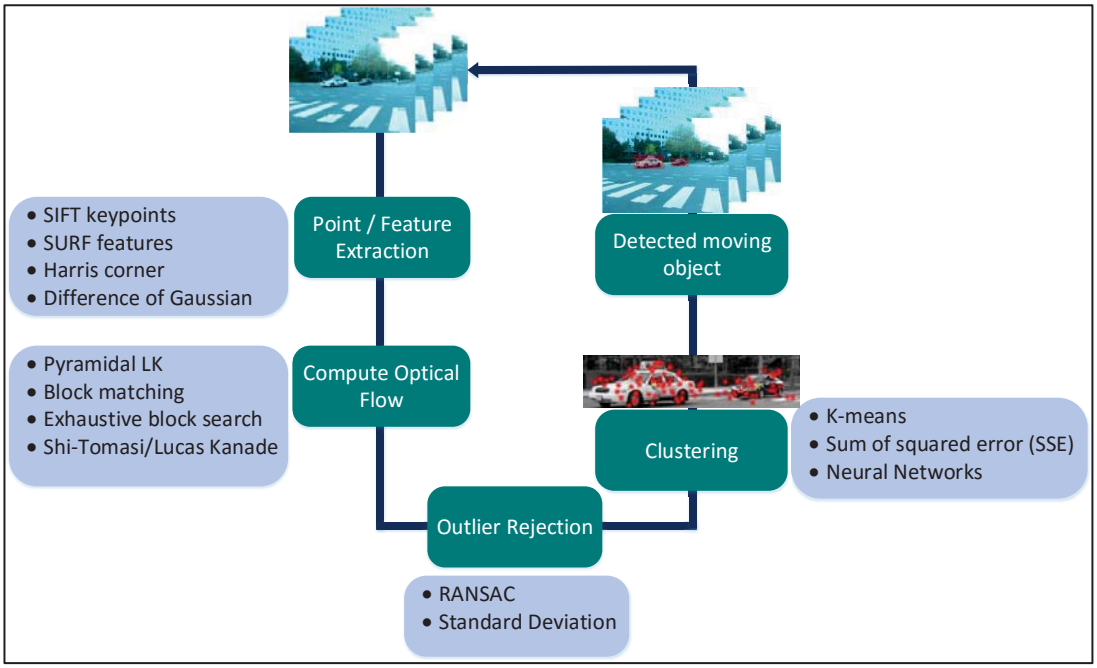



Figure 2-1: Simplified Conceptual Diagram of moving object detection on non-stationary background

## 2.1 Image Registration, Optical Flow computation and clustering

Detection of moving objects in real world applications began back in 1994 by Koller et al. [3] for the purpose of surveillance and traffic safety, where the traffic camera was stationary. Zang et al. [4] also carried out some research in this area. Feature extraction along with object tracking was their prime focus for the purpose of traffic safety and surveillance. On the other hand, pixel layer scheme was also adopted by Patwardhan et al. [5] to detect moving objects on a static background by decomposing the image into several layers. Earlier, object detection was based on detecting objects while the background was static which introduced methods like background subtraction, thresholding the error between an estimate amid images and frames and using filters and optical flow. This research aimed at developing a low-level surveillance system and vehicle tracking system, which was among the initial attempts on detecting moving objects through contour tracking, and an affine motion model based on Kalman filters to extract the object and compute its trajectories over a sequence of frames. On the other hand, Gaussian model for object detection was presented by Wren et al. [6] while Stauffer and Grimson [7], worked on a similar issue of object detection on a stationary background. However, they approached the detection technique by modelling the values of a particular pixel as a mixture of Gaussians rather than explicitly modelling the values of all the pixels in one particular type of distribution and processing the image as a whole. Elgammal et al. [8] applied kernel density estimation with the fast Gauss transform for background modelling. The model background estimation was computed as the parameters of the mixture model of each pixel change, which was the starting point of splitting the task of object detection into smaller components. Since then, a lot of work has been done in the area of background modelling also incorporated by [9-12] as well as background subtraction approached by [13-15].

Many researchers have worked on the problem of detecting moving objects when the background is also moving, which often happens when the camera is non-stationary or the video is shot from a moving camera. Detection of objects is based on feature point extraction. There is no predefined feature extraction technique that researchers should use but it solely depends on the type of image or frame, and the type of extraction technique that needs to be performed on the image. There has been a significant amount of research done on detecting moving objects on a stationary

background, but due to the rise of autonomous vehicles, vision for aerial robots, surveillance through drones and object tracking through mobile robots made it has become necessary to detect moving objects while the camera itself is also moving. Unfortunately, the methods mentioned above cannot serve well for scenarios where the observer (camera) is non-stationary, as even with slight motion of the camera these methods would not work well.

Detection of moving objects comprise of separating the background from the foreground where the objects of interest lie. Assumptions are made that the objects of interest appear in the foreground [16, 17] such as people walking in the park, while moving objects such as swaying trees or ripples on water which pertain to the background, must be excluded. The first step in detecting objects of interest is to perform feature extraction. Feature detection takes place within the image and recognizes unique information about the image. They act like the signatures of the image which help detect meaningful features, also known as feature descriptors. Work conducted by Khan et al. [18] focuses upon the detection of such objects of interest that are moving in the foreground. There are a few approaches of detecting moving objects on a non-static background.

One of these approaches is detecting unique feature points or corners in the images. Although these points provide information from both the background and the foreground, it helps to set up a foundation upon which optical flow computations are mostly processed. Bugeau et al. [19] worked on detection and segmentation of moving objects in complex dynamic scenes shot by moving cameras. First the sensor motion was computed by assuming that the apparent motion induced by the physical motion of the camera is dominant, and perform background subtraction to compute the global optical flow. Harris corner detector and automatic multidimensional bandwidth selection was chosen in the mean shift clustering algorithm to detect moving objects by simple pixel-wise motion detection, to compute the descriptor which is formed by the coordinates of the points, the motion of points and its photometric features. In short, spatial, dynamic and photometric features were used and that allowed the extraction of moving foreground objects, even in presence of illumination changes and fast variations in the background. However, Viswanath et al. [20] chose a different path to detect objects by detecting background features and proposing a solution for background modelling from a moving camera by extracting unique features from the current and model frames. The chosen method of feature extraction was Harris corner

detector, the points from which were then used to calculate homography. This gave the relation between the extracted corner points from the initial frame and the tracked corner features from the current frame. The tracking between frames was done using the Lucas-Kanade Tracker (LKT) method. The results extracted from [19] and [20] are shown in Figure 2-2. In addition to that, image homography was also adopted by Jin et al. [21] to detect moving objects with small changes in the background. The moving background was modelled by creating a panorama of the scene and performing image subtraction to arrive at the objects in motion.

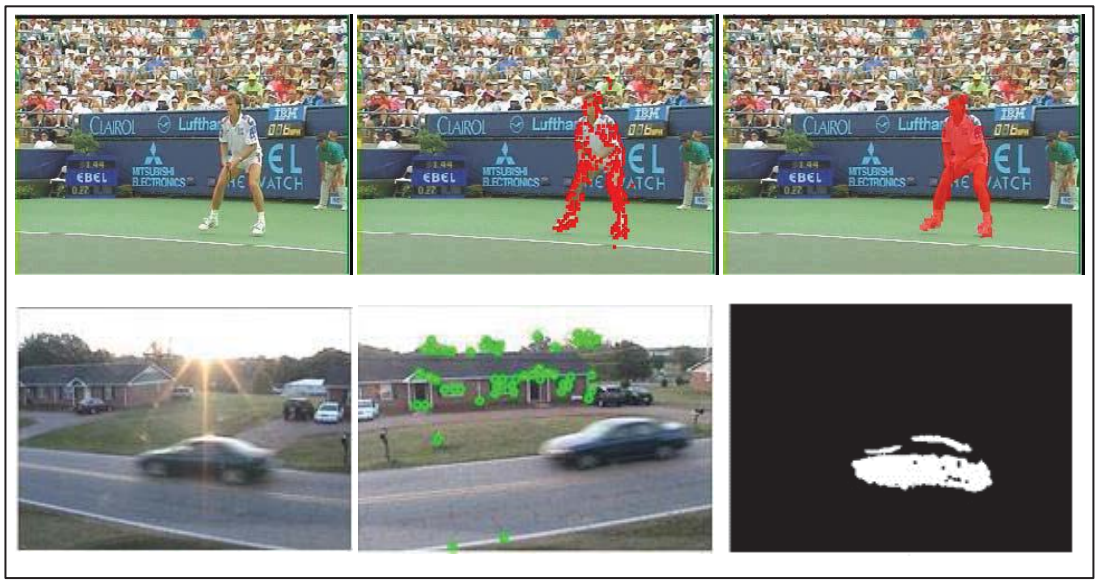


Figure 2-2: Results from [19] (top) and [20] (bottom) showing background subtraction and deleting background features respectively

A recent example of work on using Harris corner detection was presented by Kim et al. [22] who used Harris corner detector to extract points in the first frame and computed optical flow for each corner point between the first and the second frames using the pyramidal Lucas-Kanade optical flow method, followed by clustering the optical flow vectors using K-means clustering algorithm and running them through RANSAC algorithm to reject the outliers.

Harris corner detector is not the only method stated in the research [23] conducted as part of this thesis. However, it seemed to be the popular choice of feature detection algorithm because of its fast computation speed. Along the same lines of computing pyramidal Lucas-Kanade optical flow, Patel et al. [24] proposed an algorithm which comprised of Bilateral filtering of the image to smooth the image without blurring the edges or degrading the image, and then computing pixel wise optical flow using pyramidal Lucas-Kanade method followed by thresholding of the motion vectors to detect the moving objects, then performing morphological operations like dilation, erosion, etc. to arrive at the final detected moving object. The same method of pyramidal optical flow was also used in [22] and [24]using RANSAC for outlier rejection of motion vectors. By incorporating thresholding of motion vectors respectively, there could be an increase in the computation time and memory usage as pyramidal optical flow is memory intensive as explained by Sun et al. in [25]. The results obtained in [22] and [24] are shown in Figure 2-3.

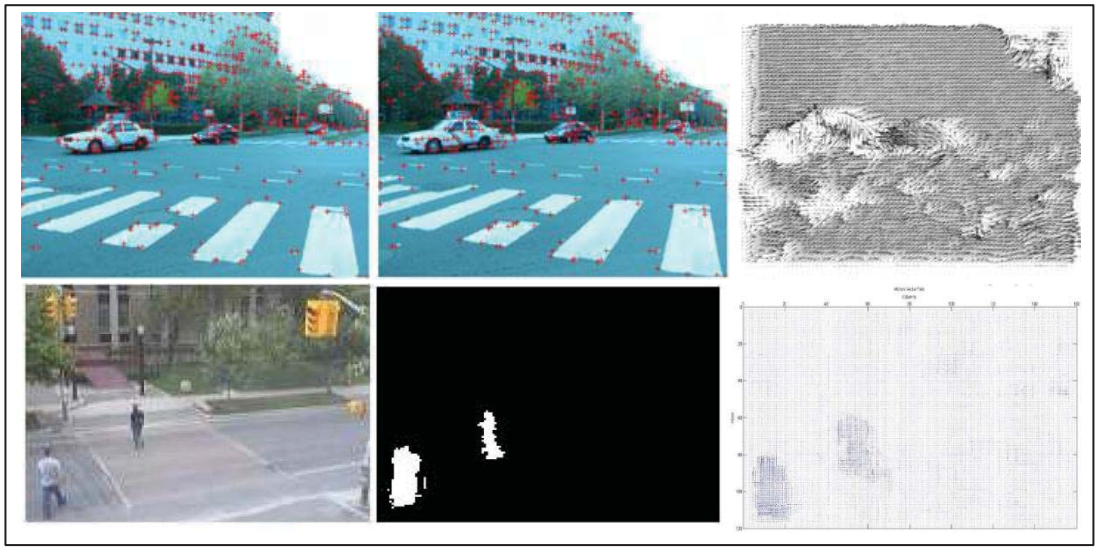


Figure 2-3: Results from [22] and [24] showing optical flow using pyramidal LK method

Unlike Harris corner detector, SIFT feature detection algorithm is known to be able to detect a lot more points compared to other detection methods [23]. This would give better results in detecting moving objects as, due to its ability of detecting an abundance of keypoints, it becomes a lot easier to detect objects of interest as shown in the paper by Sun, Huang et al. [25]. At first SIFT points were calculated to compute a consensus foreground object template for object detection, which gets stored in memory, and perform template matching for subsequent frames. But the constraint was to have objects of interest in close-up scenes. A slightly different approach was proposed by Lu et al. [26], a three-frame-difference method which works by subtracting first three frames from a video sequence assuming no background motion for the first three frames, which acts as a template. The subsequent frames were computed in pairs by performing the AND operation and comparing the result with the difference frame obtained in the beginning. SIFT feature points were extracted from this newly subtracted frame to compute the optical flow for three-frame-difference image and the image generated by AND operation. Once a moving object is detected successfully, it gets added to the template. Or if the camera motion is detected, then a new template is created. The two methods described above [25, 26] use frame differencing and template matching, which may perform fast, but would be memory intensive for storing these templates.

The second approach is compensation of camera motion by ego-motion estimation. Uemura et al. [27] proposed a feature extraction algorithm based on Kanade-Lucas Tracker to compute optical flow based on SIFT feature matching. These features were used in conjunction with image segmentation to estimate dominant features and then separated into static and moving segments regardless of camera motion, which are used for motion compensation. A slightly different approach to motion compensation using motion vectors and generic line features was proposed by Wong et al. [28] for the application of detecting moving vehicles travelling at relatively low speed. This approach was inspired by H.264/AVC video encoder and the use of generic horizontal line features that is inflicted upon most vehicles when the video is taken using a monocular moving camera while driving in the direction of motion of other vehicles. Motion vectors were computed using different inter-mode encoding techniques using only P-frames. On the other hand, an intersection safety system using dual images acquired by two individual cameras mounted on a car was proposed by Rabe, Frank et al. [29]. The dual camera technique was used to identify small distinctive image regions as well as the depth to detect features and track moving vehicles over time. For this, the Kanade-Lucas-Tomasi tracker was used which provided sub-pixel accuracy and the ability to track features robustly for a long sequence of images. The proposed system of using dual cameras enabled one of the cameras to provide ego-motion compensation to predict the object's position, taking into account its current speed of motion, and the other provided depth. The results that were obtained showed that this setup could detect slight movement of pedestrians and vehicles while providing the distance between the cameras and the objects.

Other approaches of object detection include frame coupling researched by Chen et al. [30] proposing a method based on template matching and frame coupling to handle the problem of moving object detection on a moving background caused by a moving camera. To compute motion parameters, SURF feature points were extracted. The part of the image with matched points and regions crossing or overlapping between two frames was extracted using the exhaustive search method said to be called Sub-Area Extraction. The purpose of this was to select the best matching template image. To improve the accuracy of the potential object region, the method of coupling with frames was introduced into the detection process. Figure 2-4 shows the result obtained in [30] presenting two objects detected on a moving background. The

background being very smooth, not much processing would be required in these areas, providing fast processing, hence low computation time.

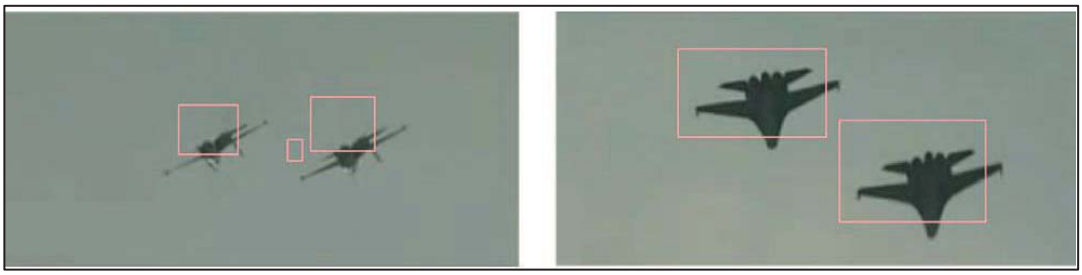


Figure 2-4: Two consecutive frames with objects detected that were used by [30]

On the other hand is sampling strategy based on the foreground probability map using the spatio-temporal properties, which was proposed by Yun et al. [31]. This approach involved predicting the position of objects in the subsequent frames by assuming that the objects move smoothly in consecutive frames. In order to keep the computational efficiency of the prediction high, foreground probability was used to predict where the objects are likely to appear instead of computing accurate velocity estimation. Foreground probability was intended to distinguish between actual objects and false detections, as well as reduce the computation time taken for searching for the actual positions of the objects. Furthermore, a continuous background update system for moving camera was proposed by Zhao et al. [32]. The moving objects were detected by using an uninterrupted video sequence to be able to update background continuously. The moving objects could easily be detected even while zooming in and out of the camera. The whole scene was first captured by the camera while it was zoomed out. The frames captured by the camera while it was zoomed in, were warped onto the original frame which was taken while the camera was fully zoomed out. The algorithm incorporates the changes that occurred in the background, such as objects that were added or removed. Other approaches involved using colour information of the objects by creating colour temporal templates [33]. Likelihood-field-based vehicle measurement model and edge feature localization combined with template matching for vehicle detection and tracking were approached by driver assistance systems [34, 35].

## 2.2 Object Recognition

The analysis of recognizing objects has become an important part, which is usually researched in conjunction with the analysis of object detection. In Section 2.1, a review of work done on detection of moving objects illustrates numerous ways on how moving objects can be detected when the background itself is also moving. However, recognizing what these objects could actually be, is an entirely different task. There has been a considerable amount of research conducted on recognizing objects with methods such as Gradient-based and derivative-based matching approaches, extracted features and boosted learning algorithms. Additionally, Bag-of-features with detectors such as SURF and MSER, Template matching, Image segmentation, blob analysis and Viola-Jones algorithm were described in the survey of object detection and recognition algorithms conducted by Moeslund et al. [36]. The more recent survey conducted by Li, Weiming et al. [37] and Ballan et al. [38] found that the most commonly used techniques are Template matching and image segmentation. Majority of the work on object recognition revolves around template matching with a database, which usually consists of a few hundred if not thousands of images. Many researchers have used databases for object recognition. However, indexing scheme and matching algorithm with the database of images have been approached in various ways.

For instance, the approach by Diplaros et al. [39] combined shape and colour information to recognize objects in the scene. According to a short survey in their research, a substantial amount of the work on appearance-based object recognition was done by matching unique features in images such as edges, corners and lines between a reference frame and target frame. The researchers realized that shape features are rarely adequate, which led to the research of incorporating colour measurements along with shape detection and took to calling this approach colour-shape context. It is a histogram that combines both colour and spatial information into one unifying, indexing framework which could be accessed as one multidimensional vector. The histogram codes the area where the colour transitions are drastic in the image. Edges were detected in the images, which were used to obtain similarity invariant shape descriptors. The colour transitions that occur at the edges are coded as illumination invariance, and were used as colour information, which led to the scheme being rotation, scale, translation and aspect ratio change invariant. Furthermore, the colour-shape context has basic robustness against small amount of translation noise as

it was based on the distribution of brightness invariant derivatives instead of their exact positions. Along the same lines of colour measurements and shapes, Jie, Xiaomin et al. [40] worked on a traffic light detection system that relies solely on colour measurements and shapes of objects by extracting rectangular and circular shapes with red, yellow and green colour measurements integrated. The researchers surveyed various ways to detect traffic lights such as template matching, circular shape detection and even colour distributions. It was found that sometimes template matching does not work correctly when matching the standard template with a changeable target. The recognition rates of colour filtration algorithm were not very high and detection based on circular shape extraction was sometimes not circular because of the disturbances of the environment. Thus, a detection and recognition algorithm was proposed for a traffic light system based on multi-feature extraction integrated with the method based on shape and colour distribution. In addition to this, recognition of traffic lights according to the colour placement and the position of the circular area in the rectangular area of a traffic lights was also proposed. The results show successful detection of the traffic light as long as it was large enough with respect to the size of the test image. Similarly, Liang et al. [41] also worked on colour extraction and template matching by proposing a method which was divided into two modules; Region of Interest (ROI) extraction and recognition modules for which supervised learning was used. The ROI module converts RGB image into grayscale and performs shape matching to detect a probable road sign which could be one of three shapes; circular with red borders, triangular with red borders and blue background with white arrows for prohibitory, mandatory and danger signs respectively. Shape matching was performed against templates of grayscale road signs to find any of the three shapes stated above. Histogram of orientated gradients (HOG) along with hue and saturation histograms were used for the recognition module. Figure 2-5 shows the results obtained in [40] and [41].

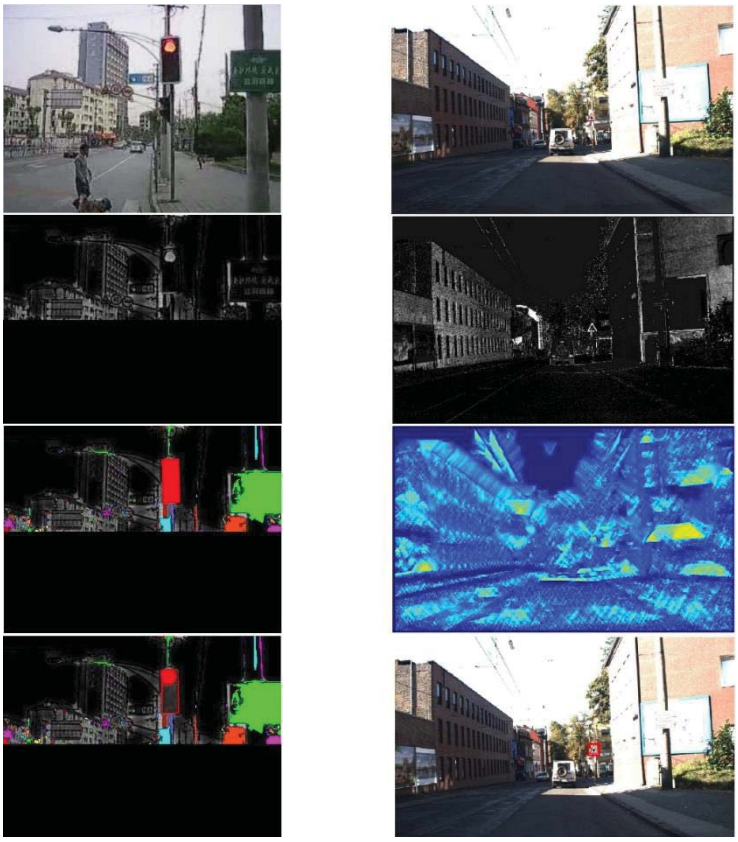


Figure 2-5: Results from [40] (left column) and [41] (right column)

Similarly, Torralba, Murphy et al. [42] detected objects by classifying indoor or outdoor environment, for example an office or playground, by presenting a set of small images that represent a part of the global structure such as a road or a street that provides relevant information for place recognition and categorization. Their hypothesis show how these small images can provide information about the context that simplify object recognition. The system was trained to recognize over 60 locations, whether indoors or outdoors, and the recognition process was divided into two parts, Global image features and Place recognition. Global image features represented properties of the scene without having to specify individual objects within a scene. It stored images of many objects and fixtures that we see and use in our day-to-day lives, for example, indoor scenes such as a coffee machine, TV, laptop, etc. and outdoor scenes such as windows, street lights, houses, letter boxes, etc. Whereas, Place recognition was built on a context-based place recognition system where colour images of size 120x160 pixels taken at different locations such as parks, streets, rooms and corridors, were all stored into a database. The database of small images taken as

part of global image features were used in a wider scene for processing place recognition. The results obtained in [42] are shown in Figure 2-6.

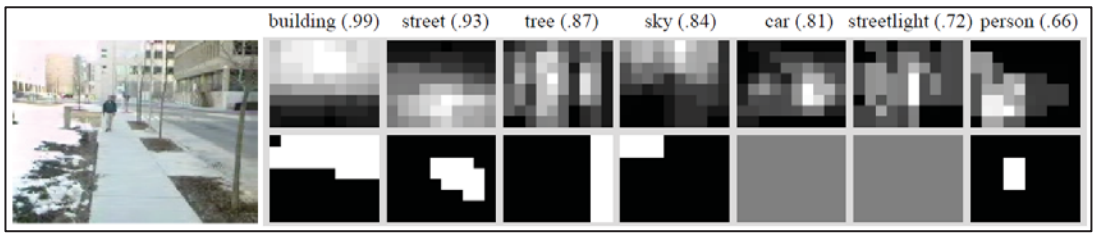



Figure 2-6: Results extracted from [42]

On the other hand, Rabinovich et al. [43] tackled object recognition by using contextual relation between object's labels to help satisfy semantic constraints by adopting the bag-of-features model but approached it by including relative interactions between objects in the scene. Image segmentation was used as a pre-processing stage for object categorization, for which SIFT feature descriptor was used to detect keypoints which were later used for matching. Test images based on segments of a scene was realized to add spatial grouping to the discriminative recognition model and it would provide for a natural representation of environmental based interactions between objects in the image. The researchers used two databases named MSRC and PASCAL with 15 and 30 training images respectively. The results were found to be computationally intensive and took about 2 to 5 minutes to compute each colour image of size 640x480 pixels. Similarly, Zitnick et al. [44] approached object recognition in a manner somewhat related to the bag-of-features technique which the researchers called bag-of-triplet-feature points. The proposed method for detection and recognition was based on triplets of feature descriptors. This method allowed for a minimum amount of feature points needed for a set number of objects. The researchers found that for 32 objects, using the ordinary bag-of-features algorithm, about 40,000 feature points were required to accurately recognize the object. However, with the use of triplet feature descriptors, this number could be reduced down to less than 1000 which was achieved by combining common feature points into a group of three called 'triplet'. To start with, SIFT feature detection algorithm was used to detect features of objects in the image and clustered them using K-means algorithm, which was then used to combine three features from each object to arrive at a triplet. While processing the recognition stage, three features at once were used to detect the object. A database was used consisting of 118 object images. Based on a modified K-means clustering

algorithm, each feature was assigned a label. Object matching was done with an inverse look up table of possible triplets. Their results as revealed in Figure 2-7 show good recognition of non-deformable objects but the quality of results was limited by the ability of the feature detector to find repeatable features in complex objects or objects with little texture.


Figure 2-7: Results extracted from [44]

The most recent work on bag-of-features or bag-of-visual words was carried out by Uçar et al. [45] proposing two Convolutions Neural Network (CNN) architectures. To start with SURF and HOG features were extracted and clustered using K-means algorithm. This was followed by dividing the test image into 9 equal patches which were used to create a feature pool using CNNs. These small patches were then added to the bag-of-visual words. The algorithm was tested against the Caltech-101 database and the Caltech pedestrian database. In the same way Ali et al. [46] proposed to detect features by combining SIFT and SURF feature detection algorithms to get better feature descriptors. The detection of features in the images was found to be insufficient using just one of the detection algorithms. Therefore, SIFT and SURF along with Principal Component Analysis (PCA) was proposed to enhance the segmentation of images into parts with dense clusters, not taking into account the size of the descriptor. These clusters were then added to the bag-of-features which were found to be most useful when matched with other images for recognition. PCA theory was introduced to decrease the SIFT feature vector, which eliminated the problem of long processing time, thereby improving the accuracy of the system. The results showed good depiction of the key feature points extracted from the images using their proposed method, but at the cost of processing time. The two most recently published papers [45, 46] which too were based on the bag-of-features technique also incorporated the extraction of key feature points using two feature detection algorithms, clustering into groups and storing them in databases was the baseline in both the approaches. Processing times and accuracy were not stated in [46], but the results from [45] claim

to have above 47% accuracy in most scenarios when the method was compared and tested against Caltech-Pedestrian database.

Feature based recognition was also one of the methods used to recognize objects. Howarth et al. [47] proposed to use feature points that depict unique parts of the object to reduce the object into a set of simple geometrical features. However before extracting the features, the image was translated into a set of contours which enhanced the image and which were the sets of adjacent points that have the same grey level. As there were many contours for each image ranging between 200 to over 1000, depending on the complexity of the object, the authors assumed that these contours would contain all the necessary components to identify the object in the image. Features were extracted one by one using a combination of Line, Arc and Lobe methods used to find the strongest features which were called 'super features'. These features were stored in a database which was then searched through using a high-speed searching algorithm, to find matches with the stored super features. In short, the authors proposed a lightweight method of computing image contours, detecting features and grouping them to capture the essentials of the object, which enables a search route to generate the likelihood that a given object exists in the image. The results show that the probability of finding a match of the extracted object in the test images is about 80%. Shown in Figure 2-8, mug is the object of interest in the scene shown in the left image followed by extraction of image contours and in the end, arrive at the super features for the object. Similarly, SURF feature points to detect local feature descriptors combined with a database of generic images with outdoor features such as windows, doors, trees, etc. was approached by Ta et al. [48].

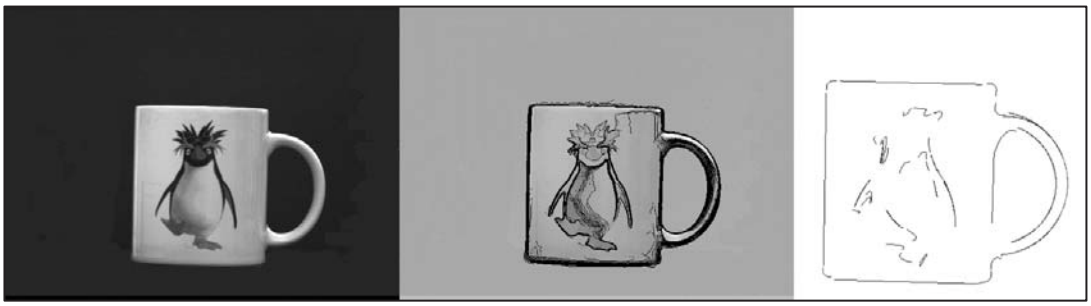


Figure 2-8: Original image, image contours, super features identified, taken from [47]

Research conducted by Ramadevi et al. [49] used a different approach to recognize objects in the scene. Different edge detection algorithms were used to evaluate how each edge detector performs in terms of segmenting parts of the image. Sobel, Canny, LoG and Roberts edge detectors were evaluated to detect the edges of grayscale images followed by implementation of object recognition algorithms. The approach to object recognition involved analysing parts of the image to decide which part or area of the image a model should occupy. Edge detection was demonstrated in this work as it was used for object recognition using image segmentation technique, as well as any of the evaluated algorithms such as Expectation-Maximisation (EM) algorithm, Otsu or Genetic algorithms. The interaction between the segmented images and object recognition was explained by the researchers in [49]. For object recognition, the hypothesis of each object based on the image area was validated and assigned to the object as well as the estimated model parameters. This indicated the familiarity of the object appearance. Expectation-maximization (EM) algorithm was used to find maximum likelihood estimates of parameters, for which the probability density function was used. Results show how edge detection along with EM or Otsu provide successful detection of objects. The results were constrained to images that displayed objects of interest covering more than 50% of the total image and a smooth background around the object was also required. Similarly, the approach by Prasad et al. [50] followed the standard machine learning process. The researchers used a set of images to train, and evaluated the edge classifiers for each object by putting together a database of about 100 images for each class. These covered a wide range of conditions such as scale, pose, illumination, partial occlusions and background clutter. Canny edge detection was used to detect edges around the objects. This was incorporated with chamfer matching for object detection and an improved version of Obj Cut algorithm for object segmentation. Only three objects across all images were chosen to be detected and these were an orange, a bottle and a banana. Results show accuracy of 94%, 63% and 61% for orange, bottle and banana respectively. Both [49] and [50] rely on edge detection techniques to detect close boundaries of objects, followed by feature matching with the help of databases of stored features. The results of both approaches output similar results. The chamfer matching method proves to be promising as it detects exact outlines of the desired object using a database of about 300 images. On the other hand, the approach in [49] does not involve the use of

database but managed to detect the object using edge detection, along with EM algorithm, as shown in the results extracted from these papers as shown in Figure 2-9.

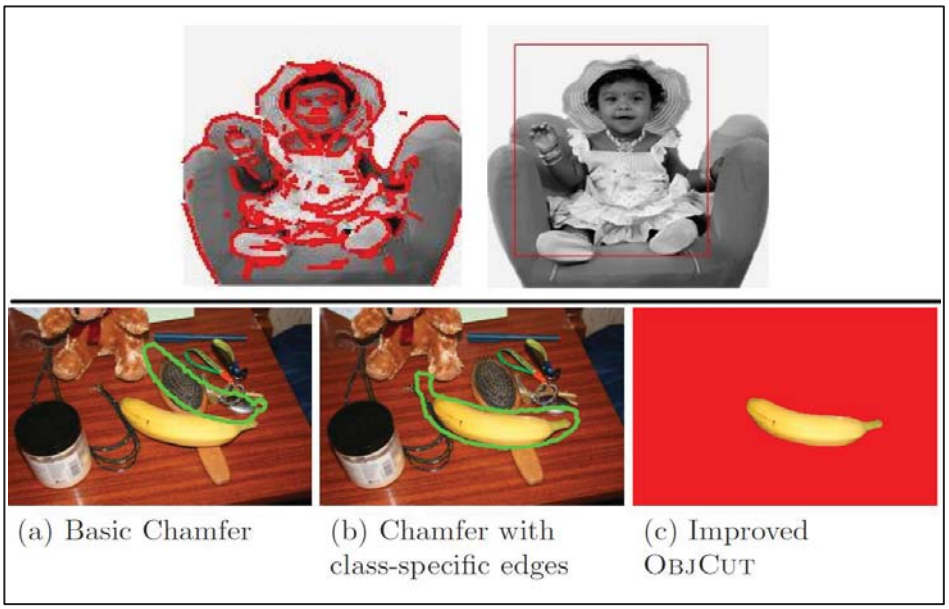



Figure 2-9: Results extracted from [49] (top) and [50] (bottom)

Another approach includes using an event recognition system based on an object's context and its motion, as studied by Sekiyama et al. [51] In this method, the system calculates the object's shape, by measuring its circularity and aspect ratio, then categorizing the object as long, thin, square or round using threshold values. If the object is long and thin, the cognitive sense demonstrates that the object could be a pen with a certain colour. To further confirm this fact, other surrounding objects that look similar are detected. which could help identify the object as a pen. For example, a rectangular object could be a writing surface such as a paper or book. The results show successful detection of objects that have sharp changes in intensity levels. These are detected together with the recognition of surrounding objects that could further improve recognition of the object under test. However, this system is susceptible to drastic changes in brightness levels which could affect detection results. On the other hand, Dheemanth [52] proposed object recognition using Eigen values to detect faces. The approach was claimed to be one of the simplest. Eigen values provide small unique features of faces which would act as a substitute for searching large databases of faces. The approach was not aimed at perfect identification but rather intended to achieve a low error matching rate. Similar methods were approached by V-Naquet et al. [53] and by Jauregi et al. [54]. In addition to that, a patch based approach to object

classification for surveillance was approached by Wijnhoven et al. [55]. Using surveillance videos from a CCTV camera, the detected moving objects were used as a template to compare against a dataset with 9000 object images reaching an average efficiency of 95%. Other approaches to traffic surveillance and safety are [56-58].

## 2.3 Summary of findings

The main objective of this review was to evaluate the methods and techniques used for object detection on non-stationary backgrounds as well as techniques used for object recognition. A considerable amount of research has been carried out in these two areas and many different methods were studied. Most of the common methods have been identified. These are methods such as colour and shape extraction, template matching, feature extraction, edge detection, template matching, etc. There have been a variety of success rates (mostly for a certain type of dataset) but overall there is no method that works for all different cases. Some researchers have used their own dataset whereas some used standard datasets so the comparison of results is also very difficult, in this case. There are a few different areas of improvement. It was seen that k-means clustering algorithm shows weakness for adaptability and RANSAC algorithm lacks the ability to adjust to the feature points and has potential to reject meaningful points. Also, literature showed drawbacks of using template machine and bag-of-features algorithm with huge databases which are both memory intensive and time consuming. With the existence of numerous methods for both object detection and recognition, the literature helped in narrowing the review down to four strong candidates [22], [30], [43] and [46] as shown in Table 2-1 for further investigation. Although these candidates have good results and high accuracy, their research suffers in two ways. Firstly, from not being able to compute optical flow efficiently, and secondly, their algorithm's inability to recognize objects using the context awareness approach. A few drawbacks have been listed below:

1. Clustering system: K-means clustering has been applied in many papers. However, a prevalent and time consuming problem is the algorithm's inability to allot points into a single cluster. Instead, it clusters points using fixed Euclidean distance, leading to detection of several, small clusters, and consolidating them into one cluster is time consuming.
2. Object recognition system. The use of large databases does not only involve a large database of images, but also introduces algorithms to organize and search through trained images which decreases processing times and adds to computational cost.

By adding these two components, we expect to see an improvement in the efficiency of detection of objects and improvement of the recognition efficiency without the use of large databases and searching algorithms, which can also be implemented for real-time applications.

Table 2-1: Summary of papers

| # | Author | Techniques used | Setting | Success Rate (%) | Paper Focus | Time consumption (sec) | Dataset | Platform | Reference |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Bugeau et al. | LK method for optical flow, colour categorization and MAP-MRF for object detection | outdoor | - | Object Detection | - | Personal dataset | - | [19] |
| 2 | Viswanath et al. | Harris corner extraction, feature tracking for background modelling, frame differencing | outdoor | ~74 | Object Detection | - | Personal dataset | OpenCV in V.Studio 2008 | [20] |
| 3 | Patel et al. | Lucas-Kanade optical flow computation, | outdoor | - | Object Detection | - | Personal dataset | - | [24] |
| 4 | Kim, Jiman, et al. | Motion compensation, Delaunay triangulation, Clustering through scatteredness of points | outdoor | - | Object Detection | - | DATA-I and DATA-II | OpenCV | [22] |
| 5 | Chen et al. | SURF feature point extraction, Frame coupling, Frame differencing | outdoor | ~95 | Object Detection | - | Personal dataset | - | [30] |
| 6 | Rabinovich et al. | bag-of-features, Conditional random field, using database | indoor/ outdoor | 68.4 | Object Recognition | 10-20 sec/image | Personal dataset | Matlab | [43] |
| 7 | Ali, Jun, et al. | Bag-of-words, SIFT Feature extraction, Principal Component Analysis theory, using database | indoor/ outdoor | 66.6 | Object Recognition | - | Personal dataset | OpenCV | [46] |
| 8 | Sun, Wang et al. | SIFT feature detection, Mean-Shift clustering, | outdoor | - | Foreground obj. detection | 2000 | YouTube Videos | - | [25] |
| 9 | Lu, Xu et al. | SIFT feature detection, k-means clustering | outdoor | 64.7 | Adaptive object detection | - | Personal dataset | - | [26] |
| 10 | Uemura et al. | KLT tracker and SIFT | outdoor | 65.4 | Obj. detection Feature tracking | - | KTH-action sequences | - | [27] |

| # | Author | Techniques used | Setting | Success Rate (%) | Paper Focus | Time consumption (sec) | Dataset | Platform | Reference |
|---|---|---|---|---|---|---|---|---|---|
| 11 | Wong, Siu et al. | H.264/AVC Motion vector extraction, Sobel horizontal edge detector | outdoor | ~90 | Object detection | - | Daimler AG sequences | - | [28] |
| 12 | Yun et al. | Motion compensation, background subtraction | outdoor | - | Object Detection | - | Personal dataset | OpenCV | [31] |
| 13 | Zhao et al. | Background modelling, SIFT Feature extraction, RANSAC for outlier rejection | outdoor | - | Object Detection | - | Personal dataset | - | [32] |
| 14 | Diplaros et al. | Vector based framework, colour recognition, using Database | indoor | ~95 | Object Recognition | - | COIL-101 | - | [39] |
| 15 | Torralba et al. | wavelet image decomposition, hidden Markov model, using database | outdoor | ~80 | Object Recognition | - | Personal dataset | - | [42] |
| 16 | Zitnick et al. | SIFT Feature extraction, feature matching via Triplet of features, k-means clustering, using database | indoor | 78.8 | Object Recognition | 5 to 20 | Personal dataset | - | [44] |
| 17 | Uçar et al. | Bag-of-words using SURF, HoG and K-means, Convolutional Neural Networks for deep learning, using database | indoor/ outdoor | 92.8 ± 0.43 | Object Recognition | - | Caltec 101 and Caltec pedestrian | Matlab | [45] |
| 18 | Jie, Xiaomin et al. | Colour recognition, circular shape extraction, template matching | outdoor | 89.7 | Object Recognition | 0.086 | Personal dataset | - | [40] |
| 19 | Howarth et al. | Gray-scale contours; Line, Arc, Lobe feature extraction, using database | indoor | ~70 | Object Recognition | - | Personal dataset | - | [47] |
| 20 | Ramadevi et al. | Different edge detection techniques for image segmentation | indoor | - | Object Recognition | - | Personal dataset | - | [49] |
| 21 | Dheemanth et al. | Eigen values and Eigen vectors, PCA | indoor/ outdoor | - | Object Recognition | - | Personal dataset | - | [52] |

# Chapter 3 Proposed Method

As per the literature review, [22], [30], [43] and [46] are amongst those papers which present reasonable solutions. However, there is scope for further improvement, particularly in the area of context-aware object recognition, which was found to be missing in the literature. Hence the proposed algorithm design looks at improving detection of moving objects when the background is also moving. This is done by introducing a revised version of optical flow and clustering algorithm, as well as introducing a new method for recognizing the detected objects with the help of heuristics and colour measurements. The proposed method is expected to be more effective and less time consuming. It could also improve accuracy and efficiency if applied to scenarios worked on by other researchers. The following improvements are realized and developed for this research:

- **Auto-Histogram Equalization**: Feature detection algorithms are brightness invariant. It becomes challenging when the image is dark or too bright. The work in this research incorporates adaptive image contrast which increases or decreases based on automatic histogram equalization.
- **Elimination of outliers**: A new outlier rejection system is introduced to reject feature points by considering the pixel shift due to camera movement. This is done to eliminate points that pertain to the background to or objects that are stationary to end up with motion vectors for moving objects in the foreground. The result of this would then be used by the clustering algorithm.
- **Clustering System**: K-means algorithm does not give the user the ability to specify the search radius. This poses a problem as the detected feature points may not always be very close to each other but still belong together in one cluster. An improved clustering algorithm has been developed with a search radius adaptable to the amount of points and the distance between them.
- **Context aware object recognition**: Many papers have used large databases of predefined or stored images of objects and people. This will require searching through the whole database to find a single match for the object under processing for recognition. This takes up to a few seconds, which is a significant amount of time. The method used in this research looks at parts of the scene to identify the class of moving objects, by incorporating colour measurements and heuristics to recognize what the

moving object might be, in context to the environment it is in. This approach adds perceptive object identification to this research.

In Figure 3-1, the flowchart shows an improved algorithm for object detection and a new algorithm for object recognition. The diagram depicts an enhanced algorithm for automatic histogram equalization of images and frames, along with an improved

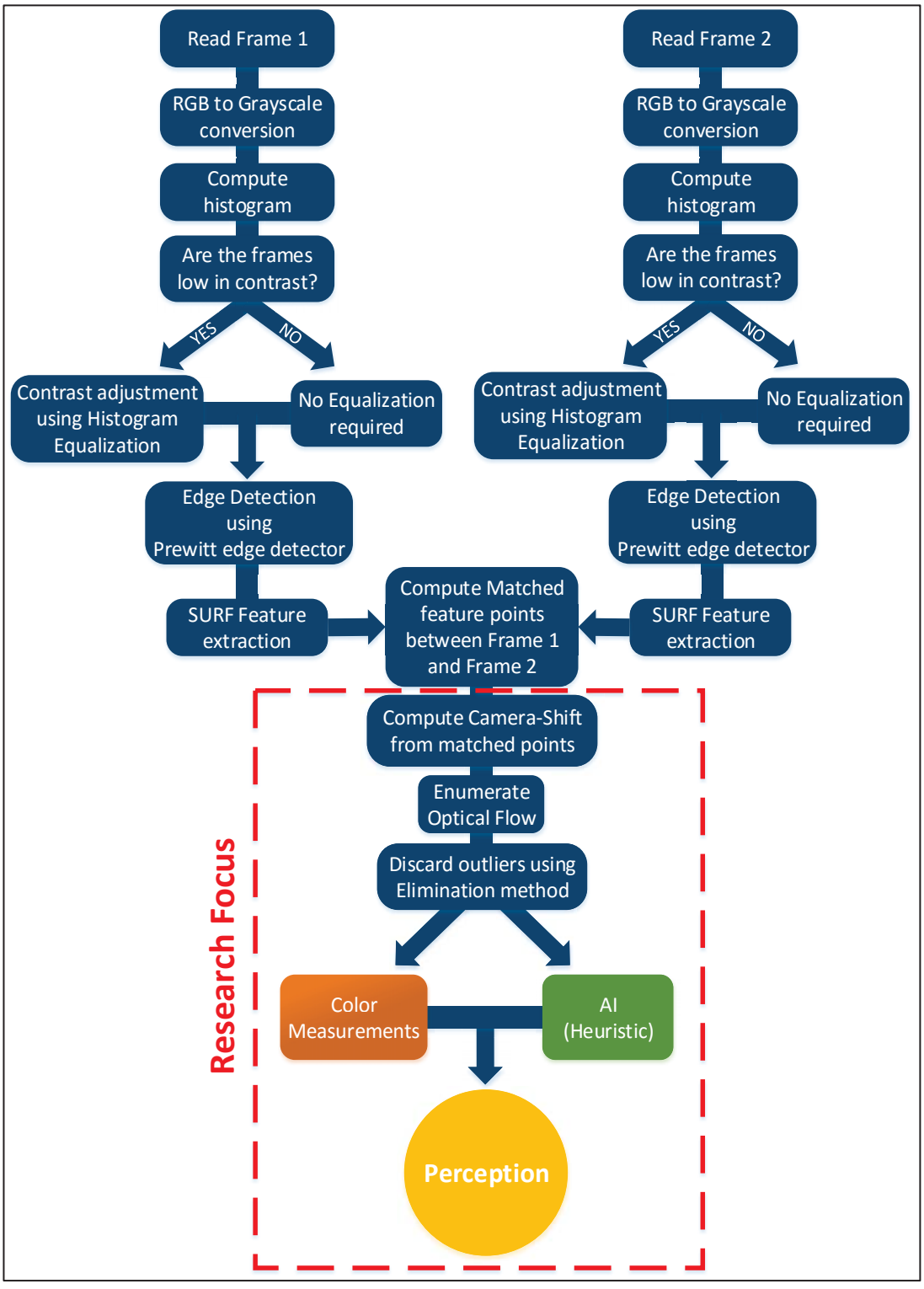



Figure 3-1: Algorithm Design

method for clustering of points, compared to the popular K-means clustering algorithm.

This chapter breaks down the approach into small segments and explains, in detail, the purpose and working of each segment with snippets of Matlab code. The algorithm is divided into a total of nine sections, which follows the overview shown in Figure 3-1. The algorithm was simulated in a technical computing environment to demonstrate the working of the algorithm and test its performance against the work done by other researchers in the same realm of moving object detection and recognition. The algorithm is aimed at validating the improvements that can be made in the areas of object detection and propose a new method for object recognition.

There are a few computing environments that support simulations for image processing such as Scilab, Octave and Matlab. Scilab is an open source platform which can be used for signal processing, image enhancement and modelling. It provides a good environment to carry out small mathematical calculations, but lacks the important toolboxes that were needed to carry out simulations for image processing. Octave, on the other hand, uses exactly the same syntax as Matlab, however, being a freeware and not having sufficient support with libraries written for Octave makes it a less preferred platform. Although Octave supports Matlab scrips, it lacks the built-in functions that are needed for Image Processing. Furthermore, Matlab provides an extensive toolbox for Image Processing, which is capable of image enhancement, image analysis such as identifying colours, shapes, counting objects, etc., image segmentation and much more. Matlab provides help and support for all their built-in functions. Besides being the most commonly used environment for simulation, it is used exclusively by computer vision researchers. In addition, past experience with Matlab, makes it the preferred platform to carry out the simulation and perform tests under different scenarios.

The experiments were carried out on Matlab running on a computer with Windows 7, 2.5 GHz CPU with 4 GB of RAM.

## 3.1 Auto Image contrast

Enhancing image contrast is a useful technique that allows better processing of the image. Test results have shown that images or frames, with low pixel intensity levels detect less feature points compared to a bright image. This means that important information could be concealed behind a silhouette of low contrast. Not every image taken will be bright and this could result in information being lost when compared to its brighter version. Therefore, there must be a method to detect the overall contrast of the image and adjust the contrast without any human intervention. This would be a part of the pre-processing stage.

An automatic equalization of pixel intensity level has been implemented in this work. There are 256-pixel intensity levels with 0 being the darkest pixel intensity and 255 being the brightest. Taking into account a threshold intensity value of 70. Dark pixels are associated between 0 to 70 and bright are associated between 71 to 255. If the mean intensity level for a dark-to-bright ratio is greater than 4, then histogram equalization with value of 70 is applied to the image using the Matlab function *histeq()*. This function seeks to flatten the image histogram to increase the specified pixel intensity level using a probability distribution function. Figure 3-2 shows the original image on the tip left and the same image processed with histogram equalization of 70, top right image, using the SURF feature detector for this example. It can be seen that SURF gives a wider depiction of the scene for the brighter image than its darker counterpart. SURF feature points were able to detect more points in the processed image than in the original image.

For the purpose of testing the automatic histogram equalization, one of the test images from Matlab's database has been put under speculation. A file named '*office_1.jpg*', shown in Figure 3-2 labelled as Image (a), was processed through the Matlab code shown in Code snippet 3-1. The built-in Matlab function, *histeq()* intends to average out the pixel intensity level. In other words, it tries to flatten the histogram in an effort to increase the contrast of the image. As seen from the set of images in Figure 3-2, the Histogram (a) pertains to dark Image (a) which has most of its pixels below 50. A value of 70 in *histeq()* brightens all the pixels with values of 70 and below to arrive at an image with Histogram (b). The test shows that Image (b) detects 1596 feature points, which has far more than Image (a), which detects only 59 points.

```
im1 = rgb2gray(imread('office_1.jpg')); % read test image
[COUNTS,pix] = imhist(im1); % compute histogram count for each
pixel

% compute average of pixel intensities between 0 and 70
thres70 = mean(COUNTS(1:71));

% compute average of pixel intensities between 71 and 255
thres71 = mean(COUNTS(72:256));

% compute ratio between dark and bright intensity thresholds
ratio = thres70 / thres71;

if ratio > 4
    % compute histogram equalization on im1 and store it as im2
    im2 = histeq(im1,70);
end

% detect SURF Feature points for both the original image im1 and
the histogram equalized image im2
points1 = detectSURFFeatures(im1);
points2 = detectSURFFeatures(im2);

% Display the two images and plot their SURF feature points
figure,
subplot(1,2,1), imshow(im1), impixelinfo
hold on
plot(points1.Location(:,1), points1.Location(:,2), 'b.');
subplot(1,2,2), imshow(im2), impixelinfo
hold on
plot(points2.Location(:,1), points2.Location(:,2), 'b.');
```

Code snippet 3-1: Matlab code showing automatic equalization for pixel intensity

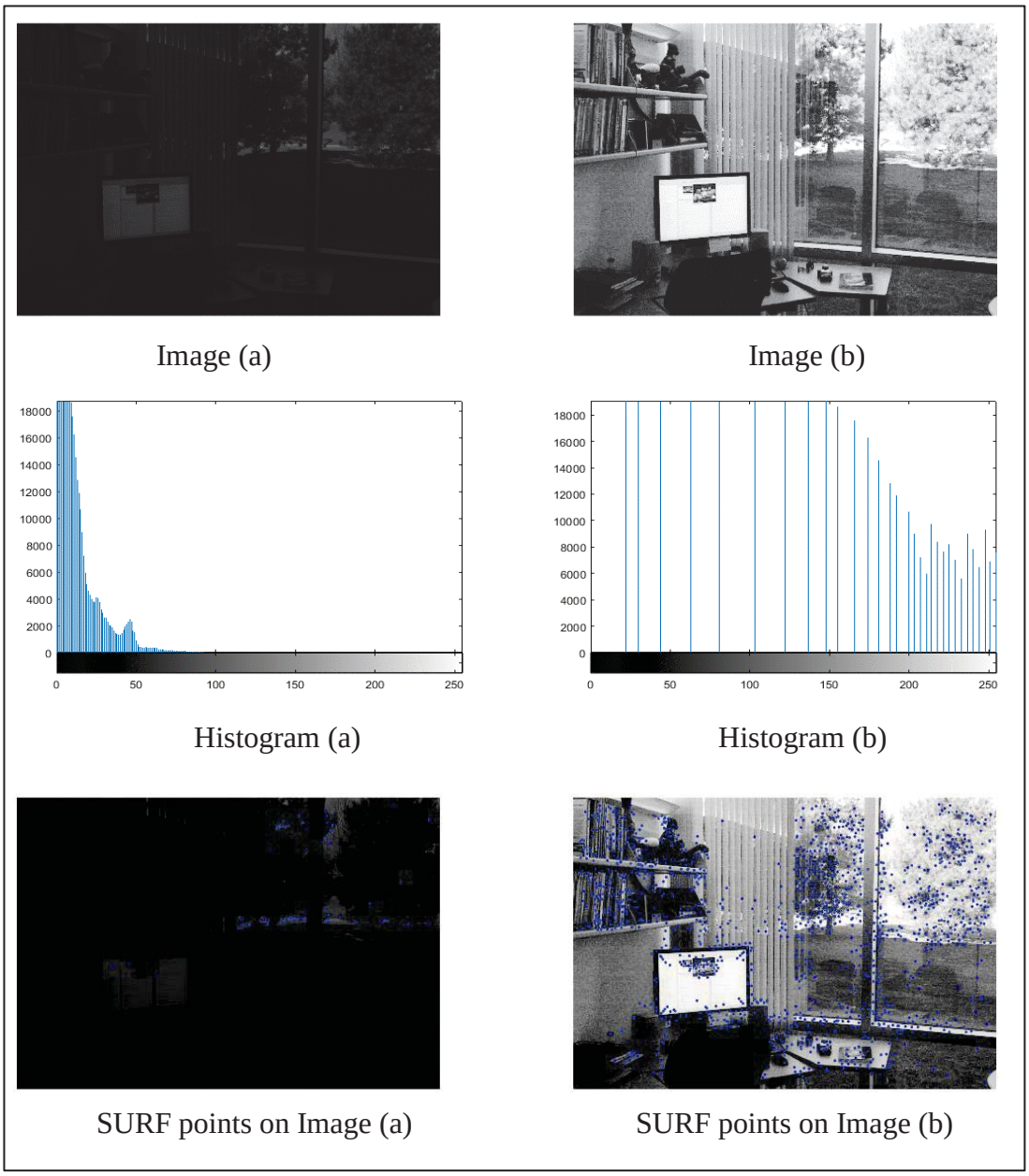



Figure 3-2: Shows original image (a) and its histogram equalized counterpart image (b). Plotted histogram (a) and histogram (b) for images (a) and (b) respectively. Plotted SURF feature points for the images (a) and (b).

## 3.2 Evaluation of Edge detection algorithms

Edge detection is used to find edges in the images. This is achieved by detecting discontinuities in brightness levels. Sharp changes in the image brightness usually occur at the borders or boundaries of objects in the image. Sometimes the points detected on an image are not sufficient and the user has to settle with other options such as using two feature detection algorithms.

To detect moving objects in the scene where the background is also moving, the video frames do not turn up sharp like stationary images. This makes the sharp transitions in pixel intensity levels of objects in the image quite smooth, thereby making it hard for feature detection algorithms to correctly detect points of interest. Feature detection algorithms are the foundation pillars of image processing which provide unique features in images. But if the points do not detect the keypoints of the object that relate to the objects of interest, it would become very challenging to set apart moving objects from the moving background. Edge detection algorithms could solve the problem of insufficient keypoint detection. There are different edge detection methods such as Sobel, Prewitt, etc. These methods were put to the test to compare the amount of feature points detected with and without edge detection.

Table 3-1: Number of feature points detected using different edge detection methods

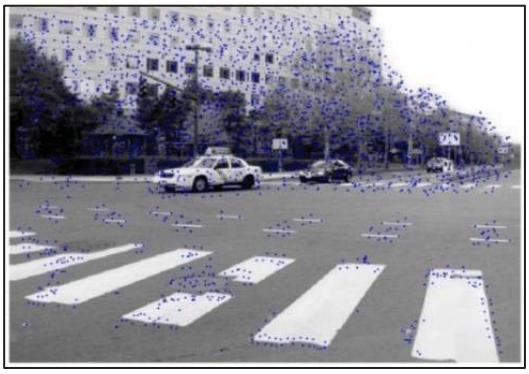
| **Frame name** | **Before edge detection** | **Edge detection method** | | | |
|---|---|---|---|---|---|
| | | Sobel | Canny | Prewitt | Zero-crossing |
| Shooting | 306 | 881 | 2244 | 841 | 1329 |
| Car | 467 | 1206 | 2981 | 1176 | 2446 |

Detection of SURF feature points using various edge detection algorithms that were evaluated for frames extracted from two video sets named *Car.avi* and *Shooting.mp4* are shown in Figure 3-3 and Figure 3-4 respectively. Detection techniques such as Sobel, Canny, Prewitt and Zero-crossing edge were used for this evaluation as these are the most widely used techniques. The edge detection algorithm under test was first used to compute the edges in the frame. The edge detection frame was then used to compute SURF feature points. More feature points are detected with the use of any edge detectors than without. This is a helpful aspect as there would be more feature points for the objects of interest in the frame, which would be helpful during the object recognition stage later on. Test results displayed in Table 3-1 show that Canny and

Zero-crossing edge detection algorithms detect feature points in abundance which include many superfluous points that relate to the background. On the other hand, Sobel and Prewitt detect points that are very similar to points detected before edge detection is applied but covers greater detail of the objects. From the performance perspective, the computation time for all the detectors was about 0.4 seconds on average. Canny and zero-crossing detect points in abundance, reaching over a thousand points for the car frame but Sobel and Prewitt detect sufficient amount of points with clear distinction between the objects of interest and the background. Therefore, Prewitt edge detector was chosen because of its performance of detecting the optimum amount of points.

SURF Features before edge detection

SURF Features after Sobel



SURF Features after Canny

SURF Features after Zero crossing

SURF Features after Prewitt

Figure 3-3: SURF Feature points plotted without edge detection (Top) and feature points detected using each edge detector for Car frame (middle and bottom rows)

The Matlab Code snippet 3-2 detects edges using Prewitt followed by detecting SURF Features which are then plotted on to the image.

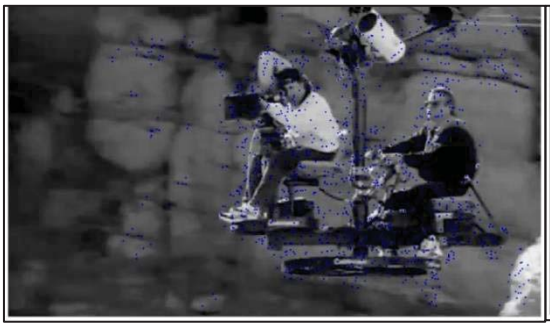
```
im1 = rgb2gray(im1_orig); % convert image to grayscale
points1 = detectSURFFeatures(im1); % detect SURF feature points
imshow(im1), hold on % show the image
title('SURF features before edge detection')
plot(points1.Location(:,1), points1.Location(:,2), 'b.'); % plot
points

im2 = edge(im1, 'Prewitt'); % detect the edge using Prewitt E.D.
points2 = detectSURFFeatures(im2); % detect SURF feature points
figure, imshow(im1), hold on
title('SURF features after Prewitt edge detection')
```

Code snippet 3-2: Extracts SURF feature points and detects edges using Prewitt

SURF Features before edge detection

SURF Features after Sobel

SURF Features after Canny



SURF Features after Zero Crossing

SURF Features after Prewitt

Figure 3-4: SURF Feature points plotted without edge detection (Top) and feature points detected using each edge detector for Shooting frame

## 3.3 Evaluation of Feature detection techniques and K-means clustering algorithm

Feature detection takes place within the image and recognizes unique information about the image. This unique information acts like the signatures of the image to help detect meaningful features also known as feature descriptors [23]. A wide variety of feature detection algorithms have been described in the literature as being able to compute reliable descriptors. These include algorithms such as FAST (Features from Accelerated Segment Test), MSER (Maximally Stable Extremal Regions), Harris corner detector, SIFT (Scale-Invariant Feature Transform) and SURF (Speeded-Up Robust Features), to name a few [59, 60]. SIFT, SURF and Harris corner detectors seem to be the most promising due to their ability of producing good results.

A comprehensive evaluation of many feature descriptors was described in [61] which concluded that, in general, SIFT outperforms other detectors in terms of the amount of detected points. However, SURF was not included in the comparison. Although SURF is a speeded-up variant of SIFT, which was put to test as part of this research. This section outlines a comparison between Harris, SURF and SIFT detectors and tests the algorithms on typical transformations such as blur and brightness variance.

The first pair of frames of size 120x160 pixels are extracted from the commonly used '*viptraffic*' video sequence which captured moving traffic from a stationary camera. Test images shown in Figure 3-5 have a fixed background with objects (cars) moving in the foreground and these are the objects of interest.

Figure 3-5: Two grayscale frames named Test frame 1 and Test frame 2. The camera is stationary with only the objects (cars) moving

The two frames shown in Figure 3-5 are used to detect feature points using each of the Harris, SIFT and SURF feature detection algorithms.

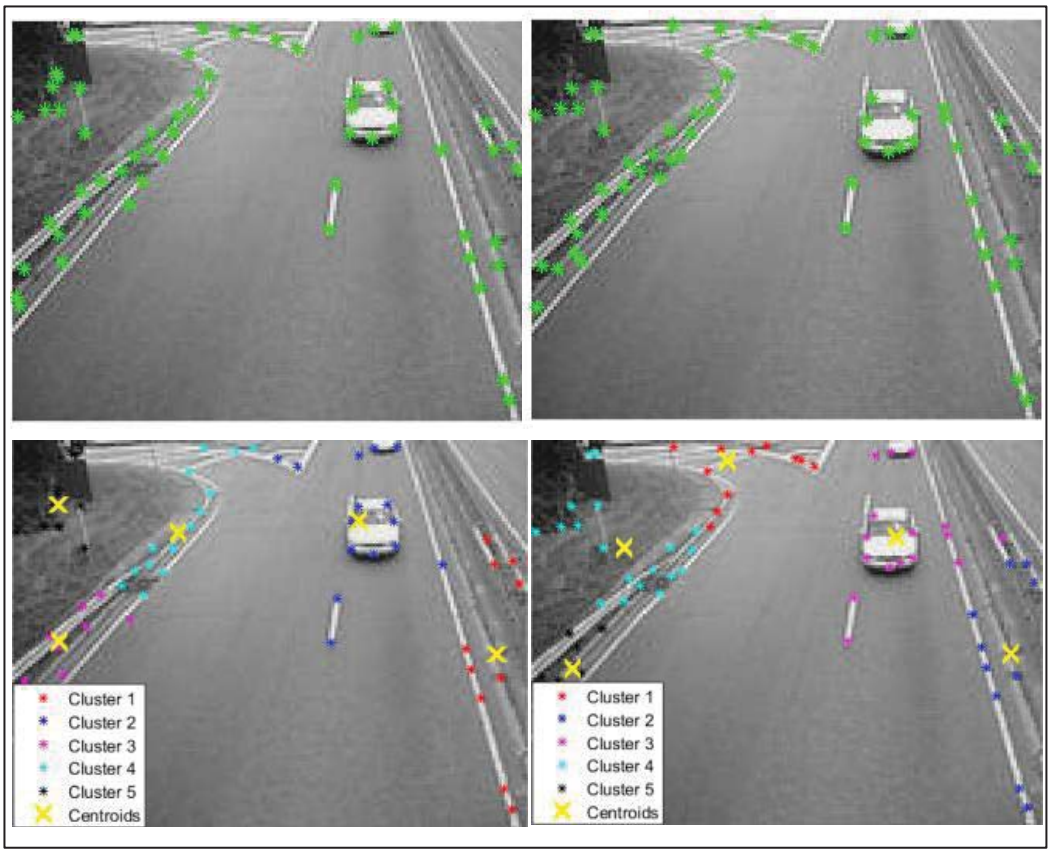



Figure 3-6: Corners detected using Harris corner detector (top row) with clustering through k-means (bottom row)

The top two frames shown in Figure 3-6 display an overlay of points in green detected using the Harris corner detector. The bottom set of frames are processed through k-means algorithm which divides the corner points shown above, into five groups called clusters, that are labelled as cluster 1, 2, 3, 4 and 5. The algorithm outputs the location of a centroid for each cluster which is marked in yellow.


Figure 3-7: SURF feature points plotted (top row) with clustering through K-means (bottom row)

Figure 3-7 depicts SURF feature detection points, which did not detect as many points as Harris corner detector. This is because the SURF algorithm looks for significant features which works best in high resolution images. However, the detector performed well in detecting features around the object of interest, the car. The bottom row displays the centroids of the three clusters as a result of running them through k-means algorithm. The centroids in this case have worked best in utilizing the points around the car and represent the car with just one point which would make computation much easier later.

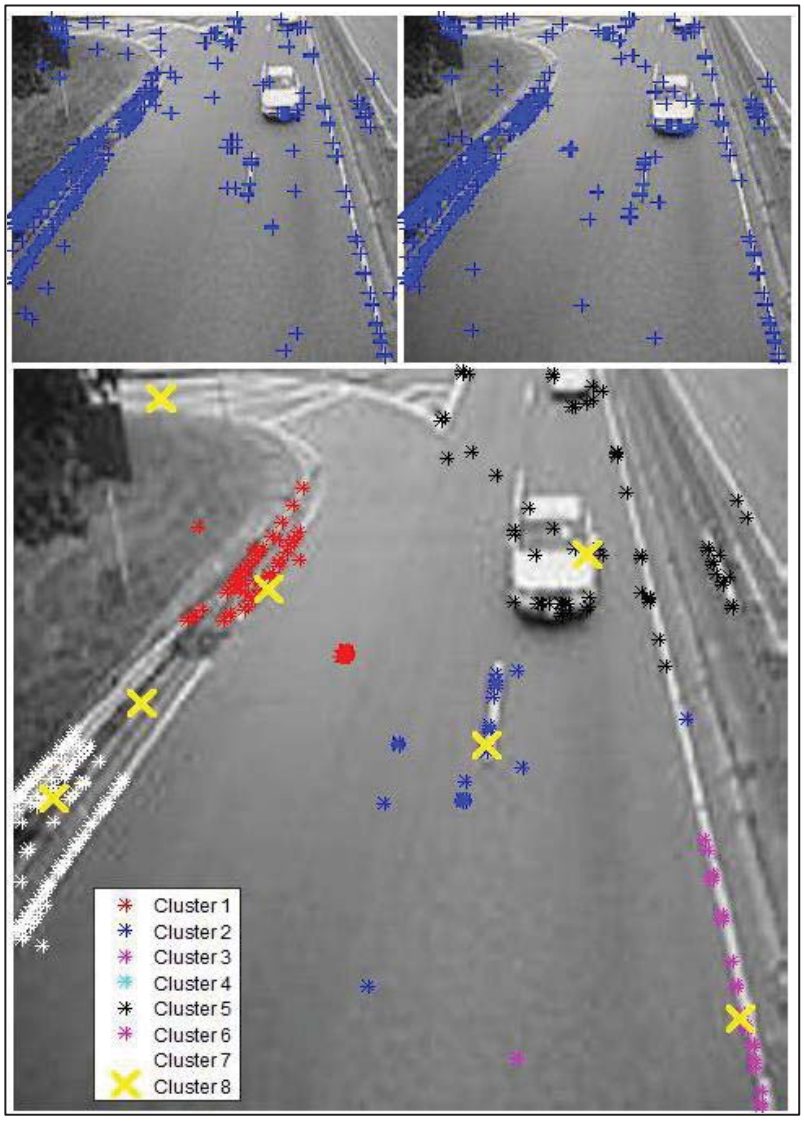



Figure 3-8: Features detected using SIFT feature detector (top row) and clustering through k-means algorithm (bottom)

It can be seen from Figure 3-8 that feature points detected by SIFT algorithm are thorough in detecting minute feature points in the image. It can be seen that SIFT detects a lot more points than Harris corner detector or SURF even though SURF is a speeded-up variant of SIFT. Running these SIFT points through k-means algorithm

results in sorting out data points into clusters and centroids. Due to a large number of data points, processing time for k-means algorithm to divide the points into clusters and compute their centroids is quite high ranging from 1 second to about 2.5 seconds depending on the number of clusters required for analysis. The number of clusters required is to be determined by the user while processing. SIFT is more susceptible to changes in pixel intensity and noise, which is why it can be seen from the images above, that some points are detected on the road, even though the road is a fairly smooth part of the image compared to its surroundings. Whereas, SIFT feature detector and Harris corner detector identified points where there are drastic changes in pixel intensity.

Table 3-2 shows the results of running the simulation. The execution times does not include displaying the image with plotted points which would take a bit of time to execute and display as an overlay over the frames. The results solely depict the run times of the algorithm. It can be seen that both the Harris corner detector and SURF feature detection algorithm have very fast computation times. When looking at the numbers, the computation times may seem to be very quick but execution time would be significant if high resolution images were to be processed.

Table 3-2: Computational time for each detector

| **Detector** | **Execution time (Sec)** |
|---|---|
| Harris Corner detector | 0.14 |
| SURF Feature detector | 0.18 |
| SIFT Feature detector | 0.31 |

Some high-resolution images of size 720x1280 pixels with non-stationary background were also tested with these three algorithms to detect moving objects. Figure 3-9 shows the two images that were used for this test where the cyclist acts as the moving object.

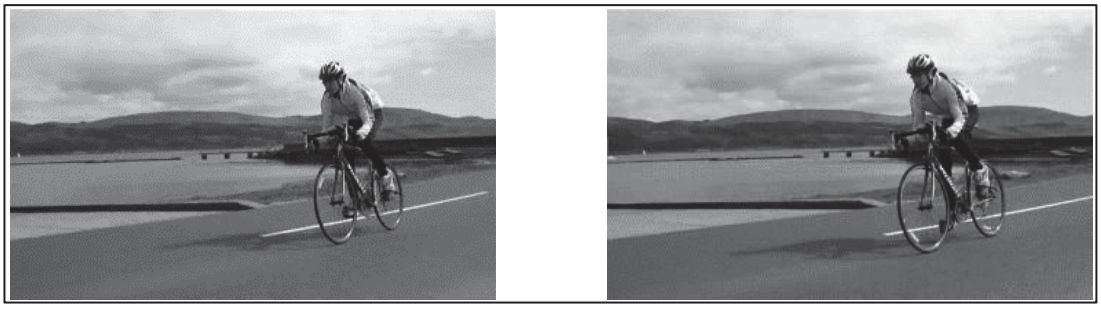


Figure 3-9: Second set of test frames

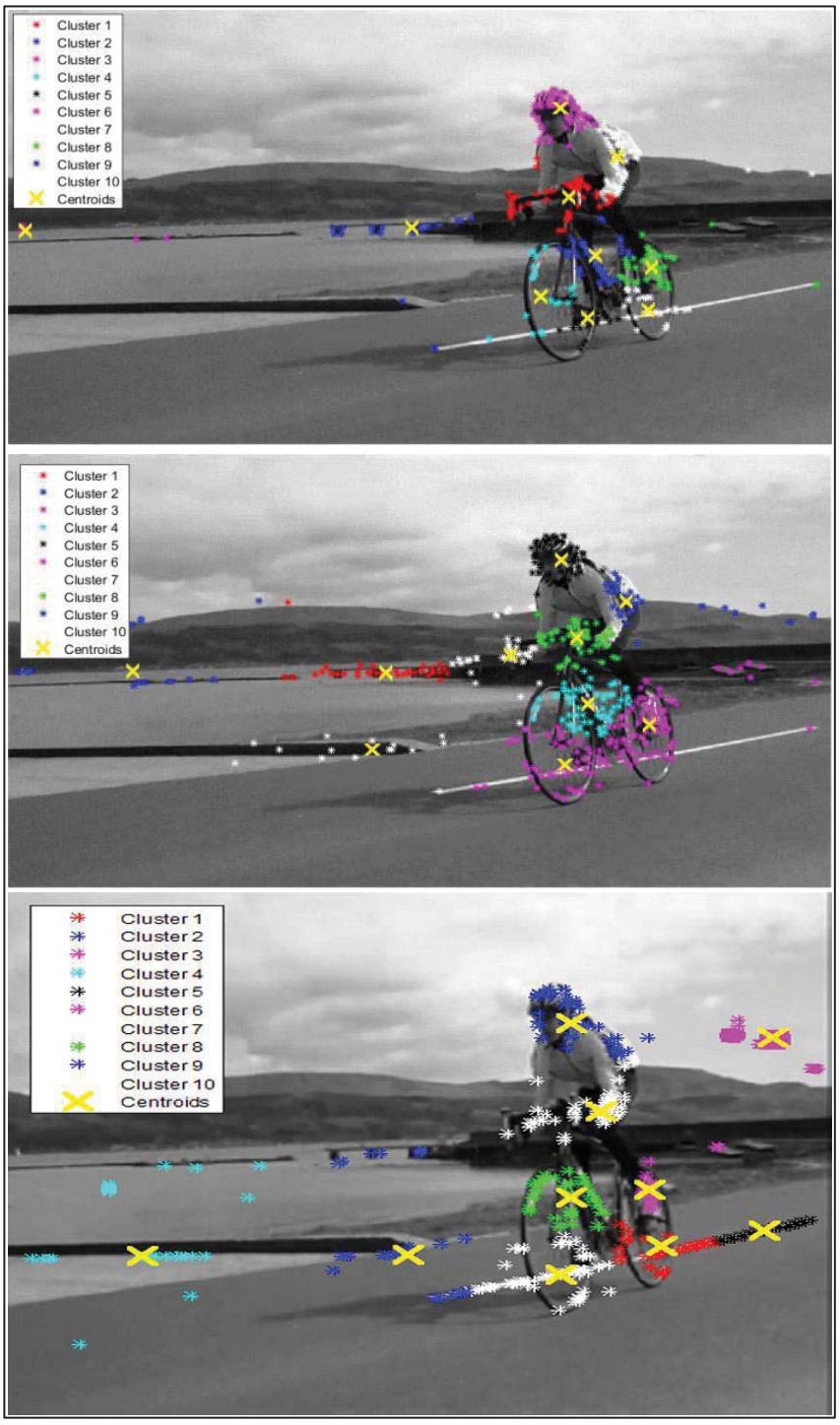



Figure 3-10: Harris, SURF and SIFT feature points with colour coded clusters and their centroids shown from top, middle and bottom respectively

As can be seen from Figure 3-9, the frame is sharp and consists of less noise compared to frames that were put to test earlier in this section. This is due to a significant amount of points detected by all the three detection algorithms. For the Harris corner detector shown in the top frame of Figure 3-10, it can be seen that no points are detected on the

smooth or plain parts of the image such as the road or water in the background. Harris detector has worked great with this high-resolution frame which is neither blurry nor noisy.

By analysing Figure 3-10, it is seen that all three algorithms perform somewhat the same when sharp high resolution images are used. The processing time to detect feature points for each algorithm increased, ranging between 0.5 seconds to about 1.2 seconds. The increase was due to the larger size of the images but it was not much different compared to the time taken for processing images through feature detection algorithms used in Table 3-2.

Feature point detection on blurry images is also considered. Images from Figure 3-11 show detected points of each of the feature detection algorithm discussed in this section. Blur on images from Figure 3-11 was created on Matlab using the *imfilter()* function to depict a camera motion blur.

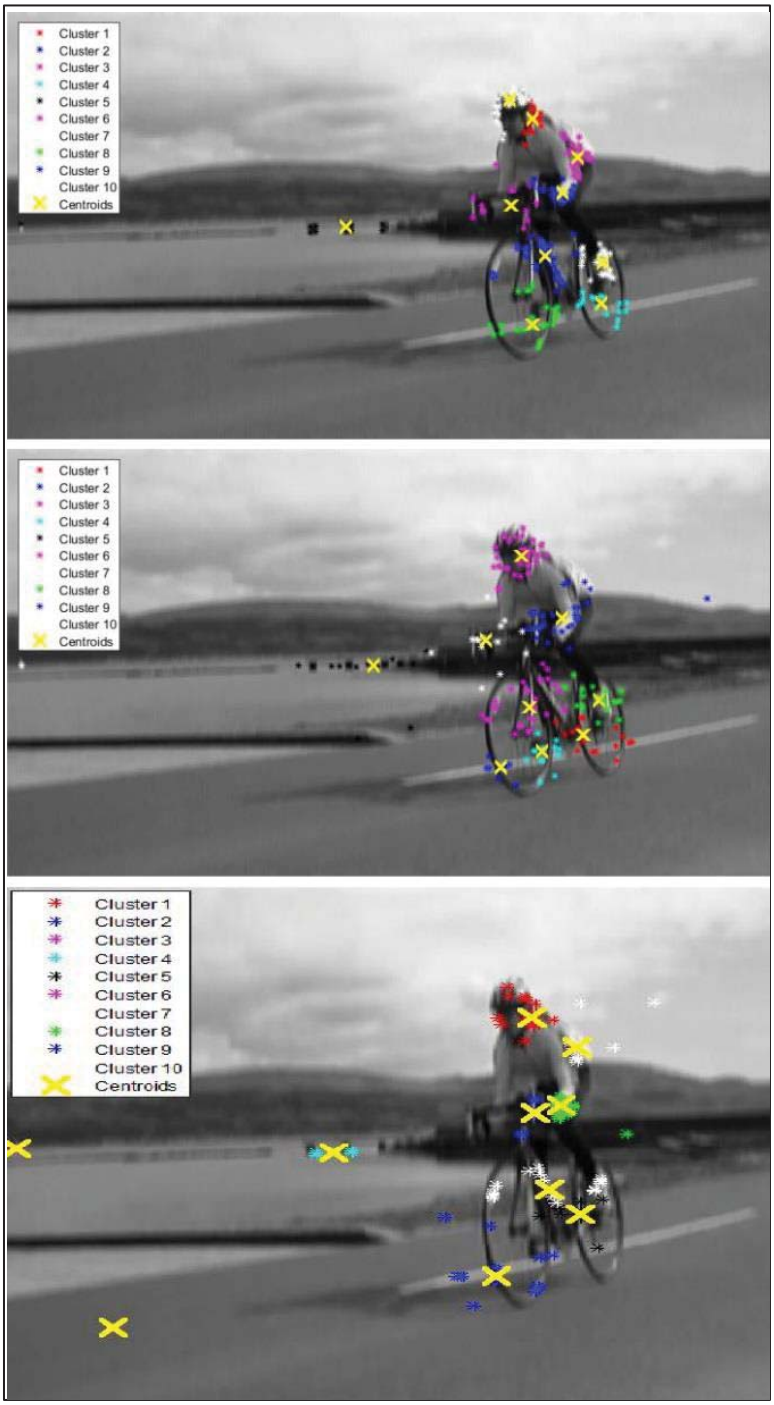



Figure 3-11: Harris (top), SURF (middle) and SIFT (bottom) features detected for blurry frames

Analysing the three frames shown in Figure 3-11, it can be seen that the feature detectors still work well with blurry frames, detecting enough points of the moving object in the foreground. Again, all three algorithms perform very well. It is obvious that the points detected in the blurry frames are comparatively less than the points detected in the original frame. SURF was able to detect more points in the background compared to Harris and SIFT.

The analysis of Harris, SURF and SIFT feature detection algorithms discussed in this section were evaluated on efficiency, performance, time of execution and the overall robustness. It was noticed that the algorithms are susceptible to noise which causes incorrect detection of points. All three algorithms performed well with both low and high resolution frames that were used for the simulation.

Simulations show that Harris corner detector processes the data much faster as opposed to SIFT. Although Harris and SURF process information relatively quickly and both have approximately the same computation time. However, SURF seemed to be more susceptible to noise and smooth pixel intensity levels.

SIFT algorithm performs well, detecting maximum amount of feature points, but at the cost of processing time. Referring to Table 3-2, time taken for SIFT to detect points was found to be almost double of Harris and SURF processing times. SIFT algorithm is complex and slow but its variant SURF, successfully upholds its name of being a speeded-up version of SIFT with processing times of almost half of that of SIFT's. Looking at Figure 3-10, SURF detects a considerable amount of points of the moving object as well as the background, which would ease the computation of differentiating between the background and foreground. Whereas, Harris detector does not detect enough points of the background. Taking both performance and time of computation into consideration, SURF comes out on top with a decent amount of points detected under 0.2 seconds for a frame of size 120x160 pixels. These feature points can be very useful in separating the moving foreground objects from non-stationary objects, or scene, in the background. And with the help of k-means algorithm of clustering, these points can be very useful in distinguishing between moving objects and the background. Which is why SURF feature detection algorithm was chosen for this research work to detect keypoints in the frames.

## 3.4 Computing Camera-Shift

Camera-shift occurs due to the motion of the camera which introduces the concept of moving background. This presents two types of motion in video frames which are, motion of pixels that pertain to the background due to camera movement, and the motion of pixels that occur due to the moving objects themselves. Computing this shift becomes a useful tool when trying to separate the moving background from the moving objects of interest in the foreground.

The assumption made is that the frames are free of any tilt or rotation. The most common method to compute the shift is to compute the motion field which is vector at each point in 3D space that is projected onto a 2D plane. Motion fields along with optical flow computation have been very commonly used in the past. Lee et al. [62] and Rabe et al. [63] used motion fields to explicitly separate the moving background from the motion of objects in conjunction with optical flow. Computing pixel wise motion field, which is in turn used to compute the optical flow between two frames. The individual motion vectors express the velocity, direction and angle of the motion of pixels in the image that provide information about the background as well as the objects of interest in the foreground. These motion vectors would include outliers as well as mismatched points between two frames. Experiments carried out for this research show that computation became a little faster and easier when matched feature points between two frames were used to calculate the camera-shift. This is because matched points gave good results that were common to both frames. These feature points might include points from both background and foreground and so the camera-shift might not be a 100% accurate, but it provides a good estimation about the shift of pixels upon which calculations for the average shift of pixels, angle and direction due to camera motion, can take place. These values were later to assist eliminating outliers after optical flow computation.

It can be assumed that the object of interest would have vibrant or sharp changes in pixel intensity levels at the edges, hence there could be points bunched up around objects which could relate to the object moving in the background. Clustering algorithm along with optical flow calculations can then be used to compute the movement of these objects with respect to the moving background. Objects of interest would obviously have a different velocity, angle or direction at which the objects might be moving.

Camerashift parameters are used to recognize the outliers and eliminate them. To figure out if the points pertain to the background or the foreground, optical flow is computed and the motion vectors are then compared against the cameraShift which is calculated using  Code snippet 3-3:

```
% Convert the original RGB frames to grayscale
im1 = rgb2gray(im1_orig);
im2 = rgb2gray(im2_orig);

% Detect and extract features from both images
points1 = detectSURFFeatures(im1);
points2 = detectSURFFeatures(im2);
[features_im1, validPts_im1] = extractFeatures(im1, points1);
[features_im2, validPts_im2] = extractFeatures(im2, points2);

% Match feature vectors
indexPairs = matchFeatures(features_im1, features_im2);
matchedPoints1 = validPts_im1(indexPairs(:,1));
matchedPoints2 = validPts_im2(indexPairs(:,2));

% Exclude the outliers and compute the transformation matrix
[tform,inlierPtsOut,inlierPtsIn] = estimateGeometricTransform(...
     matchedPoints1, matchedPoints2, 'similarity');

% Extract x,y coordinates from matchedPoints and round them up
matchedPoints1 = round(matchedPoints1.Location);
matchedPoints2 = round(matchedPoints2.Location);

% Extract the total number of matched feature points
amountOfPoints = round(length(indexPairs)/2);

% Initialize variables to store the camera shift and angle
avg = zeros(amountOfPoints,1);
ang = zeros(amountOfPoints,1);

% calculate shift due to camera movement
for i = 1:length(matchedPoints1)
    x = ((matchedPoints1(i,1))-(matchedPoints2(i,1)))^2;
    y = ((matchedPoints1(i,2))-(matchedPoints2(i,2)))^2;
    base = abs(matchedPoints1(i,1) - matchedPoints2(i,1));
    perp = abs(matchedPoints1(i,2) - matchedPoints2(i,2));
    ang(i) = round(atand(perp/base));
    avg(i) = round(sqrt(x+y));
end

cameraShift = mode(avg); % compute camera shift
cameraShift_ang = mode(ang); % compute angle
```

Code snippet 3-3: Matlab code to compute camera shift

Code snippet 3-3 reads two colour frames, converts them to grayscale using the *rgb2gray()* function and detects SURF feature points on both frames. Next, it extracts features from both images and then uses the *matchFeatures()* function to compute the preliminary features that are common to both frames. These features would contain outliers which are rejected using *estimateGeometricTransform()* function which runs the points through RANSAC to arrive at accurate feature points between the two frames. These matched points are then used to compute the camera shift and the angle by calculating average distances between two matched feature points. Running the Matlab code returns the statements shown in Figure 3-12 and generates an overlay of two frames with the matched points plotted, as shown in Figure 3-13.

```
      cameraShift:  11
cameraShift_angle:  56
    dir of motion:  left
```

Figure 3-12: Result of running Code snippet 3-3

In scenarios when the background is smooth, like the top set of frames in Figure 3-13, the matched points detected only on the object of interest because that is where most of the activity is, in regard to changes in pixel intensity levels and, in addition the majority of the image consists of smooth background. On the other hand, the bottom set of images involve vibrant intensity changes in the background. Due to camera motion and abundance of matched points all around the frame, it can be realized that these points do not pertain to the moving objects but relate to the background. Keeping this in mind, since matched feature points are checked if the points are bunched up close to each other and not spread out for the most part of the image, then it can be assumed that it would be an object or else, it is sent straight to the elimination round.

Figure 3-13: Two set of frames displaying matched feature points.

In Figure 3-13, matched points between a pair of frames are depicted by red circles which are points from Frame 1, and green crosses which denote matched points on Frame 2. The distance between these two points provide an approximation of pixel movement due to camera motion.

## 3.5 Optical Flow computation

Optical flow is the motion of pixels between two consecutive frames that occur due to camera motion or the motion of objects moving in front of the camera. As evaluated in the literature review, there are a few types of optical flow methods that researchers have used, such as block-matching based optical flow and pyramidal Lucas-Kanade optical flow which are a few very commonly used optical flow techniques.

Full Search block matching algorithm acquires a block of a certain size from the first image and searches through the second image by shifting the block along to search the entire image while calculating Sum of Absolute Difference (SAD). SAD measures similarity between two blocks and stores the x and y coordinates of the centroid of the matched block. Often, this method is also referred to as the Exhaustive search because each block from the previous frame is compared to each block in the next frame which is computationally intensive and time consuming. Pyramidal Lucas-Kanade method involves computing the pyramidal levels of the image and processing the motion vectors from the top level. The top level has the highest pyramidal reduction of the image where level zero is the original image, level one has half the height and width of the original image and level two is half the size of image in level one and so on. For testing purposes, pyramidal level of four is chosen with iteration count of three. To calculate the optical flow vector with the pyramid, total number of pyramids must first be defined. Starting with the highest pyramid level, find the partial derivatives of pixel intensity $I_x$, $I_y$ and $I_t$ and update the motion vector $v_x$ and $v_y$ using Lucas-Kanade method. Initially the values of motion vectors are zero. Optical flow is computed across all the levels, starting from the highest level and repeated down to the lowest until the number of iterations are all computed. Cross correlation works in a similar manner to full search block matching algorithm. A block of certain size is acquired from a previous image or frame which is used to find a match in the next frame using cross correlation. The patch is shifted one pixel at a time in every direction to search for the best match. This makes computation lengthy and time consuming.

All the three methods that have been stated above involve acquiring blocks from the first frame and using it to search through the whole of the second frame. For example, a set of frames of size 640x360 with a block size of five pixels comes to 46,080 blocks that will need to be searched one by one through the second image.

Table 3-3: Processing time taken for different optical flow methods

| Optical Flow Methods | Time taken (sec) | Description | Observations |
|---|---|---|---|
| Pyramidal LK method | 30.9 | 3 iterations with 4 pyramid levels | produces a lot of motion vectors for both moving background and moving objects |
| Full Search block matching | 101.6 | block size of 8 pixels using sum of absolute difference (SAD) | produces a lot of outliers especially on smooth surfaces |
| Full search cross correlation | 114.6 | block size of 8 pixels using cross correlation (xcorr) | produces a lot of outliers especially on smooth surfaces like FSBM |
| Feature point based cross correlation | 20.4 | block size of 7 pixels in each direction of the feature point | fast computation time, produces bare minimum outliers |

Results obtained from evaluation of different optical flow methods shown in Table 3-3, tests the performance of three most commonly used techniques, with respect to time taken for each method and the observations. The evaluation was carried out with test images of size 640x360 pixels. It can be seen that Full search block matching and Full search cross-correlation methods were computationally intensive, hence taking a long time to compute optical flow. Whereas, Pyramidal LK method takes a third of the time compared to full search block matching methods. Therefore, the method chosen to compute optical flow for this research is an improved version of the cross-correlation method explained above. The diagram shown in Figure 3-14, illustrates how motion vectors are computed using feature points from one frame, which is then searched through the whole of the second frame to find a match.

The results from the processing of this optical flow technique, in Table 3-3, shows that the execution time is the shortest, compared to other methods. Feature based cross-correlation has been adopted mainly because of its short computational time and also due to the fact that this method generates nominal outliers. Features depict unique keypoints in the images. These keypoints from frame 1 can be used to detect the same or similar points in the next frame by extracting a patch around the original feature point from frame 1 of certain size and using this patch to find a match in the next frame using cross correlation. This way the smooth parts of the image or the parts that do not have drastic changes in pixel intensity, could be thought of as parts that belong to the

background and can be excluded from optical flow computation. Another benefit of using feature point extraction for optical flow computation is that the feature points capture objects well, which becomes more efficient and becomes less reliant on detecting moving objects on just optical flow alone. Figure 3-14 is an example displaying how optical flow is computed using feature points detected for the previous frame (Frame 1), searched and matched using cross correlation in the next frame (Frame 2). After cross correlation, outliers get discarded using the elimination phase, which makes use of the camera-shift, angle and direction to reject incorrect matched points. The final result shows the optical flow vectors of the objects that are detected.

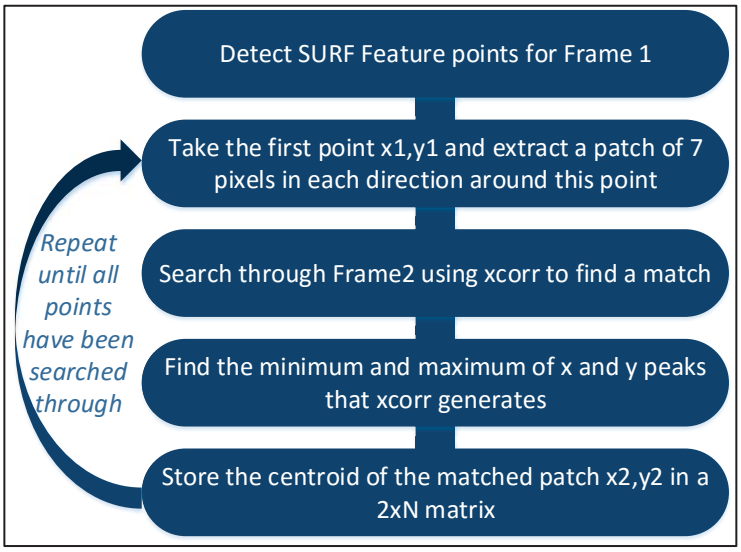



Figure 3-14: Optical flow diagram

Two frames named ‘Target_01.jpg’ and ‘Target_02.jpg’ in Figure 3-15 depict the matching process between two frames. As it can be seen from the two frames, there are a good amount of points detected for the object of interest, which is the person on a bike and also the tree in the background because there are variations in pixel intensity levels at these two parts of the frame. But the smooth regions which are the road, the mountain and even the sky have no points detected which saves a significant amount of computation time since the two frames do not need to find any matches for these areas. In the same way, out of most of the points that are detected around the object of interest, not many points can be considered outliers. Since the points are already close together, the clustering of points would not be as computationally expensive as it would be, for instance, to sort process all the motions vectors from a full search block matching optical flow algorithm.

```
pSize = 7;
bSize = 15;

% create padding around im1
B = padarray(im1,[bSize bSize]);
```

Code snippet 3-4: Padding the image for full search block matching with specified padding side

The Code snippet 3-4 creates a padding around the first frame from which patches around feature points are extracted if the points are close to the edge of the frame. The number of pixels that get added to the frame in every direction is *bSize*. The patch size, which is the number of pixels from the feature point under processing in every direction is *pSize*. The *padarray()* function is a Matlab function which creates zero padding around the image.

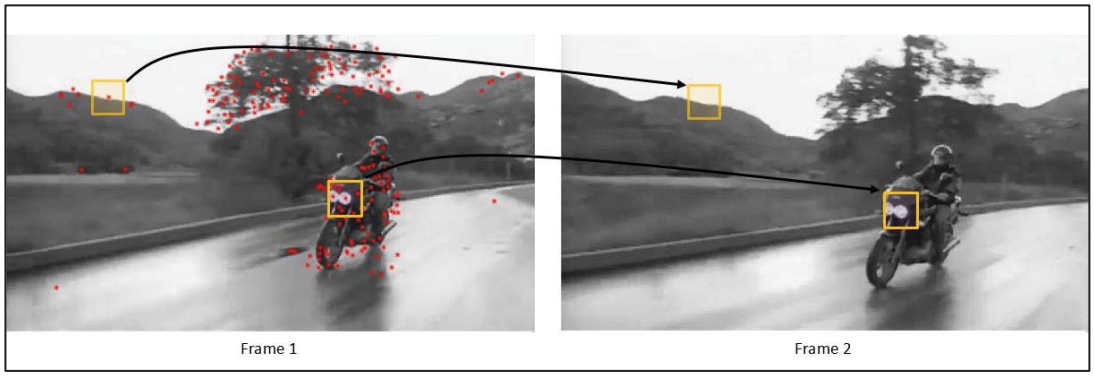



Figure 3-15: The matching process of feature points between two consecutive frames using cross correlation

The feature points that get detected by SURF features detection algorithm in Frame 1 are stored in an array named *points1* which include properties of points such as Scale, SignOfLaplacian, Location, etc.

```
points1 =

  456x1 SURFPoints array with properties:

              Scale: [456x1 single]
    SignOfLaplacian: [456x1 int8]
        Orientation: [456x1 single]
           Location: [456x2 single]
             Metric: [456x1 single]
              Count: 456
```

Figure 3-16: Result of detecting SURF features on images shown in Figure 3-15

```
% extract all the coordinates from SURFPoints properties
points1 = (points1.Location);
% round up all the location points to whole numbers
point_im1 = round([points1(i,1),points1(i,2)]);
```

Code snippet 3-5: Extraction of (x,y) location and rounding to whole numbers

The statements in Code snippet 3-5, extract all the centroid locations of all the points detected by SURF and stores them in points1 array. Each point that is used to extract a patch from image1 is acquired from points1, and rounded to the nearest whole number, as the x,y coordinates that are returned by SURF algorithm are not integers.

```
% extract a patch of size (pSize+1+pSize) from padded image
patch = B(point_im1(2)+bSize-pSize:point_im1(2)+bSize+pSize,...
          point_im1(1)+bSize-pSize:point_im1(1)+bSize+pSize);
```

Code snippet 3-6: Acquiring of a small patch of a certain size from the padded image B

Code snippet 3-6 extracts a block from the newly created image B which is a padded version of Frame 1. *bSize* gets added to point_im1 which compensates for the offset occurred due to padding and *pSize* is the number of pixels in each direction of the feature point at hand.

```
BB = abs(detected_points(i,1) - xy_vect(i,1));
PP = abs(detected_points(i,2) - xy_vect(i,2));
ang(i) = round(atand(PP/BB));
```

Code snippet 3-7: Computation of distance between two matched points in image 1 and image 2

BB and PP variables in Code snippet 3-7 are the Base and Perpendicular distances between the detected SURF points in image 1 and its corresponding matched point in image 2.

```
% search for a match between patch and im2
c = normxcorr2(patch,im2);

% find the x and y peak from c
[ypeak, xpeak] = find(c == max(c(:)));

% search again for max values if x and y peaks return more than 1
value
ypeak = max(ypeak);
xpeak = max(xpeak);

% compensate for the offset that normxcorr2 function introduces.
yoffSet = ypeak-size(patch,1)+1;
xoffSet = xpeak-size(patch,2)+1;

% find the centroid of the matched patch in im2 and save it in
points_im2
point_im2_x = (xoffSet*2 + size(patch,2)-1)/2;
```

Code snippet 3-8: Code to perform cross-correlation

Referring to Code snippet 3-8, *normxcorr2()* computes normalized two-dimensional cross-correlation. Obtained from Code snippet 3-6 for Frame 1 of size 15x15 pixels was the extraction of the *patch*, which searches through Frame 2 labelled as *im2* in the *normxcorr2()* function. When the *patch* finds a match in *im2*, a peak is created which signifies a match. Plotting a 3D coloured parametric surface using the *surf()* function of Matlab, Figure 3-17 is produced with not just one but a few peaks. These peaks indicate outliers. To eliminate these, *[ypeak,xpeak]* is searched through *c* to find the peak which results not just an integer, but a vector which is again used to find the maximum value.

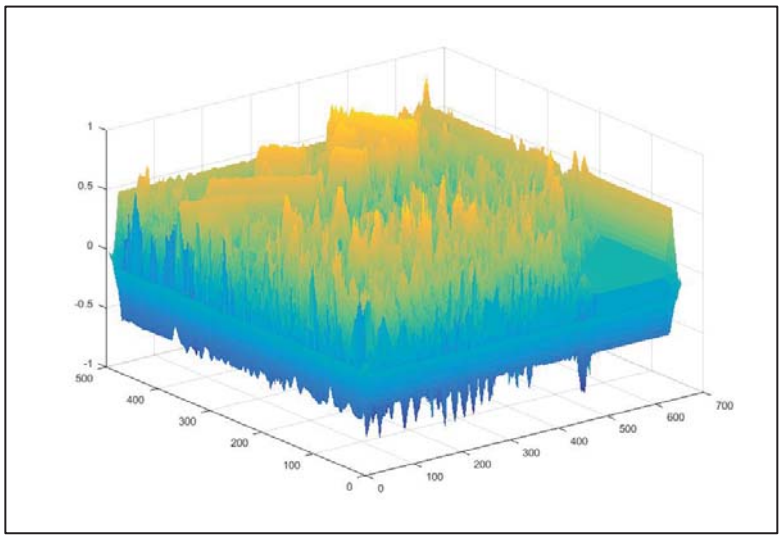



Figure 3-17: Surf plot of x,y peaks as a result of normalized cross-correlation

Code snippet 3-9 computes the distance between the centroid of the extracted patch from Frame 1 and the matched block of Frame 2. The distance between the two centroids are computed and stored in the *diff* vector.

```
% compute the distance between two centroids
diffx  = (point_im1(1)-point_im2_x)^2;
diffy  = (point_im1(2)-point_im2_y)^2;
diff(i)= round(sqrt(diffx+diffy));
```

Code snippet 3-9: Computes the distance between the detected points and the new points

```
% plot the centroid of the block from Frame 2
plot(point_im2_x,point_im2_y,'g+')
hold on

% plot vector arrow with each iteration
quiver(point_im2_x,point_im2_y, (point_im1(1)-point_im2_x),
(point_im1(2)-point_im2_y), 1, 'Colour', 'y');
```

Code snippet 3-10: Plotting the vector arrows using the quiver() function

Code snippet 3-10. This function takes four parameters (*quiver(X,Y,U,V)*). X,Y is the point from which to plot the vector arrow with components U,V. These points could either be integers to plot one arrow or a vector to plot several motion vectors. In Code snippet 3-10, X and Y are a centroid location of matched block from frame 2 and the distance between the two centroids are denoted by U and V plot arrows in yellow with scale of 1. The outcome of the optical flow computation is shown in Figure 3-18.

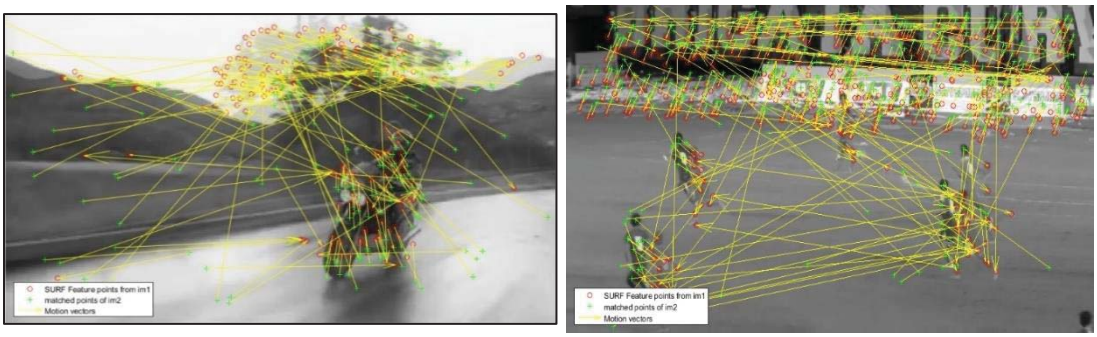


Figure 3-18: Optical flow between two pairs of frames named Target (left) and Soccer (right)

## 3.6 Elimination of outliers

As discussed in Section 3.5 computation of optical flow between two frames is used to compute matches between frames. All techniques of optical flow are based on approximation of finding the best match across the frame set up on various measures, such as Sum of Absolute Difference (SAD), Mean Squared Error (MSE), etc. These are all estimates that work well for the most part, but consist of some degree of inaccuracy. Susceptible to noise and smoothness, these estimates generate some matches that are incorrect and these mismatches are known as outliers.

The purpose of rejecting the outliers is to eliminate all the superfluous motion vectors and eventually be left with the vectors that pertain to the objects of interest, so that these points can be sent over to the clustering function for the vectors to be grouped into clusters. With the presence of outliers, the clustering of points would not be accurate, hence detection of moving object would be even more challenging.

Outliers inevitably occur and have to be eliminated, which in most cases influence the accuracy of results. There are quite a few interpretations to identify outliers, such as graphical techniques, which include plotting scatter plots or box plots to detect outliers when distribution is normal [46]. But this technique requires user intervention. RANSAC is a very commonly used algorithm which is used to discard outliers. It stands for Random Sampling Consensus which uses a polling scheme to find the optimal result. The built-in function in Matlab called *estimateGeometricTransform()* uses RANSAC algorithm to reject outliers which is how camera shift is computed in Section 3.4. Experiments carried out to reject outliers have shown that RANSAC sometimes perform false rejection of points, whereas points that relate to the background are the ones that need to be rejected, so that in the end, only the points that relate to the objects of interest remain. RANSAC focuses more on points that have prominent variations in pixel intensity levels, than points on the objects of interest. Results shown in Figure 3-19, show RANSAC does not provide the results that were expected, which is why a more manual approach is taken in this research. Results have shown that the new method to eliminate points works slightly faster than RANSAC. Experimental results have also shown that RANSAC has a high risk of eliminating useful points that it considers as outliers, hence high risk of false rejection. Referring back to Figure 3-18, the motion vectors shown as a result of optical flow computation for two pairs of frames are processed through RANSAC for outlier rejection. The

motion vectors are rejected using the Matlab function *estimateGeometricTransform()*. It is obvious that the object of interest is the person on a bike and the soccer players on the field for figures (a) and (b) in Figure 3-18 respectively. Therefore, the motion vectors that do not involve vectors for the bike and the soccer players must be rejected. But, instead, RANSAC performs incorrect rejection of points that belong to the objects of interest as seen from Figure 3-19. Thus, this function in Matlab, which works in conjunction with RANSAC, does not provide the desired result in such situations.

Figure 3-19: Showing pair of Target (top) and Soccer (bottom) frames depicting elimination of points using RANSAC

As part of this research, an elimination method is introduced to eliminate feature points whose optical flow vectors distinguish between the point being an inlier or an outlier. This would help narrow down and arrive at the motion vectors for only the moving objects in the foreground. Referring to Figure 3-18, optical flow computation using cross-correlation results in long length of outliers as a result correlation of smooth

surfaces between two frames. Therefore, elimination of these outliers become straightforward. As stated in section 3.4, it is assumed that the camera shift gives information about the global motion of pixels due to camera movement. The parameters of global motion are used to discard points that belong to the background. Keeping in mind that the elimination of points only takes place if the points are scattered across the image or at least cover a certain percentage of the image just as caution, to make sure that the points relate to both the background and the foreground. If this is not the case and if the points are bunched up at a certain part of the image or frame, then it can be assumed that the points belong to the object of interest and elimination phase can skip over these points.

Code snippet 3-11 shows how vector arrows are discarded by finding the most common length of vector arrows, which is assumed to belong to the background as shown in Figure 3-19. The variable named *diff* is of size Nx2 which comprises of the length between each feature point from frame 1 and its corresponding match in frame 2. The feature points that pertains to the background will have the same length of vectors arrows as well as the angle. Discarding these points would eliminate most of the irrelevant points from the background. The variable *discard*, stores the location of all the points in *diff* that are the same as camera shift. The variable named *points1* is of size Nx2 that consists of feature points that SURF algorithm detected in frame 1. Variable *points2* is the same size as *points1* containing the corresponding matched feature points from the optical flow computation between frame 1 and frame 2. Variable *diff* is the difference of corresponding points x and y coordinates between frame 1 and frame 2. Lastly, variable *ang*, a matrix of size Nx2 stores the angle of each vector. The for-loop discards all the points using the locations stored in the *discard* matrix.

```
com = mode(diff); % find the most common length of optical flow vector
discard = find(diff == com); % find the locations of these common
lengths of optical flow vector in diff

% for-loop to discard points that pertain to this size of optical flow
count = 0;
for i = 1:length(discard)
    points1(discard(i)-count,:) = [];
    points2(discard(i)-count,:) = [];
    diff(discard(i)-count) = [];
    ang(discard(i)-count) = [];
    count = count+1;
end
```

Code snippet 3-11: Elimination of vector arrows

Now, *diff* consists of points whose motion vectors are of the most common length. As optical flow is computed using a set of consecutive frames, the outliers can easily be spotted if the motion vector of an arbitrary point is significantly greater than the rest of the motion vectors, and spans across the frame, which usually happens when there are smooth parts in the frames. The Code snippet 3-12 shows another phase of eliminating points that have an abnormal length of motion vector, thus a value of five was chosen because the vectors must not be greater than five times the most common length of motion vector.

```
com = mode(diff);
discard = find(diff > abs(com*5)); % discard motion vectors that
are 5 times the size of com

count = 0;
for i = 1:length(discard)
    points1(discard(i)-count,:) = [];
    points2(discard(i)-count,:) = [];
    diff(discard(i)-count) = [];
    ang(discard(i)-count) = [];
    count = count+1;
end
```

Code snippet 3-12: Discard vector arrows that are 5 times the length of the common vector size.

Code snippet 3-13, carries out similar process to Code snippet 3-12. The only difference is the points that are to be eliminated. In section 3.4, an assumption that the camera is kept perpendicular to the ground and that the frames do not bear tilt or rotation of any sort, was made. With that respect, the angle of motion vectors would never be 0 or 90 degrees.

```
discard = find(ang == 0); % find location of all vectors with
angle of 0

% find location of all vectors with angle of 90 and add it to
discard
discard(size(discard,1)+1:size(discard,1)+length(find(ang ==
90))) = find(ang == 90);

count = 0;
for i = 1:length(discard)
    points1(discard(i)-count,:) = [];
    points2(discard(i)-count,:) = [];
    diff(discard(i)-count) = [];
    ang(discard(i)-count) = [];
    count = count+1;
end
```

Code snippet 3-13: Discard all error points that have a value of 0

After the elimination phase, some points that relate to the object of interest will remain and there will also be some points that will have the same motion vector as the objects of interest. The remaining outliers will be excluded when the remaining points are processed through the clustering phase.


Figure 3-20: Motion vectors after elimination of outliers for Soccer frame (top) and Target frame (bottom)

## 3.7 Clustering

The function prototype for clustering of points is shown in Code snippet 3-14:

```
function [flag_matrix,centroid_locations] =
clusterPoints(originalPoints,intraSearchRadius,centroid_disp_limit)
```

Code snippet 3-14: Function prototype for the clustering function

Code snippet 3-14 shows function prototype for the clustering function that accepts three inputs. First input is the *originalPoints* of size Nx2 containing the x,y-coordinates of matched feature points that need to be clustered along with two types of search ranges. Second is the *intraSearchRadius* specifying the distance around the vicinity of the current point under progress. Third is the distance threshold between two clusters defined by *centroid_disp_limit*. This threshold value is used to merge two clusters together, if the distance between them is less than the threshold. And this returns two set of points. Variable named *flag_matrix* is a vector that stores the cluster number for each corresponding x,y feature point in *originalPoints* belongs to. And *centroid_locations* is an Nx2 matrix which stores the location of each centroid. The process of the clustering system is shown in Figure 3-21.

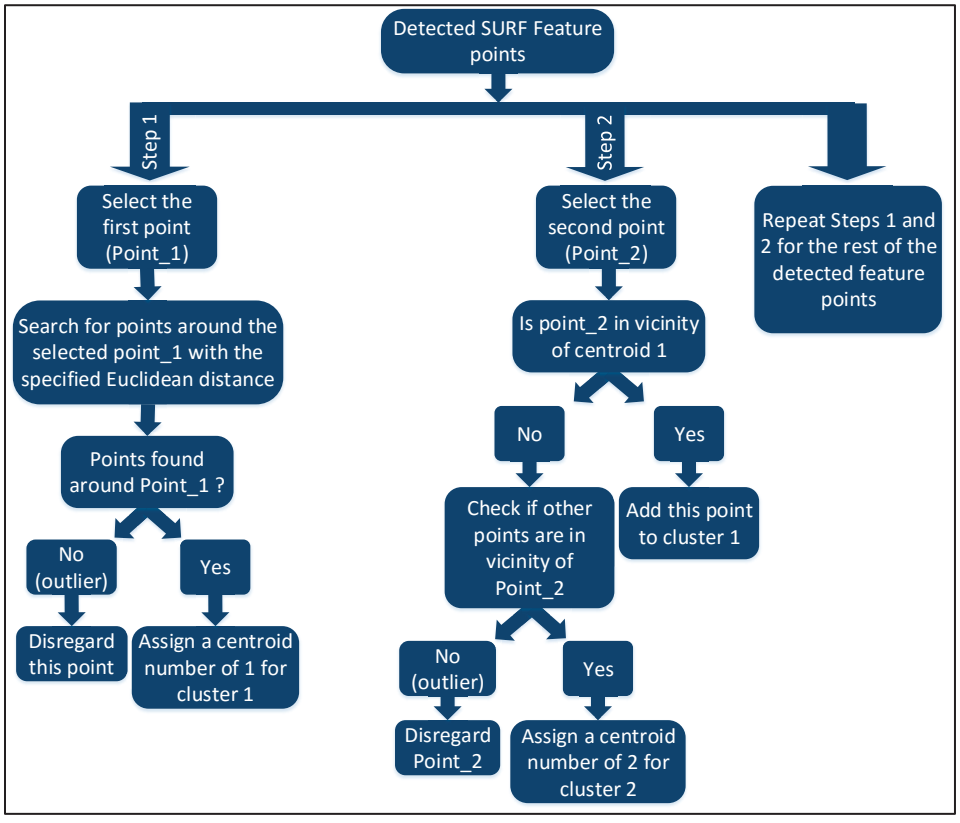



Figure 3-21: Flowchart of the clustering function

The novelty of this clustering algorithm is that it does not have a fixed search radius and it is adaptable to points that are close together as well as points that are spread out. This function searches for points within the vicinity of each feature point and assigns them to clusters, rejects outliers and merges clusters that are too close to each other, with an assumption made that clusters too close together might belong to the same object. This function performs similar to k-means algorithm, but what k-means is not equipped with is the ability to adapt to different kind of points.

There will be scenarios where points may not always be close together but sometimes, if the object covers a large area compared to the whole image, then the points detected would be scattered and the same object would be divided into several clusters, but would still belong to one single cluster. K-means would create several centroids, but using this algorithm, small clusters would be merged into one single cluster.

The *intraSearchRadius* is computed by calculating the average distance between all the feature points. The average of this distance vector suggests the approximate *centroid_disp_limit* which can be used to compute *intraSearchRadius* by subtracting 30 from the average of distance vector. For the points shown in Figure 3-22 (a), *centroid_disp_limit* came out to be 53 and *intraSearchRadius* is then 23.

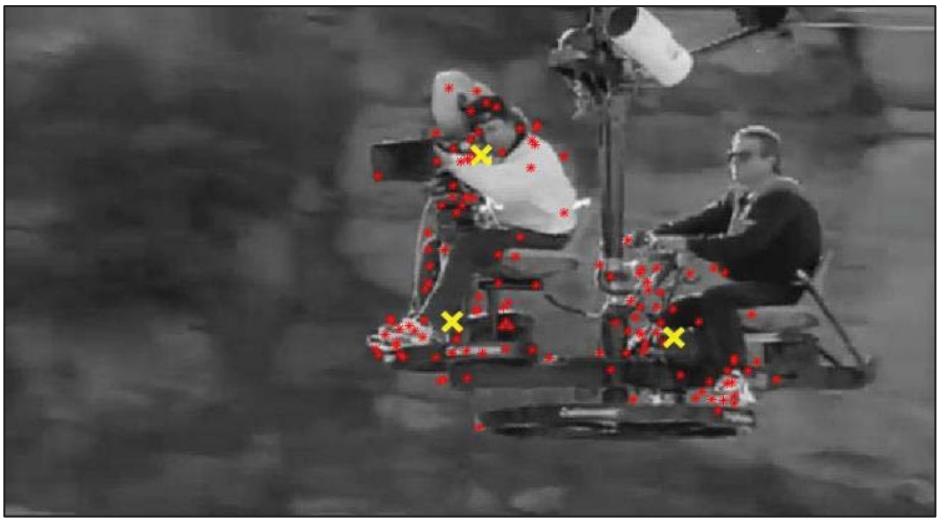


Figure 3-22: Shows two cameramen with feature points plotted in red and cluster centroid marked in yellow

Figure 3-23: Shows bounding boxes around each cluster

As shown in Figure 3-23, bounding boxes that are overlapping or have common boundaries as well as the centroids which are less than *centroid_disp_limit* apart would be merged. The resulting cluster in Figure 3-24 shows a single cluster with a bounding box displaying the whole object which is in motion while the background is also moving. This is how the *clusterPoints* function can not only cluster points using an automatic search range but also merges small broken up clusters which essentially belong to one major cluster by combining overlapping boxes and boxes are the very close to each other.

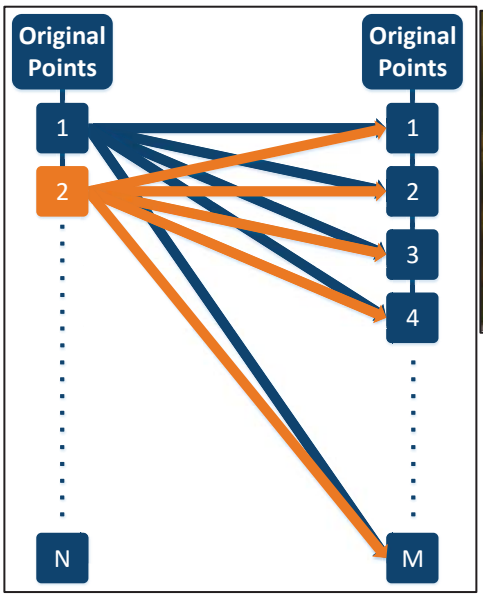



Figure 3-25: Shows the searching of points in vicinity

Figure 3-24: Single cluster comprising of all the small clusters

```
for i = 1:size(temp_matrix,1)

    % get location of start points with each iteration of index
    x1 = originalPoints(i,1);
    y1 = originalPoints(i,2);

    % search for points within each original Point vicinity
    for j = 1:size(originalPoints,1)
        x2 = originalPoints(j,1);
        y2 = originalPoints(j,2);

        % calculate displacement with each point
        x = abs(x1-x2);
        y = abs(y1-y2);
        displacement = sqrt((x)^2 + (y)^2);
```

Code snippet 3-15: searching of points in the vicinity of x1,y1, following Figure 3-26. The variables x1,y1 are SURF feature points detected on image 1

Referring to Figure 3-26, the feature point marked yellow (say point X) is a point around which all the points within its vicinity are detected. The yellow circle depicts the *intraSearchRadius* which searches through all the feature points to check which points are within the yellow circle. If point X already belongs to a centroid, then all the points that are within the yellow circle get added to the same centroid. On the other hand, Figure 3-25 depicts the searching of points within the vicinity of the point under processing.

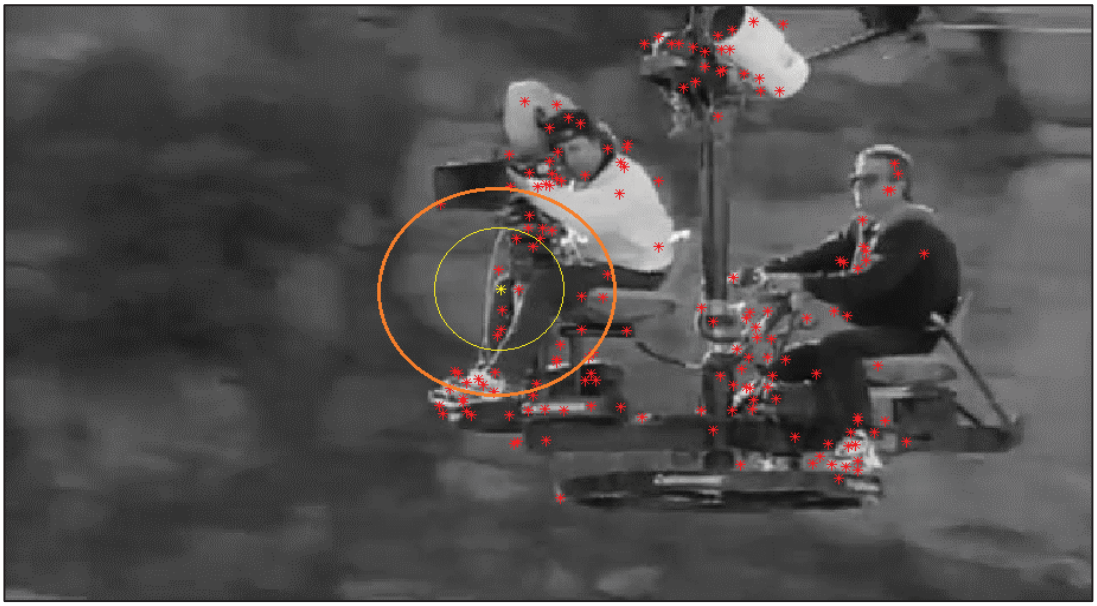


Figure 3-26: A point marked yellow with intraSearchRadius (yellow) and centroid_disp_limit (orange)

Code snippet 3-15 takes a set of points *x1,y1* and run it through a loop to search through all the points to find any points that are in the vicinity of *x1,y1*. Figure 3-25 depicts the searching process of taking a point (*x1,y1*) and search for it through all of the points in *originalPoints*. All the feature points are stored in a single Nx2 matrix. With every search iteration, the Euclidean distance between *x1,y1* and *x2,y2* is computed and if this distance is less than or equal to *intraSearchRadius* then *x1,y1* and *x2,y2* belong to the same cluster. If *x1,y1* is unable to find any points within its vicinity, then a value of zero is marked under *flag_matrix* which will be later be rejected. Code snippet 3-16 checks if *x2,y2* and *x1,y1* are within distance of *intraSearchRadius* apart. A flag is set for *x2,y2* when a match is found which reserves this point in the cluster that *x1,y1* belong to.

```
% x1,y1 must be part of this cluster as its close to x2,y2.
% if x1,y1 is within the vicinity of x2,y2
if displacement <= searchRadius && ~isequal(displacement,0)

      % set flag for location of x1,y1 in originalPoints upon
      finding a match
      flag_matrix(flag_matrix_index,1) = clusterNumber;
      flag_matrix(flag_matrix_index,2) = i;
      flag_matrix_index = flag_matrix_index+1;
end
```

Code snippet 3-16: Check if x2,y2 lies within intraSearchRadius of x1,y1

Code snippet 3-17 shows how the centroid locations are stored. *centroid_number* is the index for the matrix named *centroid* which keeps track of the centroid locations. This index variable *centroid_number* gets incremented when a new centroid is created. Each time a new point is found in the vicinity of point *x1,y1*, it gets added to *flag_matrix*, which is then used to compute a new centroid location by taking a mean of all the *x* and *y* values of the points in that cluster. The new cluster centroid is stored in *centroid* at location *centroid_number*.

```
% calculate centroid of the cluster as new points get added
centroid(centroid_number,1) = round(mean(mX));
centroid(centroid_number,2) = round(mean(mY));
```

Code snippet 3-17: Storing centroid locations

```
for p = 1:size(centroid,1)
      mX = (centroid(p,1) - x1);
      mY = (centroid(p,2) - y1);
      x1y1_cent_disp = sqrt(mX^2 + mY^2);

      % x1,y1 within vicinity of centroid
      if x1y1_cent_disp <= centroid_disp_limit
            % add x1,y1 to the existing cluster
            numberOfRows = size(flag_matrix,1);

            % cluster number to add x1,y1 to
            flag_matrix(numberOfRows+1,1) = p;
            % location of x1,y1 in originalPoints
            flag_matrix(numberOfRows+1,2) = i;

            % if centroid found, then clear this flag
            create_new_centroid = 0;
            execute_next_i_point = 1;
            break;
      else % if x1y1_cent_disp is greater than
centroid_disp_limit
            create_new_centroid = 1;
      end
end
```

Code snippet 3-18: Nested for-loops to search for points in vicinity using intraSearchRadius

After computing the coordinates of the centroid by finding out the mean for the centroid of a cluster, Code snippet 3-18 checks through all the clusters that are found, at every iteration of *x1*,*y1*, to confirm if this point can be included in the cluster even though *x2*,*y2* recognizes this as a part of its cluster. This is a good check as sometimes the outliers lie close to a cluster but do not belong to any cluster. To compute this, the distance between *x1*,*y1* and each centroid position is considered. If this distance is less than *centroid_disp_limit*, then the point *x1*,*y1* is added to the centroid under processing.

## 3.8 Colour measurements

The most commonly used colour space for images is Grayscale where there are only 256 different intensities of pixels ranging from black (0) to white (255) which makes processing of images a lot easier and straight forward. But colour measurements in image processing can prove to be a good tool for various applications, especially when it comes to identifying objects in the scene or identifying the scene itself. The RGB colour space consists of three colour channels; Red, Green and Blue, with each channel acting as a separate colour space for which intensities range from 0 to 255.

HSV colour space has been used for this research. HSV is also one of the most common colour spaces used for recognition. It separates Hue, Saturation and pixel intensity Value from colour images and provides a contrast of three grayscale images from the three channels, H, S and V. Test results have shown that the detection of specific colours on HSV scale provide better results than using different channels of RGB space. The threshold values to detect different colours in the image are quite narrow for RGB space to detect different shades of the same colour. Whereas for HSV, there is a wider range of values for basic colours like red, green and blue, each with different shades. As an example, Figure 3-30 shows the RGB and HSV colour spaces broken up into individual channels. This is done by specifying the low and high threshold values for each channel of the HSV plane. Table 3-4 shows the low and high threshold values for each colour in the RGB and HSV colour space. It can be seen from the table that the colour red for RGB colour space does not provide a wide enough threshold of values, even though the red band is the widest band in the visible colour spectrum. HSV is known to be better than RGB in regards to colour detection. In addition, HSV colour space is quite similar to the way humans perceive colour, which helps in recognizing areas of distinct colours. There are also other colour spaces such as CIE XYZ, YCbCr and Lab. Some of these have Matlab support, but for the general detection of colours, the HSV colour space is widely used and recommended.

*Note, the values shown in the table have been averaged out using values from four different colour images and not just the original image shown in Figure 3-28.*

Table 3-4: RGB and HSV low/high threshold values

| | RED | | GREEN | | BLUE | | WHITE | | GRAY | |
|---|---|---|---|---|---|---|---|---|---|---|
| Color Space | low | high | low | high | low | high | low | high | low | high |
| Red | 0.81 | 1.00 | 0.36 | 0.49 | 0.35 | 0.41 | 0.91 | 1.00 | 0.35 | 0.65 |
| Green | 0.12 | 0.19 | 0.51 | 0.65 | 0.58 | 0.64 | 0.91 | 1.00 | 0.38 | 0.68 |
| Blue | 0.14 | 0.20 | 0.23 | 0.31 | 0.69 | 0.78 | 0.86 | 1.00 | 0.40 | 0.66 |
| | | | | | | | | | | |
| Hue | 0.80 | 1.00 | 0.15 | 0.50 | 0.60 | 0.68 | 0.00 | 1.00 | 0.55 | 0.70 |
| Saturation | 0.58 | 1.00 | 0.36 | 1.00 | 0.85 | 1.00 | 0.00 | 0.36 | 0.03 | 0.15 |
| Value | 0.55 | 1.00 | 0.20 | 0.80 | 0.32 | 0.63 | 0.60 | 1.00 | 0.40 | 0.65 |

Figure 3-28: Original Soccer frame

Figure 3-29 shows different colours that were detected in the original image. This is done by specifying the low and high threshold values for each channel of HSV plane as shown in Table 3-4.

Figure 3-29: Detected colour regions using HSV thresholding from left to right, red, green, blue, white, gray.

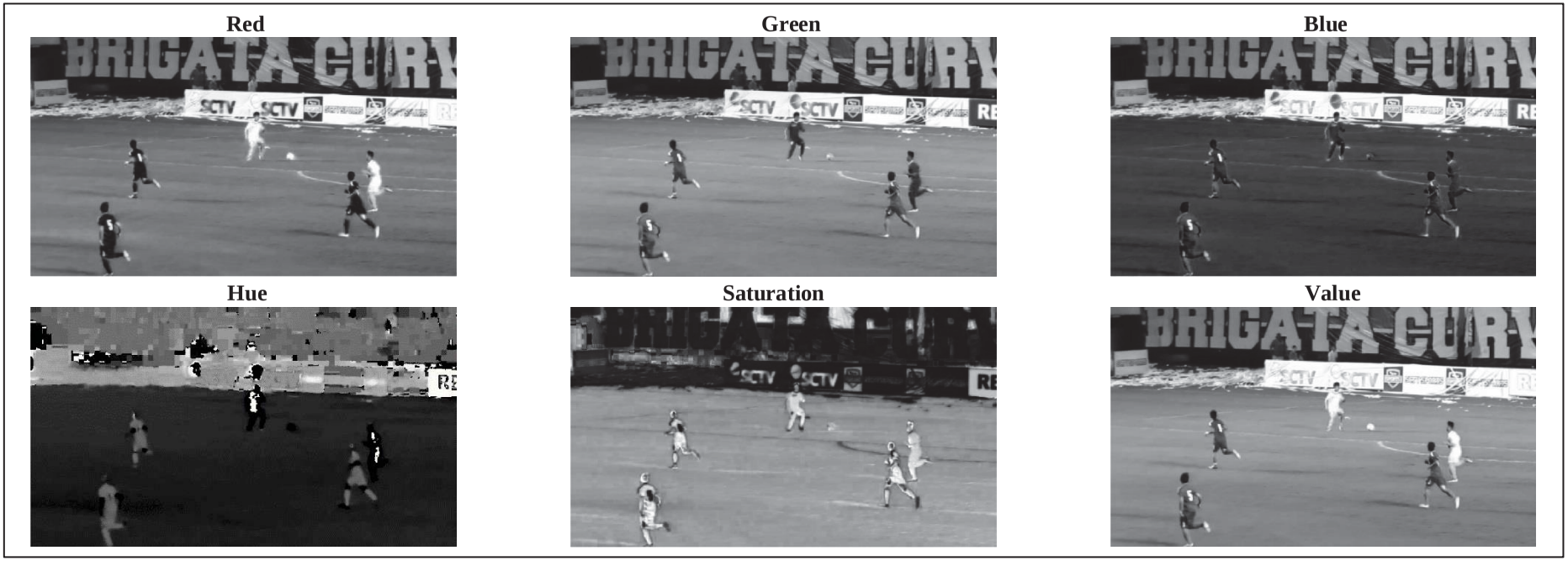



Figure 3-30: Extracted R,G,B and H,S,V planes on first and second rows respectively for the original RGB frame shown in Figure 3-28

```
function detColourImg = detectColour (rgbImage, desiredColour,
dispDetColour, displayHSVimages)
```

Code snippet 3-19: Function prototype for detection of colour

Code snippet 3-19 shows the colour detection function *detColour()* which takes four inputs. The original frame for which colour needs to be detected is *rgbImage*. The desired colour is specified by the *desiredColour* input which accepts the colour name as a string. The desired colour mask is shown by a flag named *dispDetColour*. If this flag is set to 1, the colour mask for the stated colour will be shown. Another flag is *displayHSVimages*, which is used to display the Hue, Saturation and Value of images from the *rgbImage* input. If this flag is set to 1, HSV images will be displayed as shown on the bottom three images in Figure 3-30. This function returns the image mask of the desired colour for the original RGB frame. This mask is a binary image which consists of pixels that pertain to the specified colour set of white (1). The ones that do not belong to the specified colour are set to black (0), which can be seen from Figure 3-29 which shows different colours that are detected from the original RGB image.

```
% Convert RGB image to HSV
hsvImage = rgb2hsv(rgbImage);

% Extract the H, S, and V images individually
hImage = hsvImage(:,:,1);
sImage = hsvImage(:,:,2);
vImage = hsvImage(:,:,3);
```

Code snippet 3-20: Converting the images to HSV space using hsvImage() function

Code snippet 3-20, uses the Matlab function *rgb2hsv()* to convert the colour image into HSV colour space. The individual H, S and V channels are extracted from the newly created *hsvImage* and named *hImage*, *sImage* and *vImage* respectively. The *rgb2hsv()* function converts each of the colour planes to a data type of double, which means the values range from 0 to 1.

```
% Apply each colour band's particular thresholds to the colour
band
hueMask = (hImage >= hueThresholdLow) & (hImage <=
hueThresholdHigh);

saturationMask = (sImage >= saturationThresholdLow) & (sImage <=
saturationThresholdHigh);

valueMask = (vImage >= valueThresholdLow) & (vImage <=
valueThresholdHigh);
```

Code snippet 3-21: Computing mask for each H, S and V plane

Code snippet 3-21, computes the mask for each of the H, S and V planes and applies the thresholds retrieved from the *SetThresholdsDetectColour()* function (which will be explained later in this section) to each colour plane. This function checks each pixel intensity value of the specified plane and compares it to the low and high threshold levels. The result is a binary image of the mask consisting of each pixel intensity value that is greater than the low threshold value, and vice versa.

```
% Combine the masks to find where all 3 are "true".
% Highlighting only the specified colour parts of the image
colouredObjectsMask = uint8(hueMask & saturationMask &
valueMask);

smallestAcceptableArea = 6;

% Get rid of small objects.  Note: bwareaopen returns a logical.
colouredObjectsMask = uint8(bwareaopen(colouredObjectsMask,
smallestAcceptableArea));
```

Code snippet 3-22: Performing AND operation to check where all three are true

After computing the thresholds for all the three H, S and V planes, Code snippet 3-22 combines all three to give the overall mask for the specified colour of the image. The AND operation fuses all the three masks together. This helps eliminate any wrong colour matches that occur at any of the planes and keeps the ‘true’ values where the pixel intensity of the same pixel on all the three colour planes is 1. The example shown in Figure 3-31 portrays the masking process where three matrices of 3x3 are masked together using the AND operation to find the pixel that belongs to the desired colour on all the three-colour planes. The resulting image mask shows one pixel that is white while others are black because those pixels were not true on all the three planes. The same process is undertaken for masking all the three colour planes. The masked image consists of a lot of small specs which need to be excluded from the mask in order to make scene recognition more efficient, which is what the *bwareaopen()* function in Matlab does. It removes closed components from binary images with areas less than a certain number of pixels and sets them to an intensity level of 0.

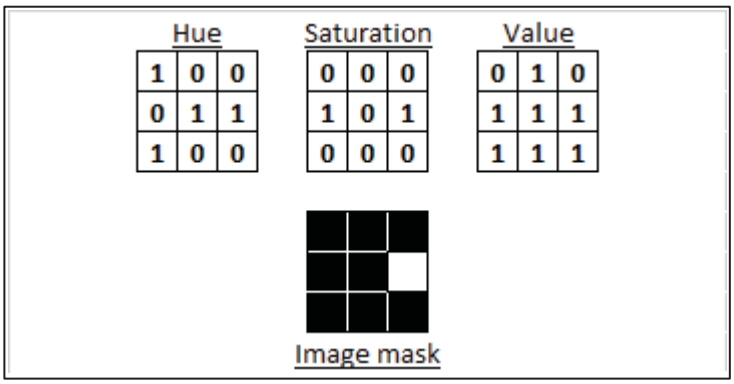



Figure 3-31: Shows the masking process using the AND operation

```
% Using the high and low threshold values for each colour
[hueThresholdLow, hueThresholdHigh, saturationThresholdLow,
 saturationThresholdHigh, valueThresholdLow, valueThresholdHigh]
=
 SetThresholdsDetectColour(desiredColour);
```

Code snippet 3-23: SetThresholdsDetectColour() function to set colour thresholds

The code Code snippet 3-23 shows the prototype of the thresholding function *SetThresholdsDetectColour()* which takes the name of the colour, by way of a string, as its only input denoted by *desiredColour* and returns the low and high threshold values for each of the H, S and V planes for particular colour. The thresholds are programmed manually into the function with low and high threshold values, also extracted manually. Four different images were used as training images to extract the thresholds with varying brightness levels. Code snippet 3-24 shows a switch-case based system to find the desired colour in the image.

```
S = desiredColour;
switch S
      case {'Red', 'red'}
            % Red.
            hueThresholdLow = 0.80;
            hueThresholdHigh = 1;
            saturationThresholdLow = 0.58;
            saturationThresholdHigh = 1;
            valueThresholdLow = 0.55;
            valueThresholdHigh = 1.0;
      case {'Green', 'green'}
            % Green
            hueThresholdLow = 0.15;
            hueThresholdHigh = 0.50; %it was 0.60
            saturationThresholdLow = 0.36;
            saturationThresholdHigh = 1;
            valueThresholdLow = 0.2; %it was 0
            valueThresholdHigh = 0.8;
```

Code snippet 3-24: Detection of colour using thresholds in a switch-case system

Figure 3-32 shows a colour detection flowchart portraying a colour detection system which starts by converting the RGB image into HSV space, followed by extracting H, S and V planes individually. Next, it detects the specified colour using the thresholds, which is then used to compute the mask by combining all the masks together. Regions that are smaller than 6 pixels in area are discarded. All the threshold values are normalized.

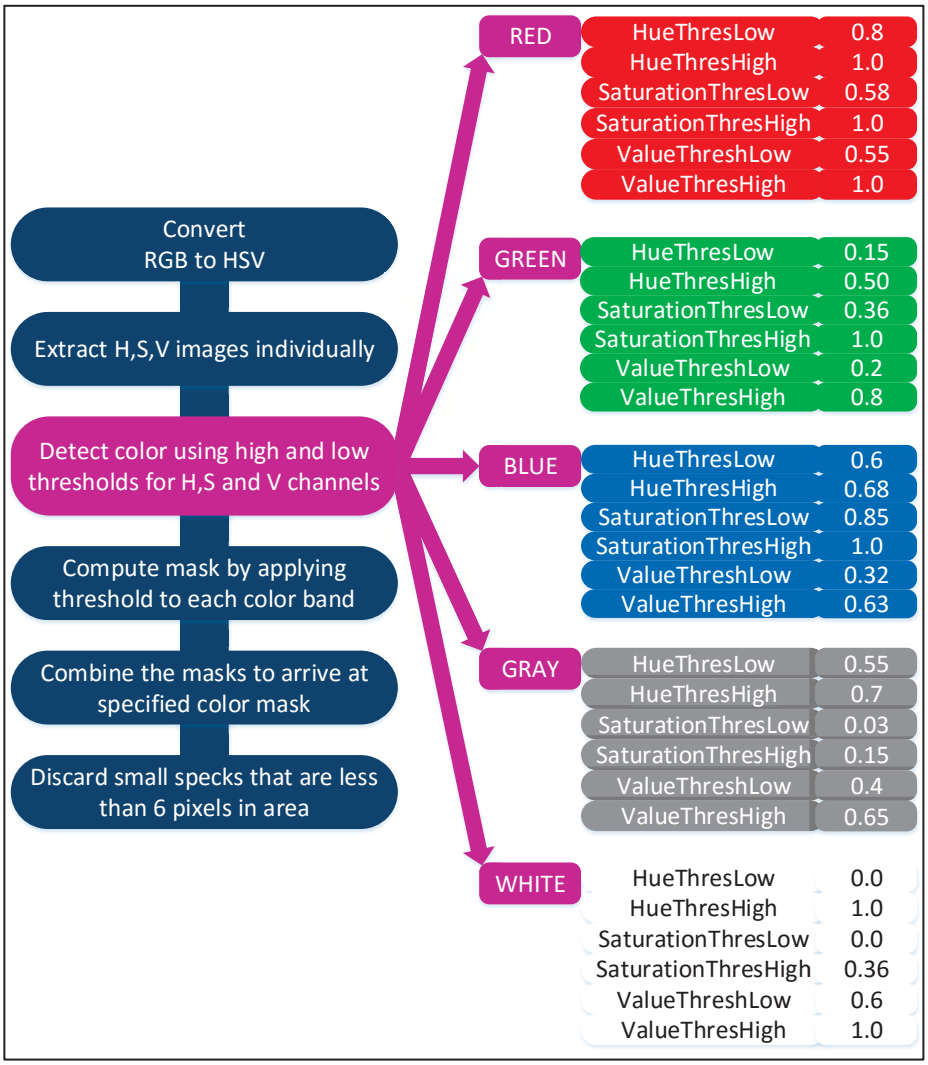



Figure 3-32: Flowchart of colour detector and colour threshold values in HSV space

## 3.9 Heuristics based Artificial Intelligence

The book by Russell and Norvig [64] explains artificial intelligence as being a very broad area with many different areas of reasoning such as logical reasoning systems, probabilistic reasoning systems and case-based reasoning systems, to name a few. For the work done in this research, Case-Based Reasoning (CBR) system has been adopted to create a knowledge base that is set upon cognitive method to build a structure on how we, as humans, perceive our surroundings.

Case-based reasoning uses precedents of prior decisions or actions. In CBR terminology, the set of previously solved problems are called the Case Base. It consists of cases which are problem instances, consisting of a problem description part and a solution part. Case Base assumes that the world is regular, so similar problems should have similar solutions [64]. CBR is inspired by the approach that humans take to solving problems. People do not solve every new problem from scratch, but instead decisions are made from past experiences. These past experiences are incorporated using heuristics.

Heuristics are shortcuts for human behaviour and perception, which involve taking data from the environment and transforming it into knowledge. Heuristics can also be thought of as the common sense that is applied to a problem, based on an individual's observation of a scene or environment. It is a practical and prudent tactic that is applied to a decision where a clear separation of right and wrong can easily be established. The use of these shortcuts and common sense is applied to this research work to answer simple questions about the awareness of the scene. Earlier, databases of object components and bag-of-features have been a focus for object recognition. Study shows that these methods takes approximately 20 to 40 seconds to find a match while consuming a lot of memory for computation which is not viable for real time applications. Some papers have been based solely on the indexing algorithm of speeding up the process of finding a match in the database. Heuristics, along with colour measurements, make up the perception technique. Perception is aimed to fix this issue of using databases which would detect what the object of interest really is, by completely bypassing databases and searching algorithms for large databases of trained images. Instead, the context of the object and its surroundings is studied, which would eliminate the need for large databases and also reduce the computation time.

Take for example, a scene of moving objects with different velocities. The shape of some objects might be tall and narrow while others might be short and wide. These shapes can be thought of as vertical and horizontal rectangles respectively. The context would provide information about the scene such as the sky, plants, trees and roads, which would narrow the context down to an outdoor scene. Often, two things that are most commonly in motion are people and vehicles. People tend to have tall yet narrow shapes, whereas vehicles would have more of a short yet wide, or even a square shape. Considering the optical flow of the objects that are moving in the scene, and their shape along with judging the environment, it can be said that it is an outdoor scene consisting of the road and greenery. So, vertical rectangular detected objects could be people whereas horizontal rectangular objects could be vehicles, if their optical flow is larger than the people's.

A new method is proposed, based on heuristics, which is implemented using case-based artificial intelligence to add a perceptive approach to object recognition. Figure 3-33 shows a diagram of an algorithm depicting perception using a case based reasoning system. This integrates the detected object's shape as well as the detects of colours present in the image. The algorithm first differentiates the setting of the scene between an indoor or an outdoor setting, using colours in the image and the object's shape. Next, the algorithm looks for distinctive features in the image that can quickly validate the setting, such as the floor or walls for an indoor setting, and concrete, road, grass or sky for an outdoor setting. The colour measurements to work out whether the image has an indoor or outdoor setting is fairly straight forward. Next follows the determination of fine details of the scene such as houses, buildings, pedestrian crossing, road signs, plants, trees, grass, etc. The light green and blue blocks in Figure 3-33 can be thought of as cases for each scenario pertaining to either an indoor or an outdoor setting. If two or more conditions in a block are satisfied, then the object can be assumed to be a part of the scene shown in dark green and blue circles.

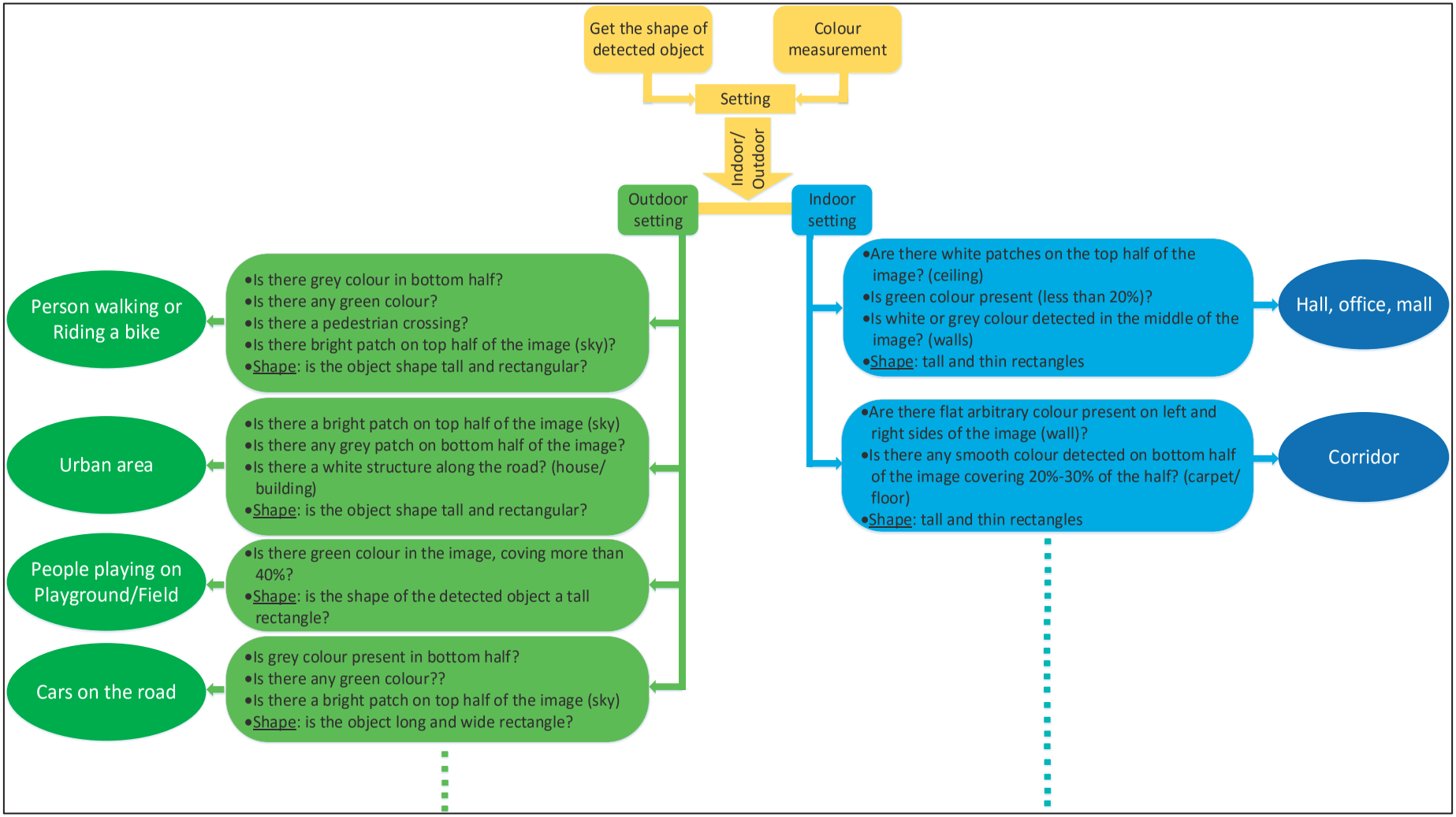



Figure 3-33: Perceptive approach using object shape, colours in the image and heuristics

### 3.9.1 TensorFlow and heuristics approach to object recognition

In this research, Hue, Saturation and Value images are converted from an RGB colour image. The colours are detected by thresholding each of the three images to find the desired colour. Each colour is detected within a certain range. If the threshold values do not fall within this range, then the desired colour is not detected. Thresholding is just an estimation that is not 100% accurate, and some minute colour information can also go undetected.

To solve this issue, we explore alternate ways of detecting the aspects of the scene without the detection of colours using thresholds. This is where recent work done in the area of AI like TensorFlow, comes in. TensorFlow is an open source library for machine learning developed by Google, which can be used to build a convolutional neural network (CNN). It can be used to detect key features of the scene in an image or frame such as walls, ceiling, floor, carpet, grass, trees, road, etc. These key features can be used instead of colour measurements to decipher aspects of the scene and estimate the class of the detected object using heuristics.

To achieve this, images of walls, ceiling, carpet, grass, road and few others were downloaded from Google Images and trained using TensorFlow. These were chosen because walls, ceiling, roads, etc. are not always the same brightness and colour changes and contrast variations are inevitable which would differ when it is bright and sunny as opposed to when it is cloudy. Hence, training these images for different scenarios would eliminate these brightness changes and colour differences.

TensorFlow is a library for building graphical interfaces and building complex deep learning algorithms. Each operation such as addition or subtraction takes an input as a tensor and outputs a tensor as well. A tensor is how data is represented in TensorFlow. Tensors are multi-dimensional arrays of numbers, and they flow between operations. Hence the name, TensorFlow.

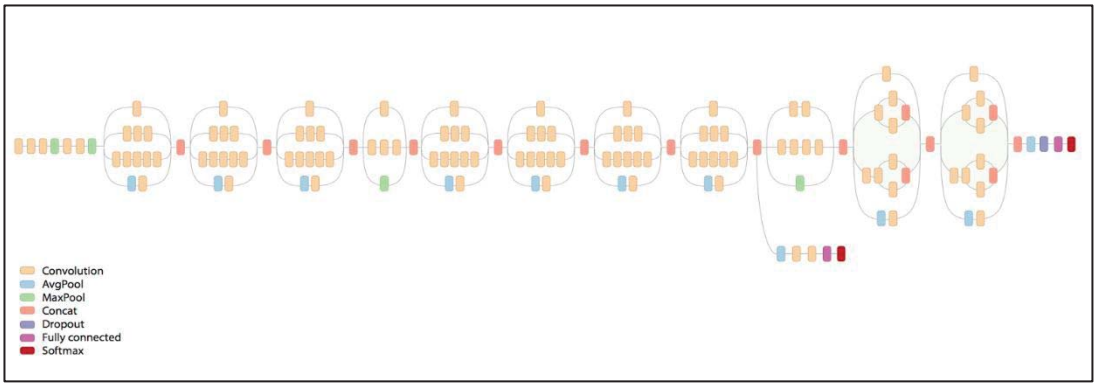



Figure 3-34: Inception model

Pre-trained CNN model as shown in Figure 3-34 taken from [65] is called inception was trained by Google on hundred thousand images with a thousand categories. Training user specified images, requires a process called transfer learning which means applying learnings from pervious training session to a new training session. Figure 3-34 shows the inception model. When data such as an image is fed in to the input, at each layer, it will perform a series of operations on that data until it outputs a label and classification percentage. Each layer has a different set of abstraction. In first few layers, techniques such as edge detection, feature detection take place, then shape detection in middle layers, and as the layers progress, the model gets more and more abstract. Martin et al. [66] defines how TensorFlow works and describe the flow of information that occurs between several layers of inception model. The scene recognition results obtained by TensorFlow are shown in Section 4.3.

# Chapter 4 Testing Results

This section displays the results that were acquired after processing through the proposed algorithm as well as some errors and challenges that were encountered. A comparison with results of other researchers have also been touched upon. Most of the test frames obtained from video sequences are from Matlab database, YouTube and BMS dataset.

## 4.1 Object detection using Optical flow and Clustering

The processing of optical flow and clustering are the two rudimentary steps to finding the objects of interest in a scene. Two types of results are shown in this section:

1. Results that have been achieved by researchers in the past as shown in Section 4.1.1
2. Results obtained by the proposed algorithm shown in Section 4.1.2.

The two results are compared on performance and efficiency in areas of optical flow computation and clustering and successful object detection.

### 4.1.1 Results obtained by researchers for optical flow and object detection

Results from few research works that revolved around detection of moving objects while the observer (camera) is also in motion are displayed in this section for achieving some degree of accuracy. Table 2-1 presents more information on efficiency of results obtained by researchers for whom the results are shown in this section.

Figure 4-1 displays two frames of the planes sequence taken from [30]. It can be seen that the background very smooth and the only pixel change that the image encounters is the change between the background and the edges of the planes. Due to the smooth

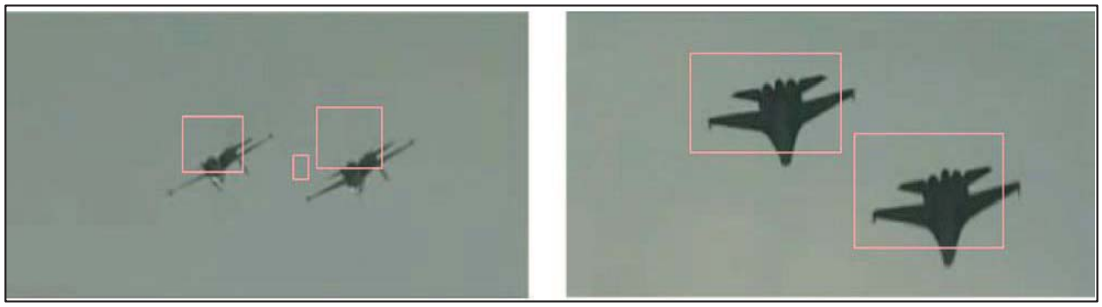


Figure 4-1: Result from [30] showing detection of objects on smooth background

background, computation would not be required for these areas, hence, low computation time. The planes are also in focus and stands out compared to the background making it rather straightforward to detect these objects.

Figure 4-2 displays result from [25] showing SIFT feature points on the car as well as the background. The car is the object of interest. SIFT detects points in abundance for both the car and the background making it easier to differentiate between the two.

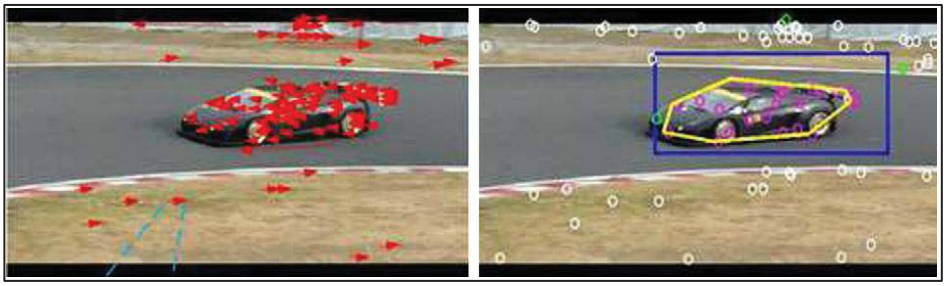


Figure 4-2: Results from [25] showing feature extraction (left) and the detected object in the foreground

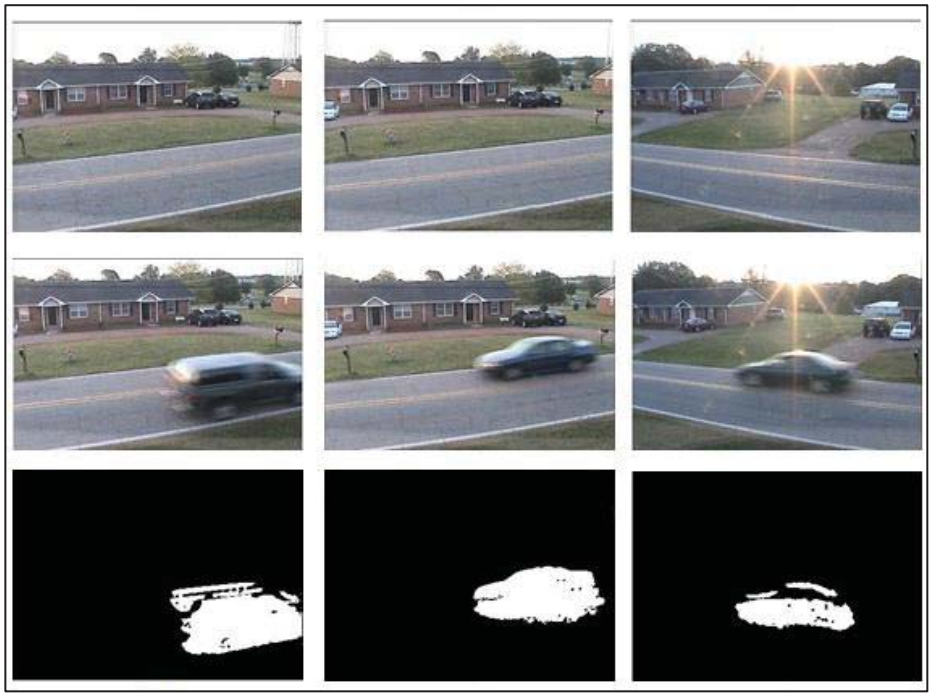


Figure 4-3: Results from [20] displays background modelling (top row), object moving in the foreground (middle row), detected object (bottom row)

Figure 4-3 displays results from [20]. Videos captured from a PZT camera. When the camera moves to a new spot, the background gets modelled first then and then processing of detecting moving objects take place. This scenario is suitable for situations where the camera itself is stationary and moves about its axis covering the

surroundings to a certain degree. Applying frame differencing when the background frame is established, hence a successful detection of moving objects even when the object appears blurry.

Figure 4-4 displays results from [19] showing a moving object while the camera is also moving. The moving object gets detected fairly well using feature extraction and MAP-MRF framework.

Figure 4-4: Results from [19] depicting detection of moving object while the background is also moving

Figure 4-5 displays the results from [22] showing two cars that are the objects of interest moving in the foreground while the camera is also moving. Using Harris corner detector and computing pyramidal LK optical flow along with K-means and RANSAC to arrive at the detected objects. Figure 4-6 displays the results obtained from [24] depicting the computation of optical flow for moving objects using the pyramidal LK method. The scene is shown to be a practical scene.

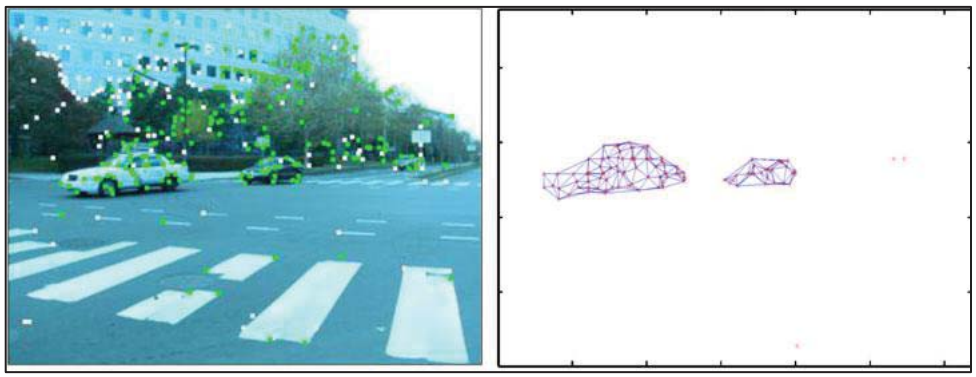


Figure 4-5: Result obtained from [22] displaying corner points detected using Harris corner detector (left) and the detected objects (right)

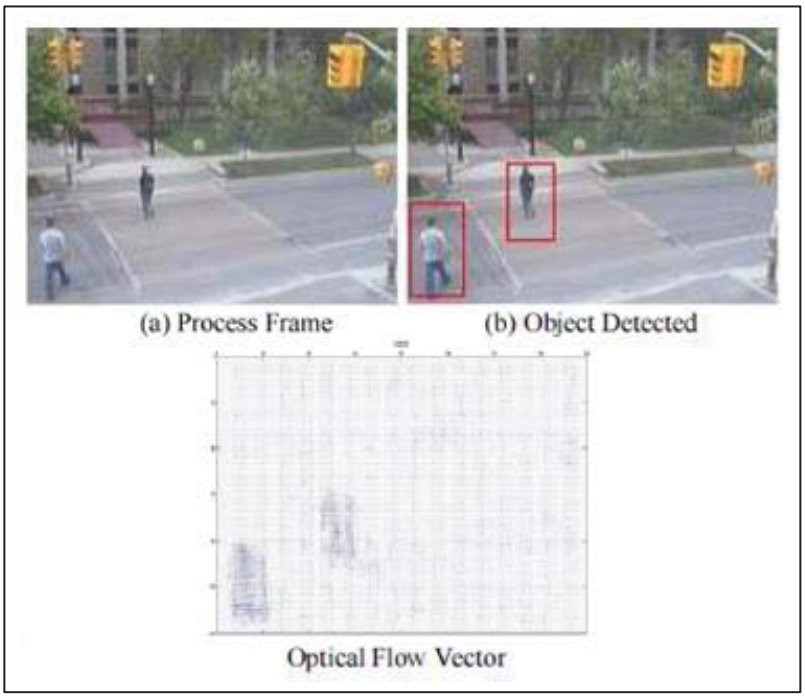



Figure 4-6: Results obtained from [24] Optical flow computed using pyramidal LK method

### 4.1.2 Results for object detection using the proposed algorithm

All the frames that are used to test the proposed algorithm were extracted from video sequences that involved moving objects in the foreground while the background was also moving. These sequences are chosen because of their relation to the real-world application.

Two pairs of frames chosen from each sequence are shown in this section. Consecutive frames have not been chosen in most cases because of nominal changes between two successive frames. Therefore, two frames were chosen with a certain amount of variation between them. Image (a) shows the optical flow computation as a result of point-wise cross-correlation between two frames. Image (b) shows the clustering of final set of points as a result of elimination of outliers. Image (c) depict the final motion vectors for the detected moving objects in the scene and final Image (d) displays a bounding box around the detected object.

Table 4-1 shows the performance of each video sequence used for testing of the algorithm. The table also presents minute details that are computed by the algorithm during execution to perform accurate elimination of points and grouping points into clusters.

**Bike Frames** (frames 127 and 129)

(a) Optical flow between frames 127 and 129

(b) Clustering using the proposed clustering

(c) Optical flow after elimination phase

(d) Detected object

Figure 4-7: Bike frames, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm (c) moving object detected as a result of optical flow (d) detected object

**Bike Frames** (frames 131 and 133)

(a) Optical flow between frames 131 and 133

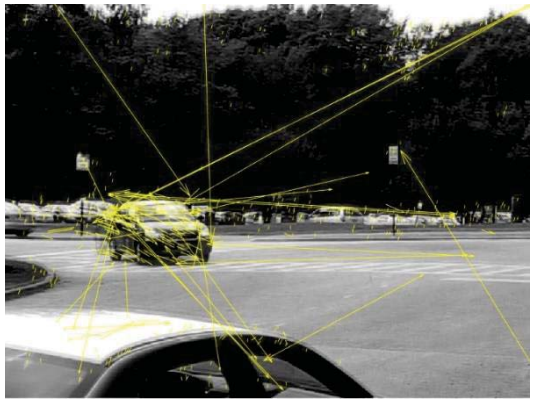
(b) Clustering

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-8: Bike frames, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

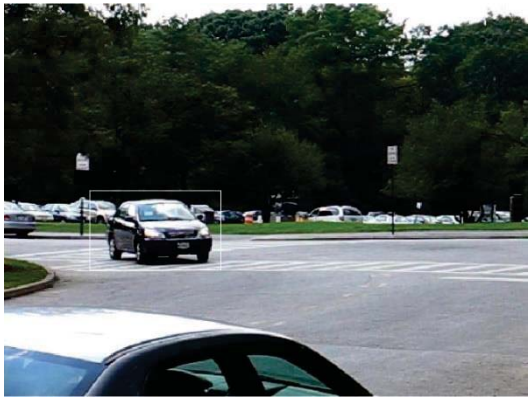
**Car sequence 1** (frames 1 and 5)



(a) Optical flow computation

(b) Clustering (discard cluster of size less than 6)

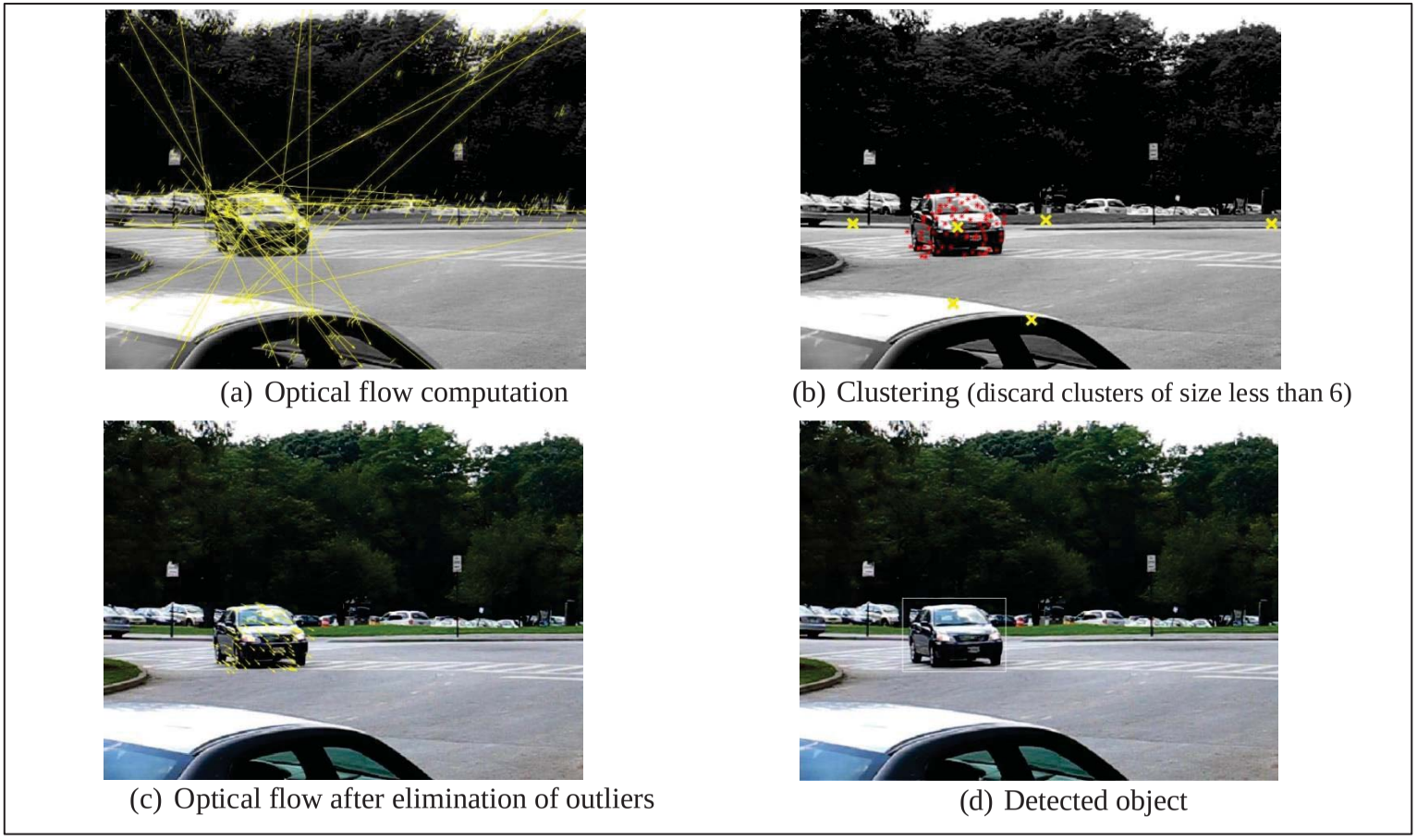
(c) Optical flow after elimination of outliers



(d) Detected object

Figure 4-9: Car sequence 1, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Car sequence 1** (frames 6 and 10)



(a) Optical flow computation

(b) Clustering (discard clusters of size less than 6)

(c) Optical flow after elimination of outliers

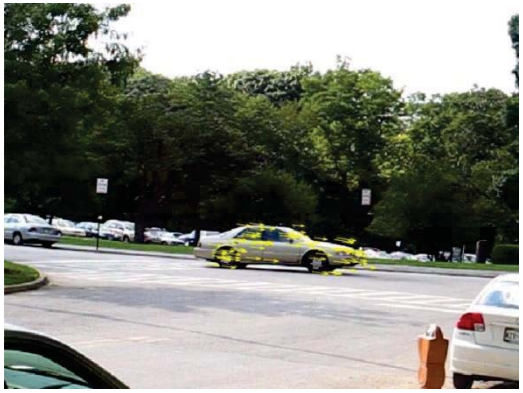
(d) Detected object

Figure 4-10: Car sequence 1, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Car sequence 2** (Frame 17 and 20)

(a) Optical flow computation

(b) Clustering

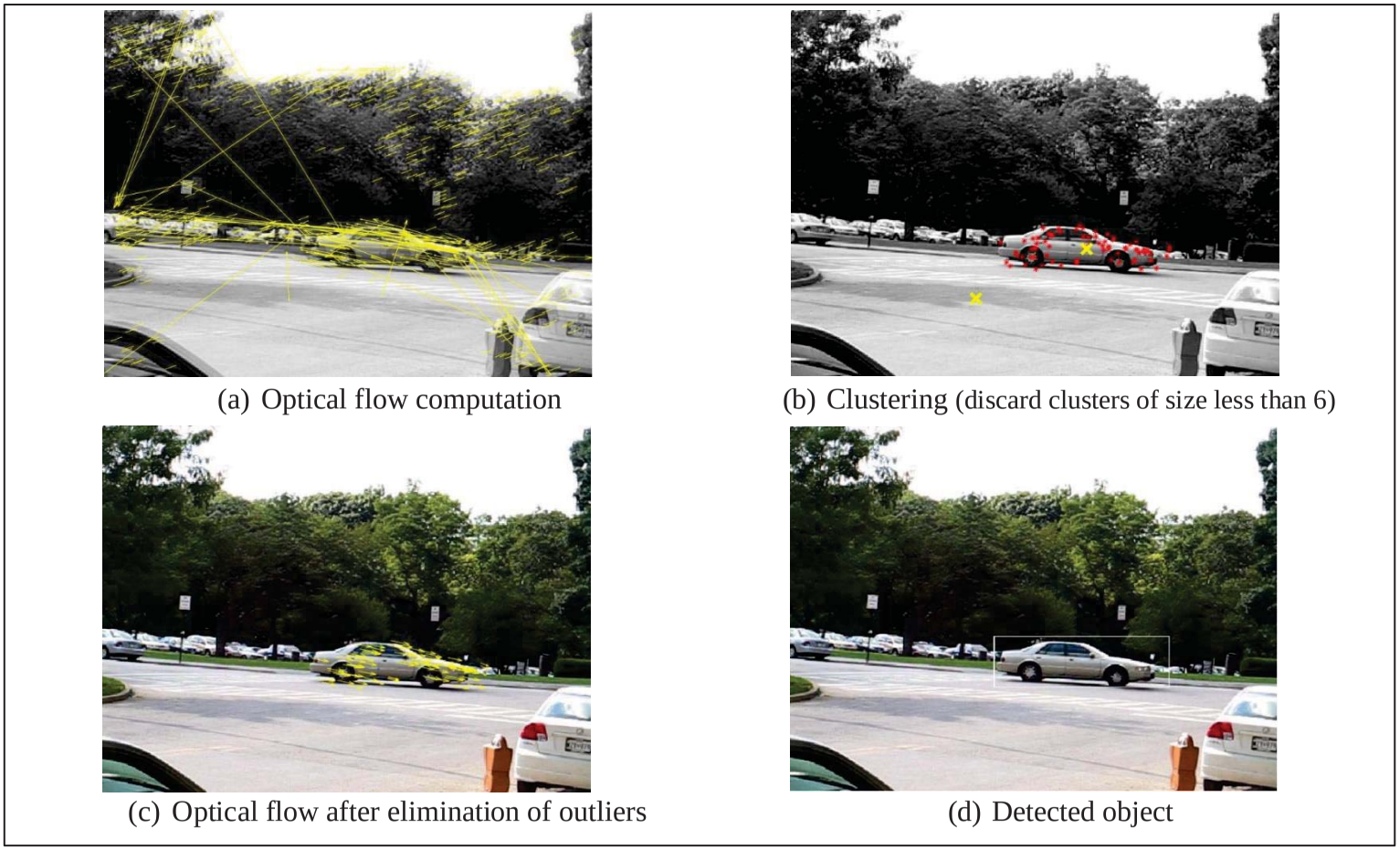


(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-11: Car sequence 2, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Car sequence 2** (Frame 20 and 24)



(a) Optical flow computation

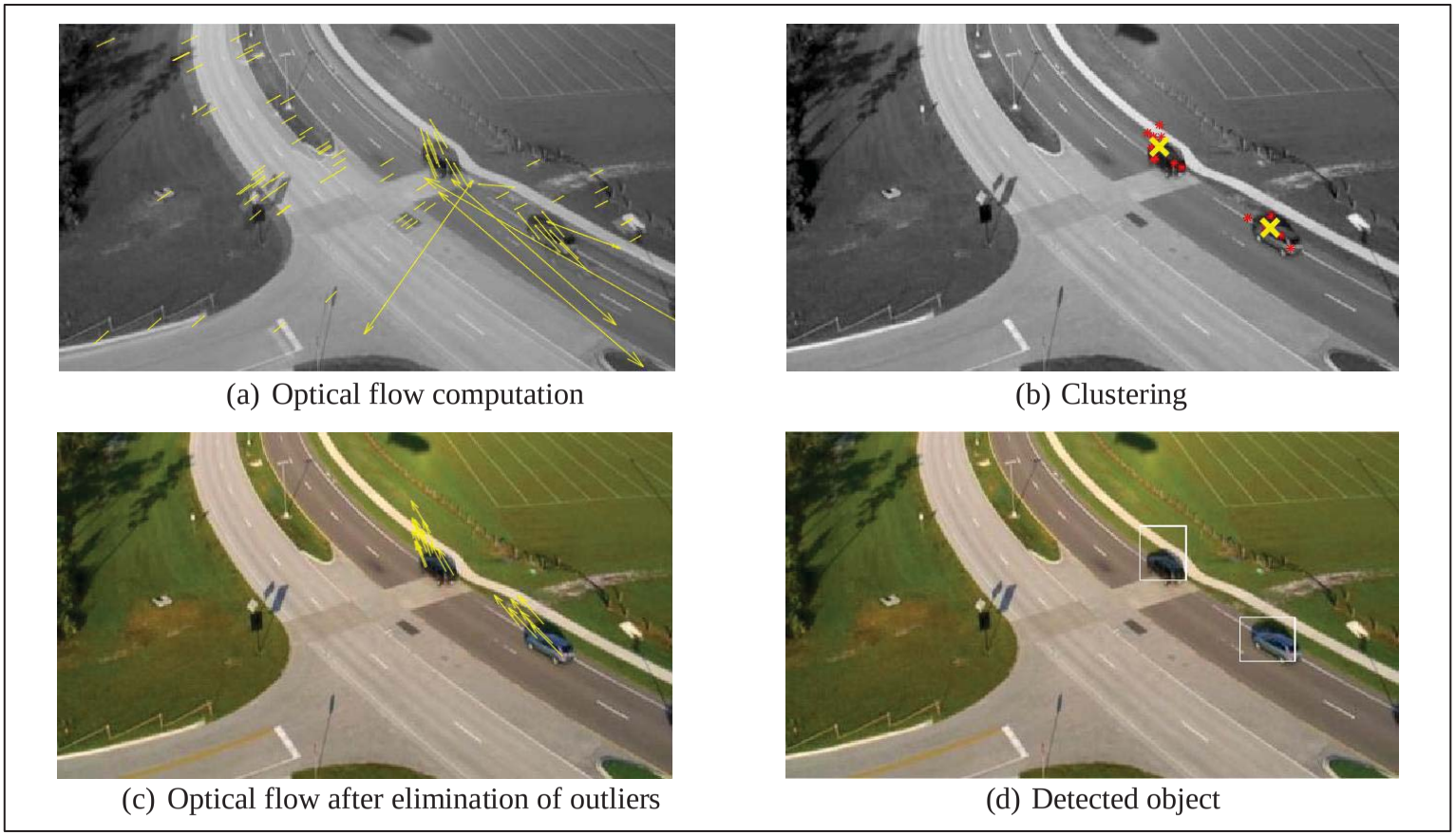
(b) Clustering (discard clusters of size less than 6)

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-12: Car sequence 2, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Aerial sequence 3** (Frames 25 and 30)



(a) Optical flow computation

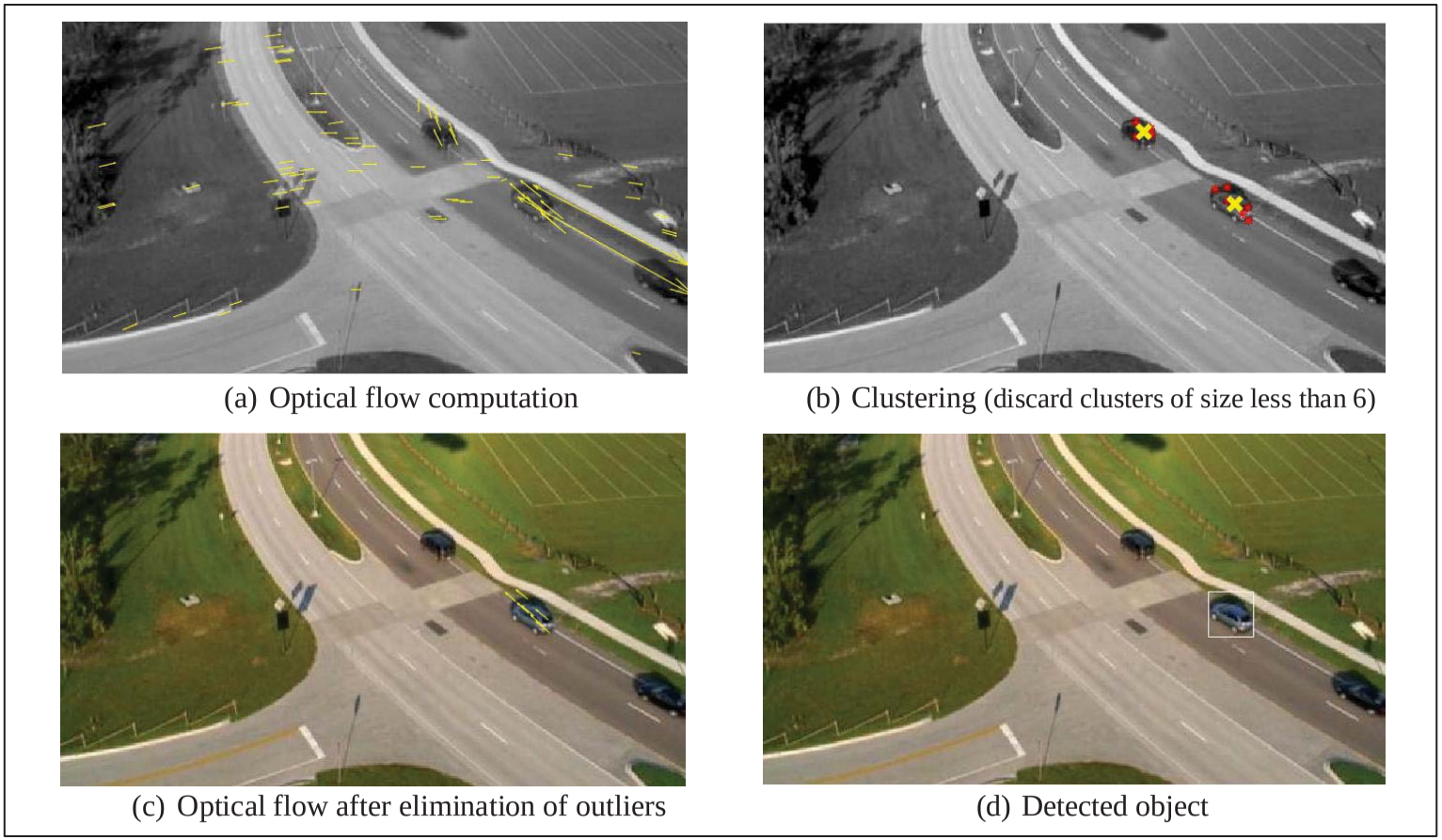
(b) Clustering

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-13: Aerial sequence 3, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Aerial sequence 3** (Frames 35 and 39)



(a) Optical flow computation

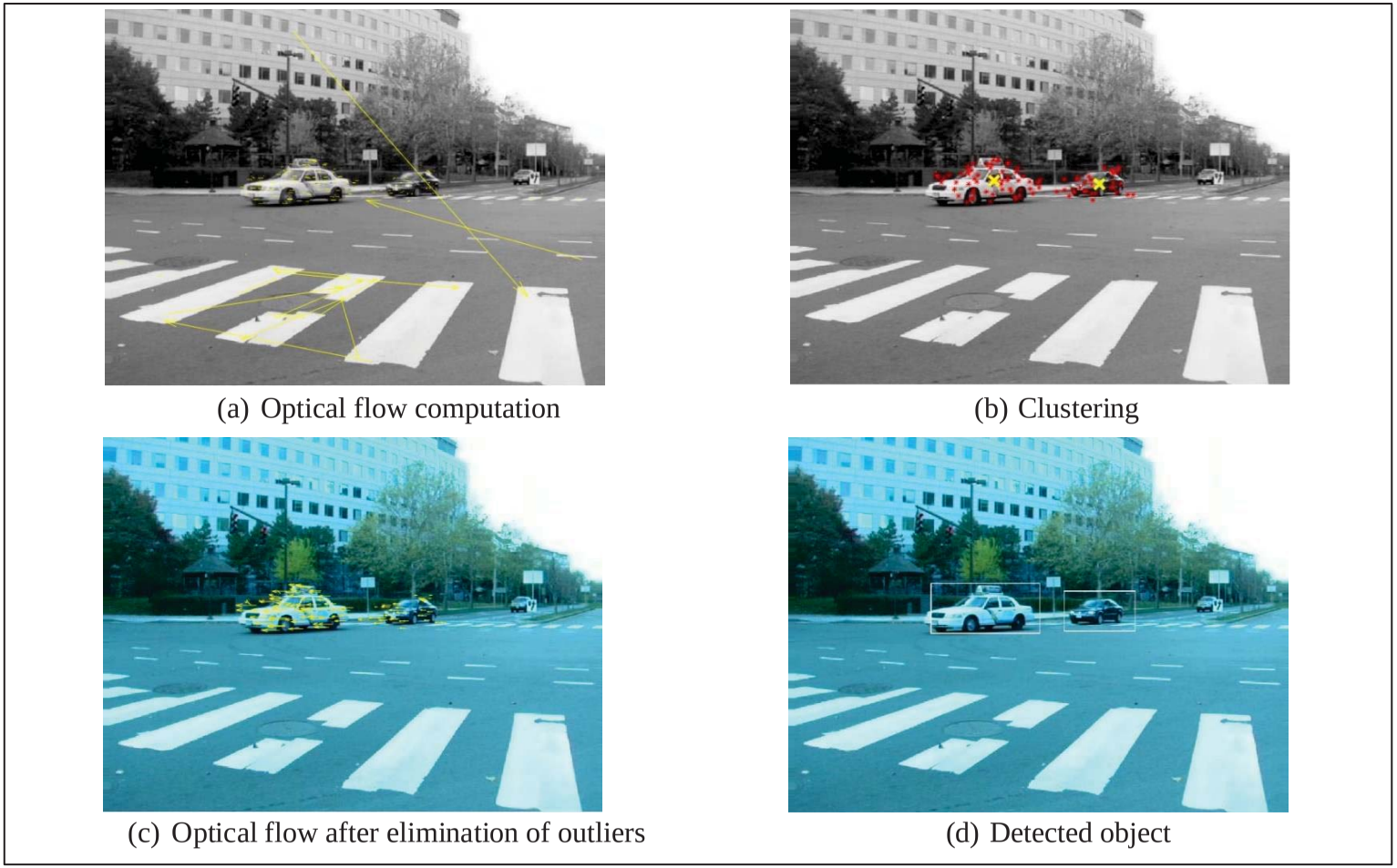
(b) Clustering (discard clusters of size less than 6)

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-14: Aerial sequence 3, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Car sequence 3** (Frames 1 and 2)



(a) Optical flow computation

(b) Clustering

(c) Optical flow after elimination of outliers

(d) Detected object

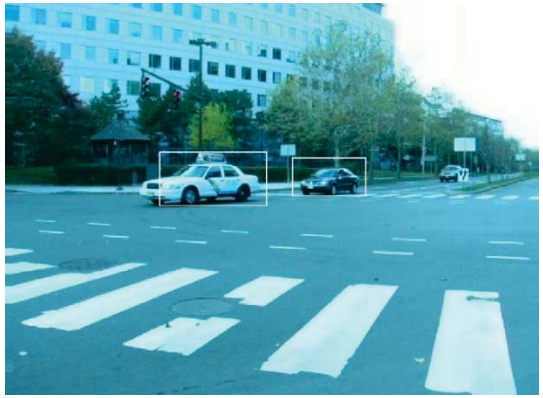
Figure 4-15: Car sequence 3, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Car sequence 3** (Frame 3 and 4)

(a) Optical flow computation

(b) Clustering (discard clusters of size less than 6)

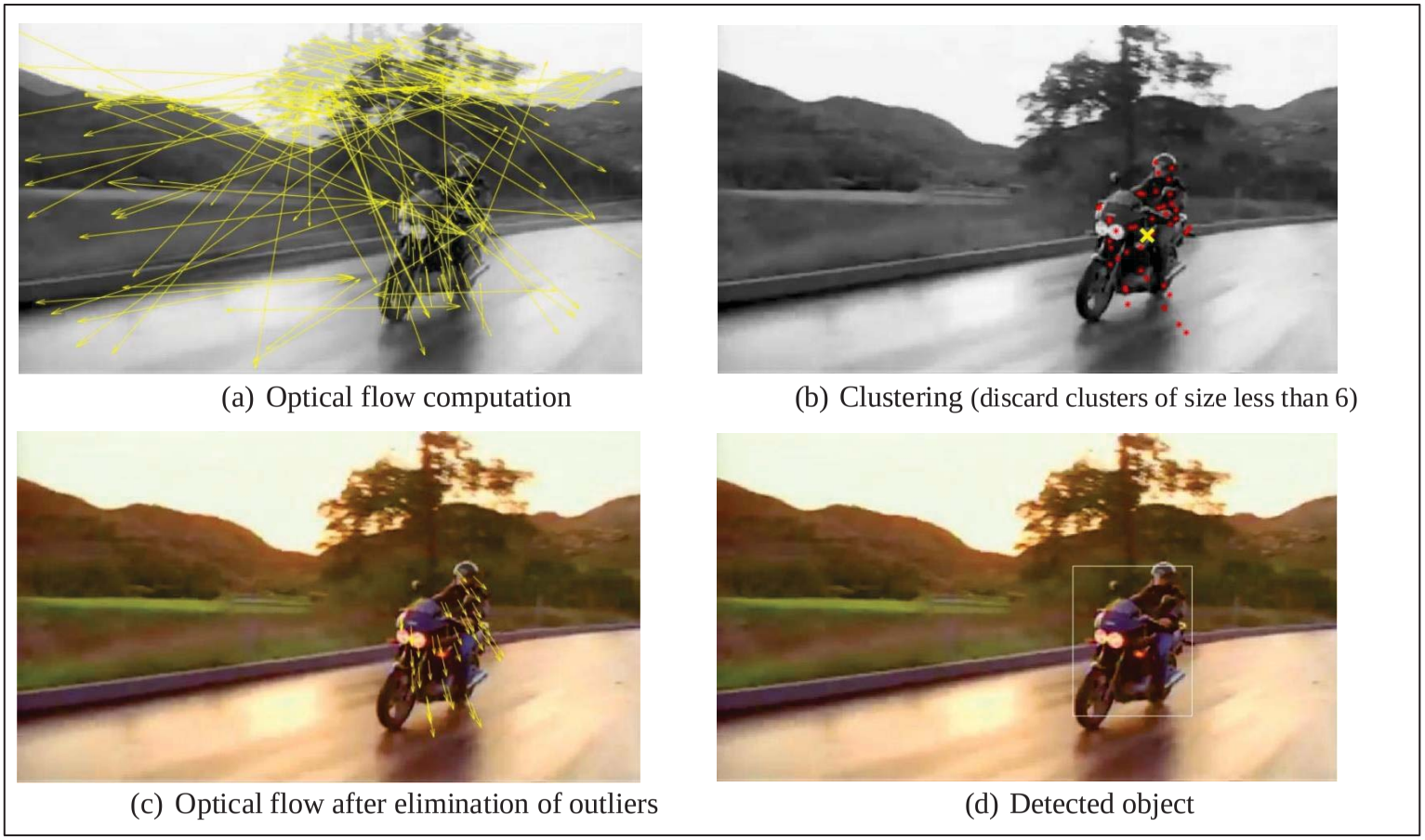
(c) Optical flow after elimination of outliers



(d) Detected object

Figure 4-16: Car sequence 3, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Target sequence** (Frames 6 and 10)



(a) Optical flow computation

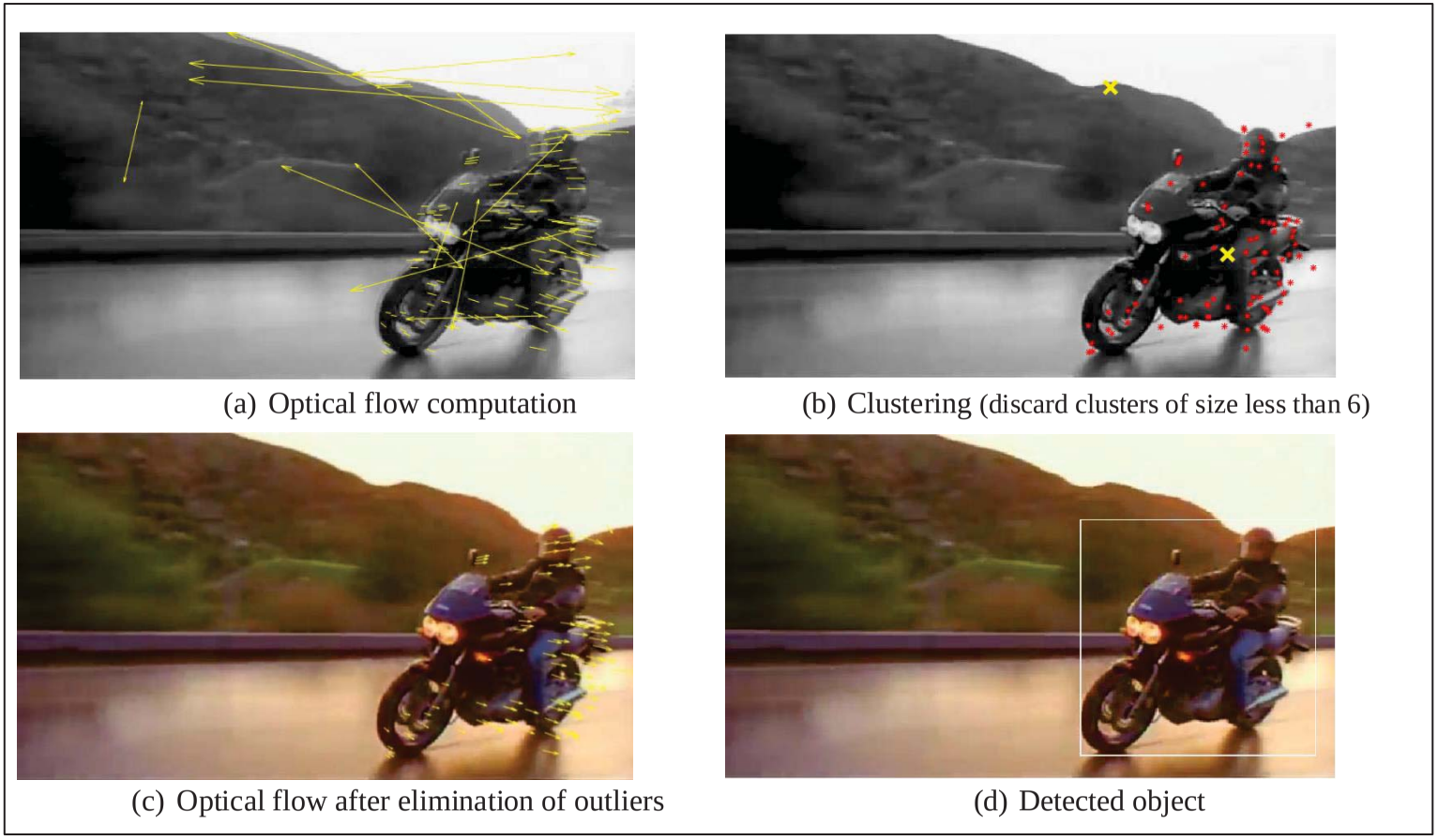
(b) Clustering (discard clusters of size less than 6)

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-17: Target sequence, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Target sequence** (Frames 35 and 39)



(a) Optical flow computation

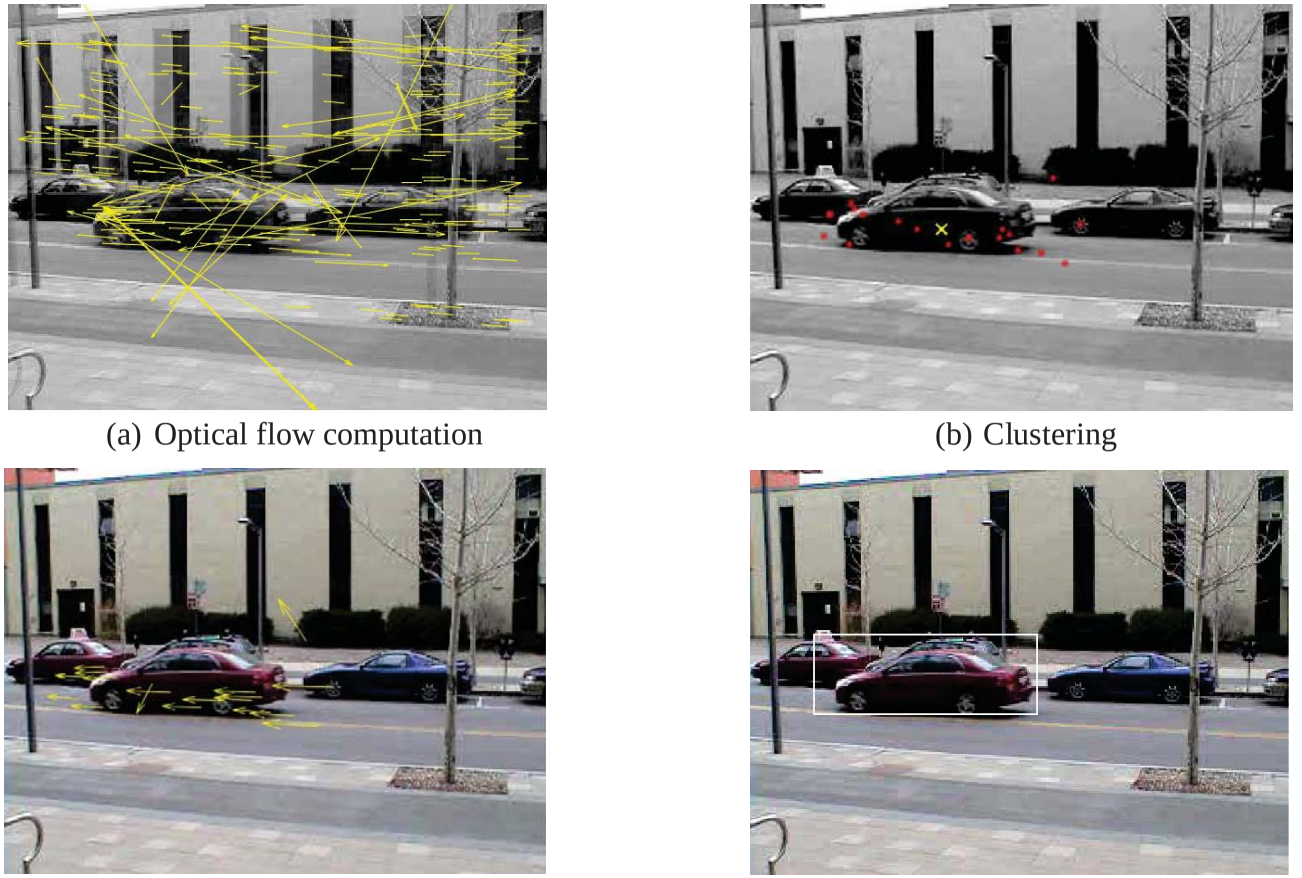
(b) Clustering (discard clusters of size less than 6)

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-18: Target sequence, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Car Frames**



(a) Optical flow computation

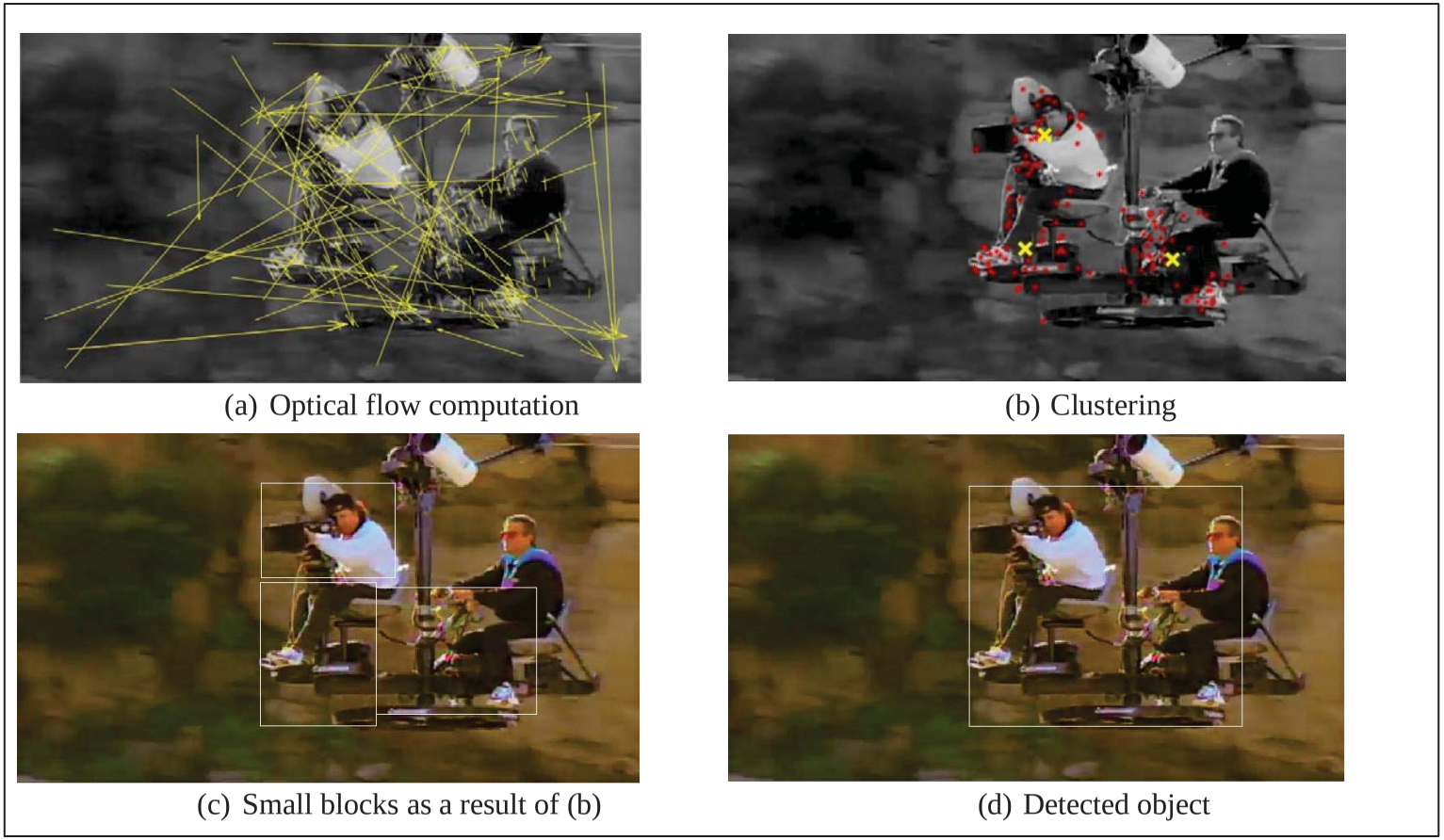
(b) Clustering

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-19: Car frames, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Shooting frames**



(a) Optical flow computation

(b) Clustering

(c) Small blocks as a result of (b)

(d) Detected object

Figure 4-20: Shooting sequence, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) small blocks detected, (d) merged blocks for detected object

**Soccer Frames**

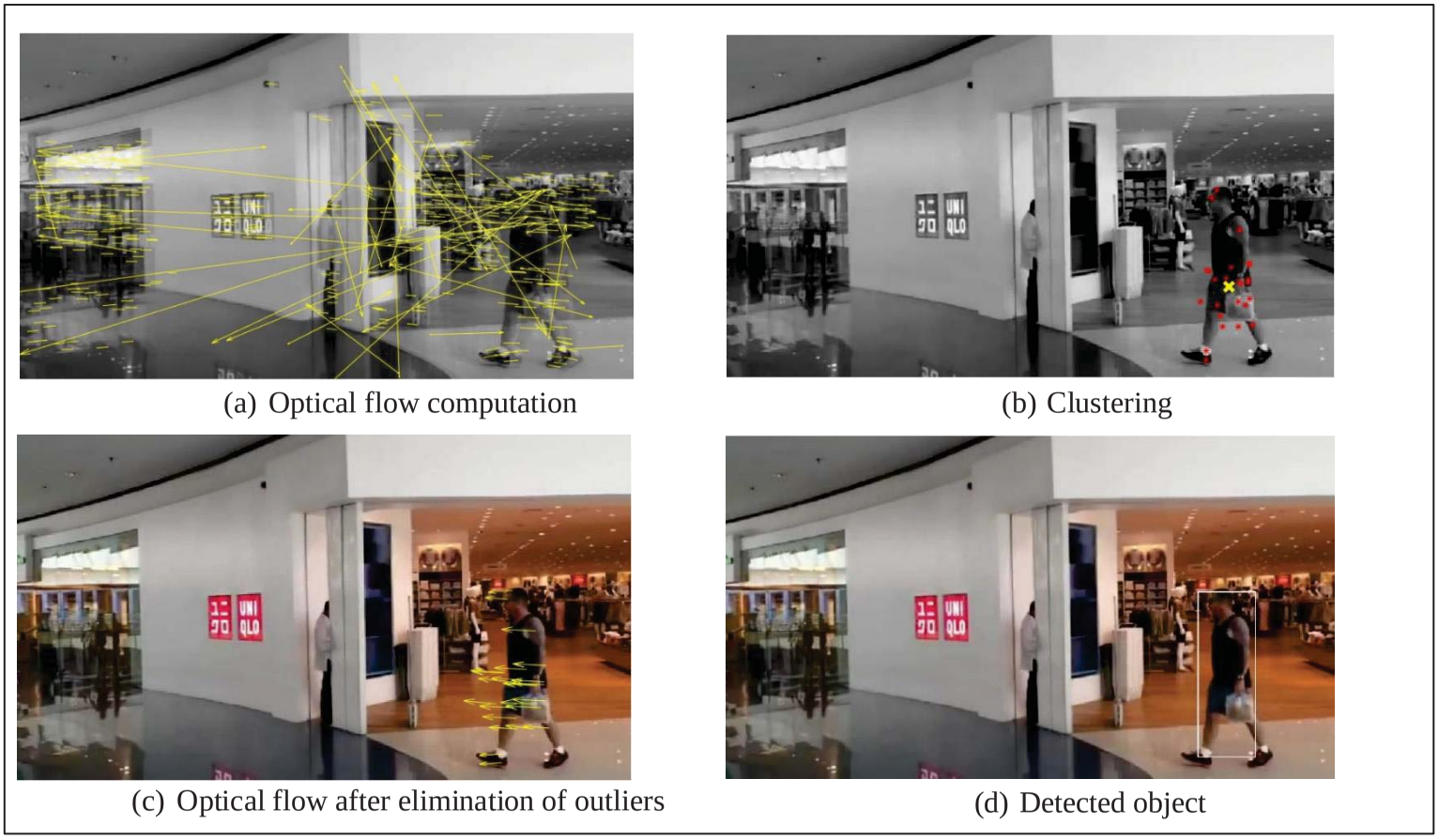
(a) Optical flow after the elimination of points

(b) Clustering using the proposed clustering algorithm

(c) Detected object

Figure 4-21: Soccer frames, (a) final motion vectors as a result of elimination of outliers, (b) clustering of points using the proposed clustering algorithm, (c) detected objects

**Indoor frames SM1** (frames 68 and 70)



(a) Optical flow computation

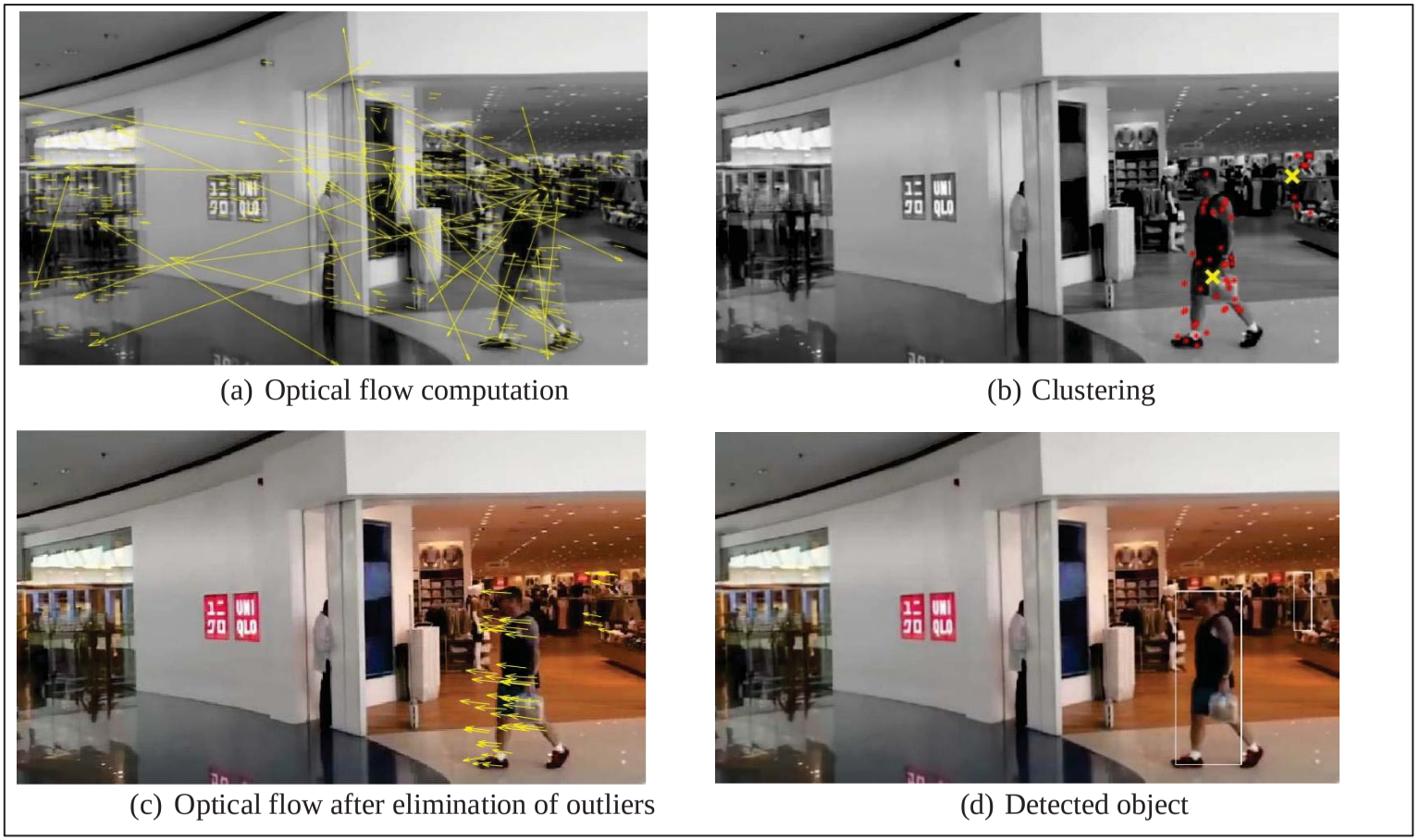
(b) Clustering

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-22: Indoor frame SM1, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Indoor frames SM1** (frames 70 and 72)



(a) Optical flow computation

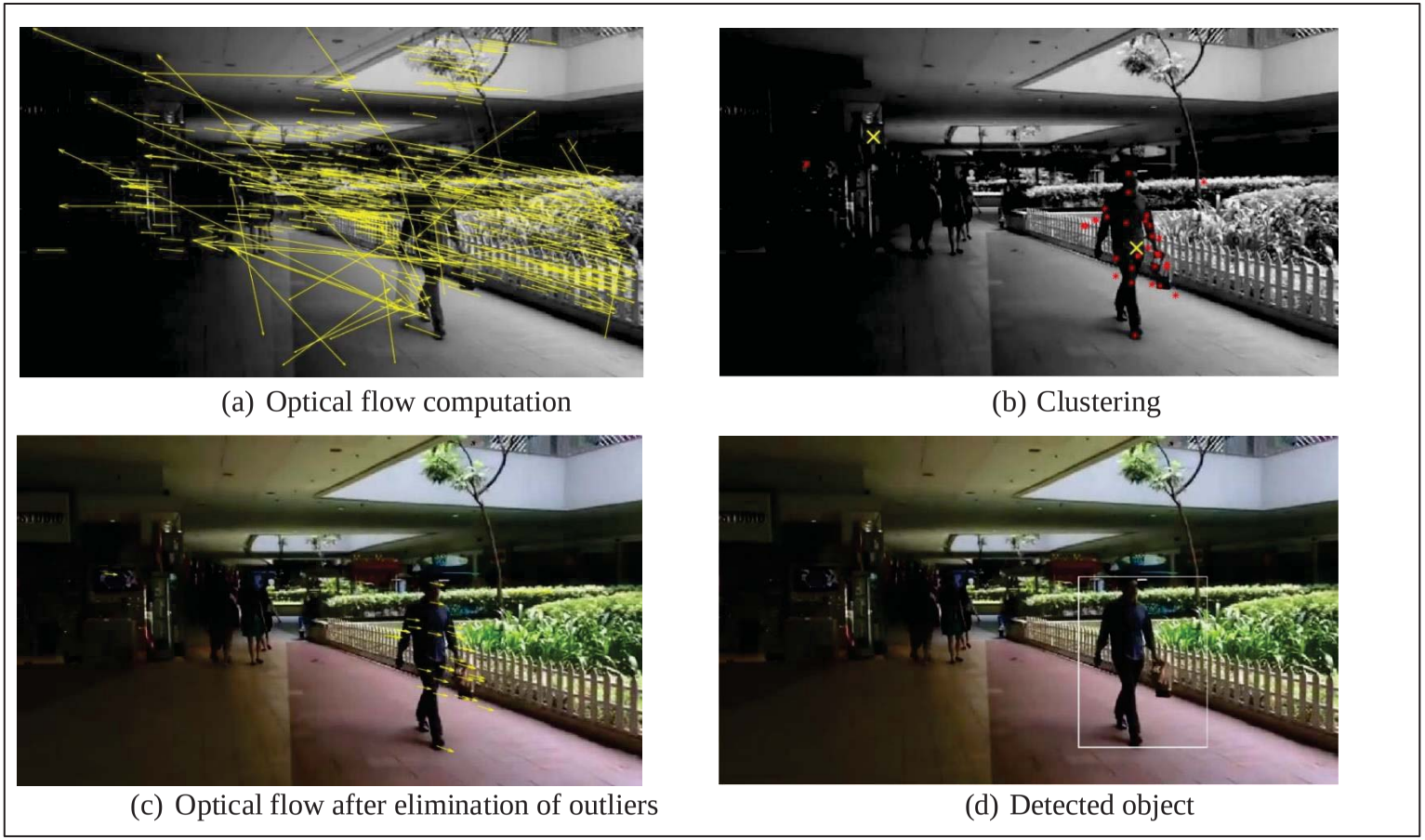
(b) Clustering

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-23: Indoor frame SM1, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Indoor Frames SM3** (Frames 22 and 26)



(a) Optical flow computation

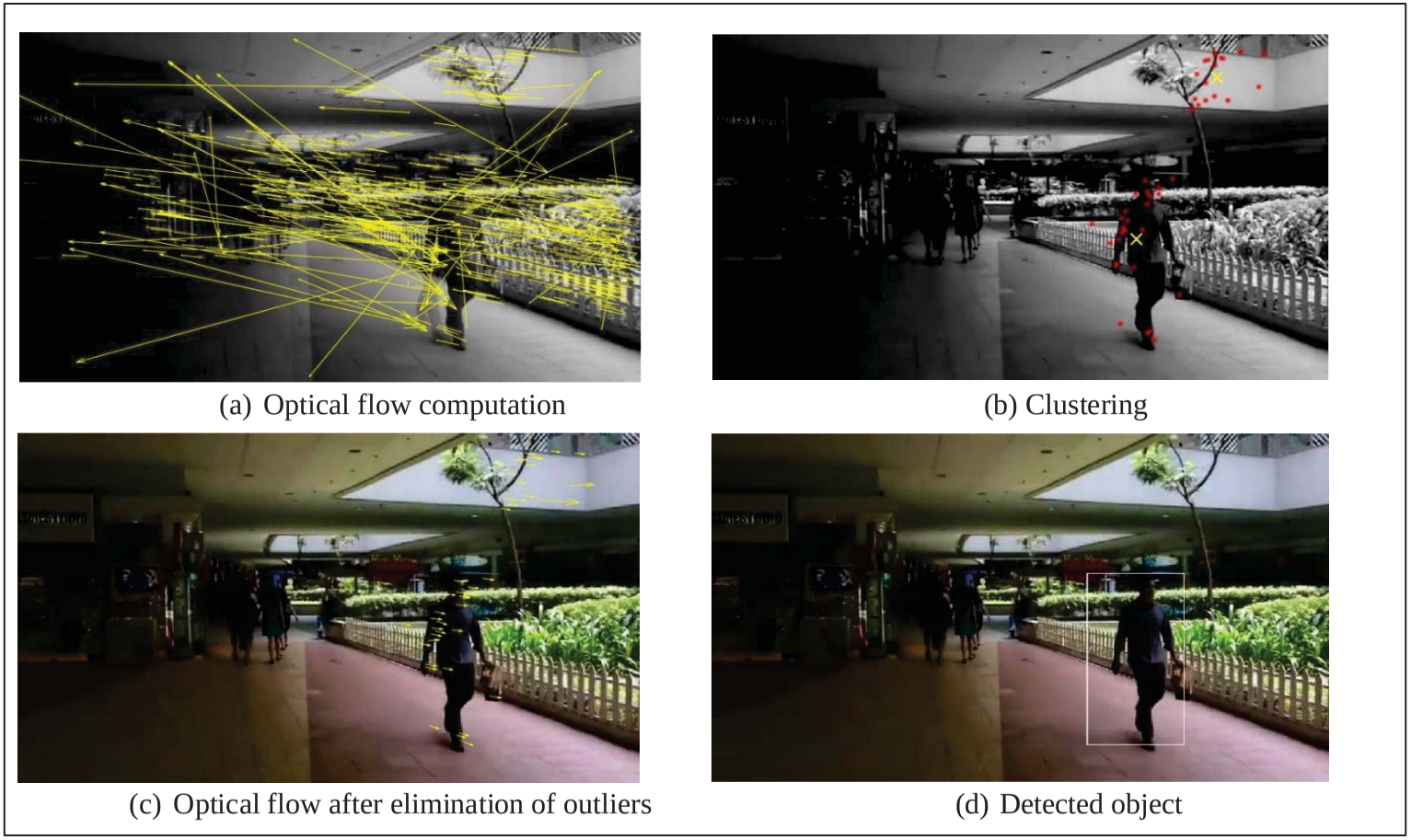
(b) Clustering

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-24: Indoor frame SM3, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

**Indoor Frames SM3** (Frames 26 and 30)



(a) Optical flow computation

(b) Clustering

(c) Optical flow after elimination of outliers

(d) Detected object

Figure 4-25: Indoor frame SM3, (a) motion vectors as a result of optical flow computation, (b) clustering of points using the proposed clustering algorithm, (c) moving object detected as a result of optical flow, (d) detected object

## 4.2 Object recognition based on Heuristics and Colour measurements

Test frames presented in this section are used to identify different colours. The identified colours are represented as binary images that highlights the corresponding colour in the test frame as white. These highlighted white areas are then put into bounding boxes of regions that are one single mass without discontinuities and their centroids are marked by a blue dot. These dots are used to display the attributes which are marked in pink. If the detected colour region is a tiny speck, it is excluded from computation. The frame displays the attributes of the environment such as grass, trees, road, pedestrian crossing, road sign, etc. The attributes, in conjunction with the shape and size of the detected objects along with the help of heuristics, results in a perceptive outcome that describes the indoor or outdoor setting and classify the object, shown in purple on top left corner of the frame.

In Table 4-1, the object recognition efficiency column relates to the efficiency of results acquired using the proposed method of heuristics and colour measurements. This efficiency portrays the performance of heuristics on various tests frames and the final outcome of recognition of the detected object.

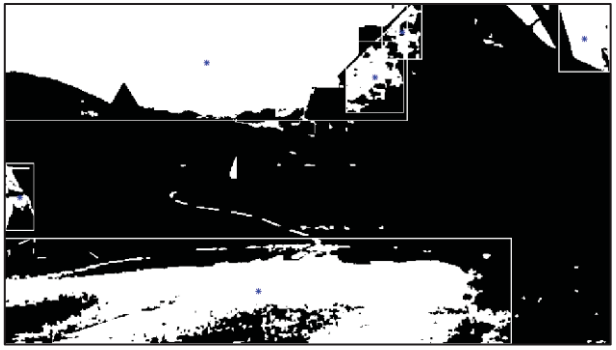
Table 4-1: Results of object detection and recognition for various test frames

| Frame Name | Frame Size | No. of SURF points detected | Computation Time (sec) | Detection efficiency (%) | Camera Shift | Camera Shift angle | Intra Frame Search | Centroid displacement limit | Object recognition efficiency (%) |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Elimination function | | Clustering function | | |
| **Outdoor frames** | | | | | | | | | |
| Lumix Bike | 360x640 | 252 | 7.79 | 100 | 4 | 45 | 101 | 131 | 100 |
| Target | 360x640 | 204 | 8.5 | 100 | 17 | 18 | 151 | 181 | 100 |
| Soccer | 360x640 | 503 | 21.9 | 80 | 15 | 67 | 114 | 144 | 90 |
| Shooting | 360x640 | 306 | 13.3 | 100 | 11 | 56 | 128 | 158 | 90 |
| car seq 1 | 480x640 | 468 | 21.44 | 100 | 4 | 76 | 106 | 136 | 100 |
| car seq 2 | 480x640 | 684 | 33.7 | 100 | 10 | 6 | 120 | 150 | 100 |
| car seq 3 | 480x640 | 467 | 19.68 | 100 | 12 | 5 | 118 | 148 | 100 |
| Aerial Seq | 270x480 | 75 | 2.11 | 95 | 7 | 0 | 90 | 120 | 100 |
| | | | | | | | | | |
| **Indoor frames** | | | | | | | | | |
| SM1 | 360x640 | 298 | 16.1 | 100 | 11 | 0 | 156 | 186 | 100 |
| SM3 | 360x640 | 564 | 24.38 | 100 | 20 | 9 | 148 | 178 | 100 |

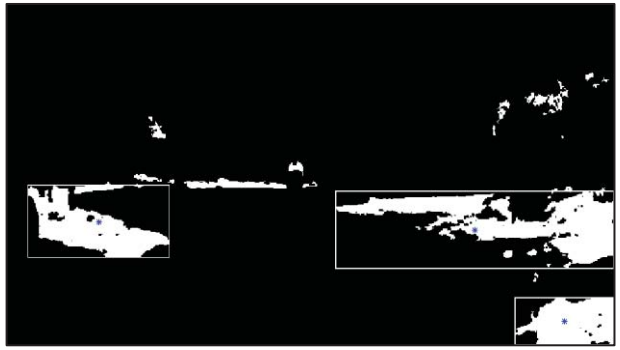
*CameraShift* and *cameraShift_angle* are the parameters needed by the elimination function.

*IntraFrameSearch* and *Centroid_disp_limit* are the parameters needed for the clustering function unique for every frame.

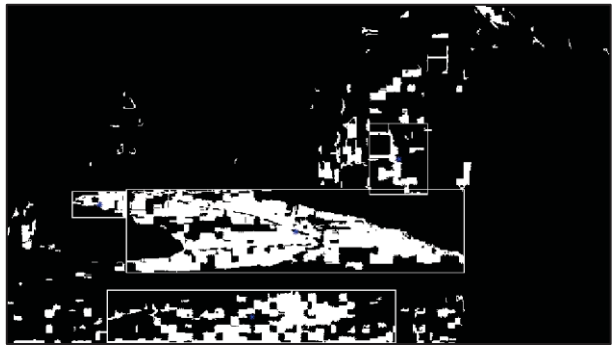
**Object recognition for Lumix Frame**



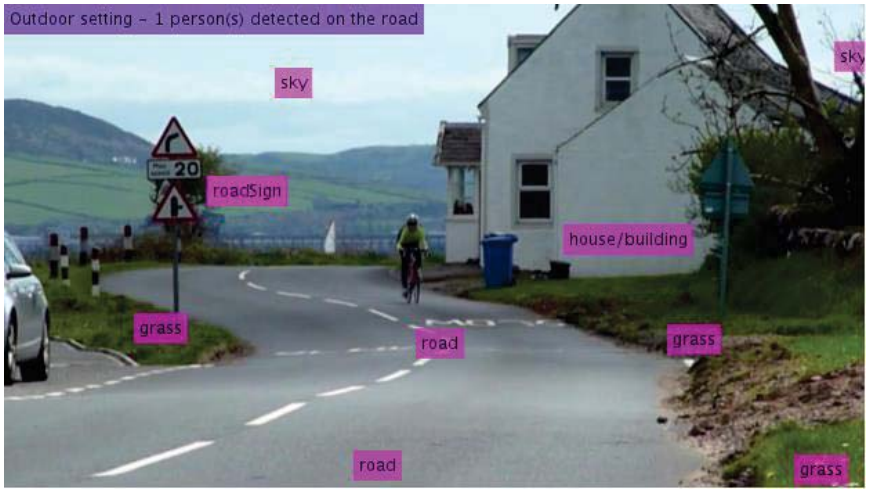
(a) White areas



(b) Green areas



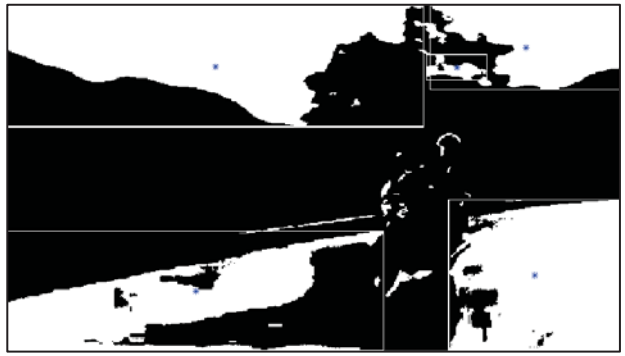
(c) Grey areas




(d) Heuristics for Lumix bike frame

Figure 4-26: Different colour areas for Lumix frame along with the recognized object in frame (c) shown in purple (top marker)

**Object recognition for Target frame**



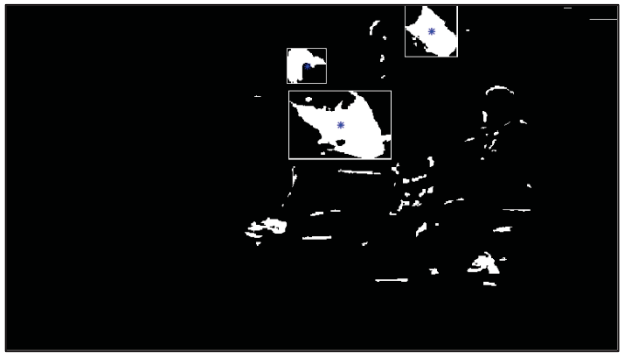
(a) White areas

(b) Green areas

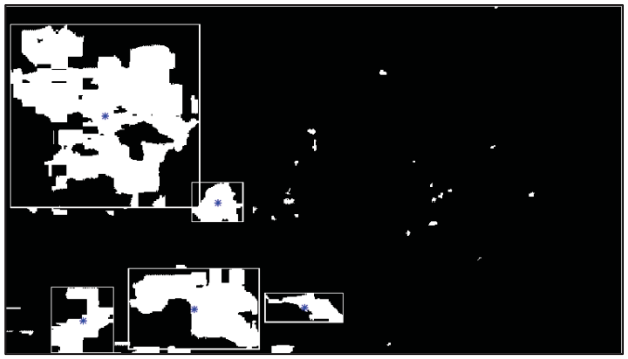

(c) Heuristics for Target frame

Figure 4-27: Different colour areas for Target frame along with the recognized object in frame (c) shown in purple (top marker)

**Object recognition for Shooting frame**



(a) White areas



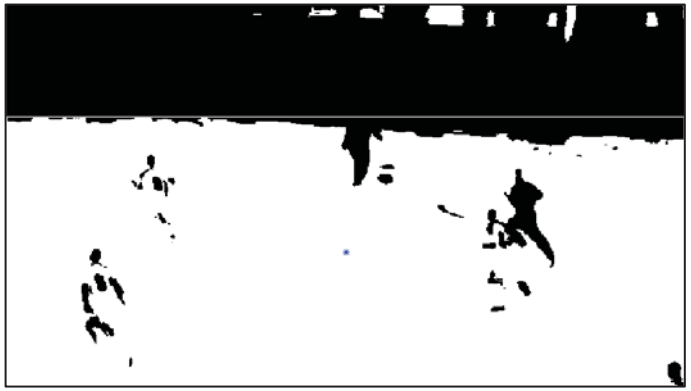
(b) Green areas


(c) Heuristics for Shooting frame

Figure 4-28: Different colour areas for Shooting frame along with the recognized object in frame (c) shown in purple (top marker)

**Object recognition for Soccer Frame**

(a) White areas

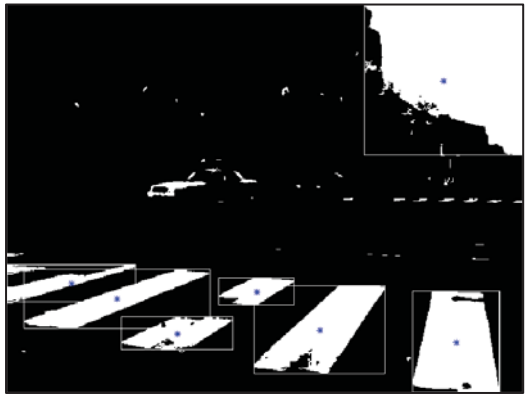


(b) Green areas

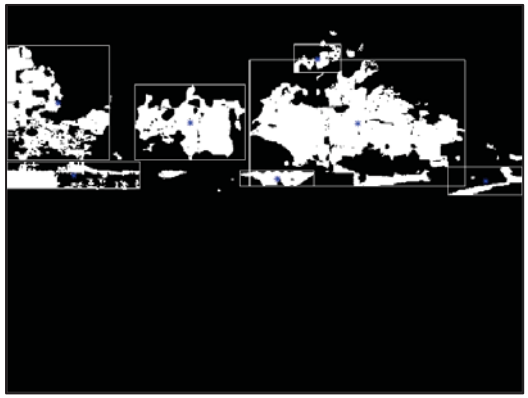

(c) Heuristics for Soccer frame

Figure 4-29: Different colour areas for Soccer frame along with the recognized objects in frame (c) shown in purple (top marker)

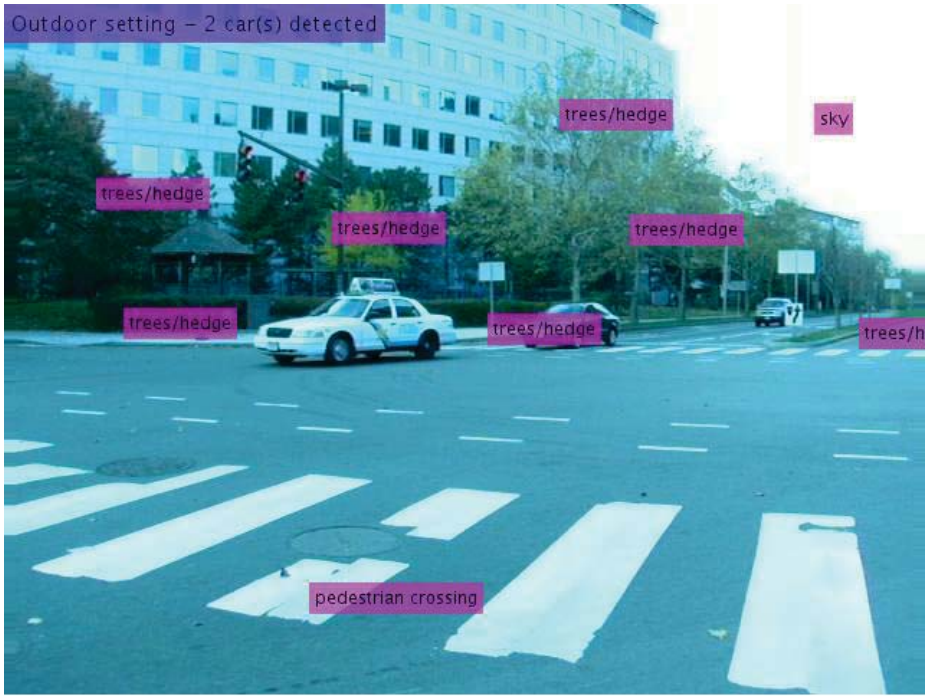
**Object recognition for Car sequence 3**



(a) White areas



(b) Green areas

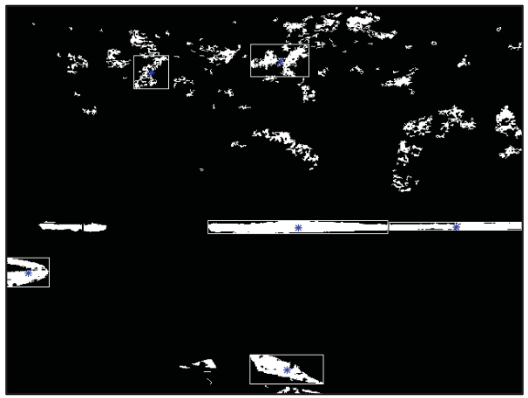



(c) Heuristics for car sequence 3 frame

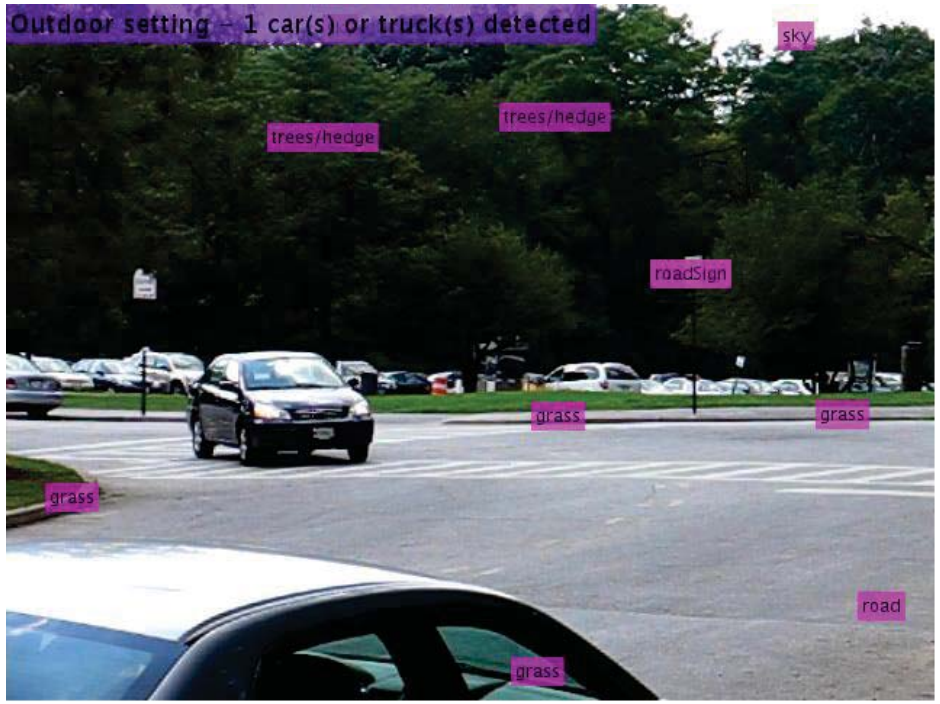
Figure 4-30: Different colour areas for Car sequence 3 frame with the recognized objects in frame (c) shown in purple (top marker)

**Object recognition for car sequence 1**

(a) White areas



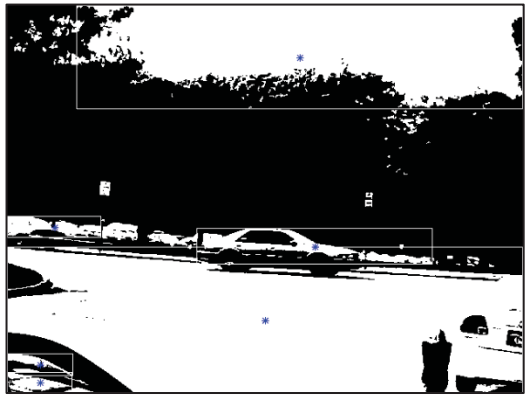
(b) Green areas



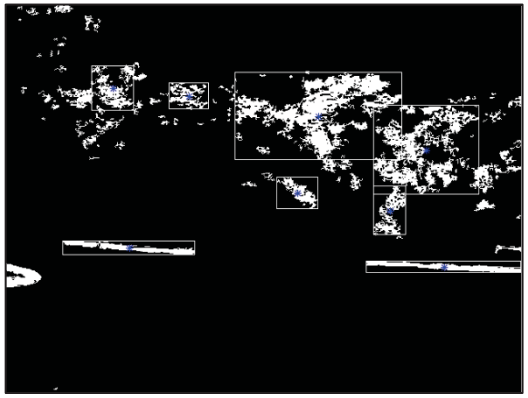

(c) Heuristics for car sequence 1 frame

Figure 4-31: Different colour areas for Car sequence 1 frame with the recognized object in frame (c) shown in purple (top marker)

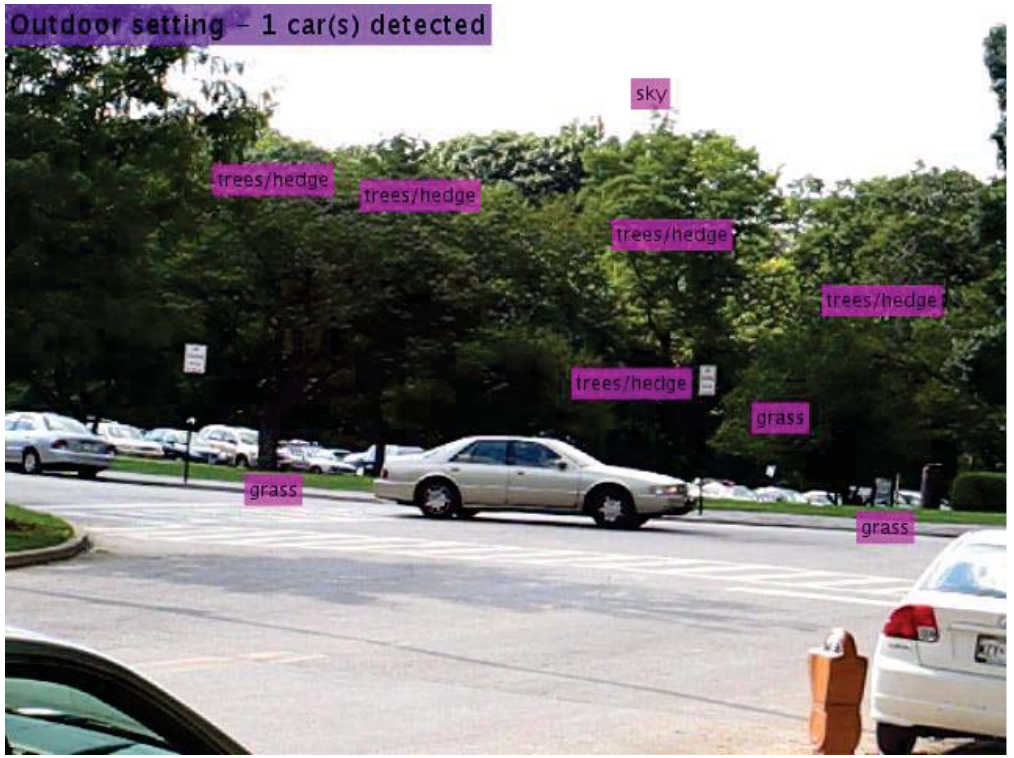
**Object recognition for car sequence 2**



(a) White areas



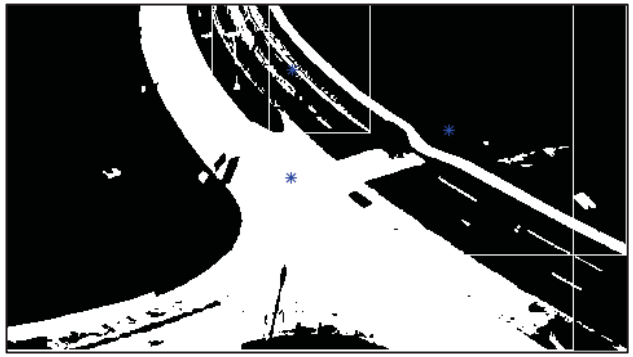
(b) Green areas



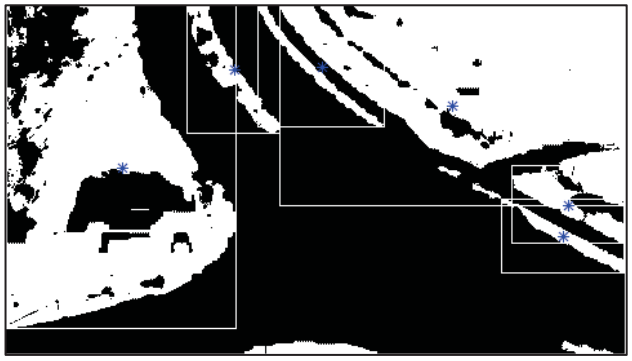

(c) Heuristics for car sequence 2 frame

Figure 4-32: Different colour areas for Car sequence 2 frame with the recognized object in frame (c) shown in purple (top marker)

**Object recognition for aerial sequence**



(a) White areas

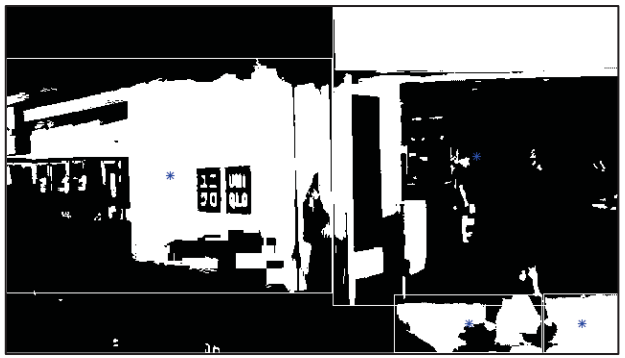


(b) Green areas


(c) Heuristics for aerial sequence frame

Figure 4-33: Different colour areas for Aerial sequence frame with the recognized object in frame (c) shown in purple (top marker)

**Object recognition for SM1 sequence frame**

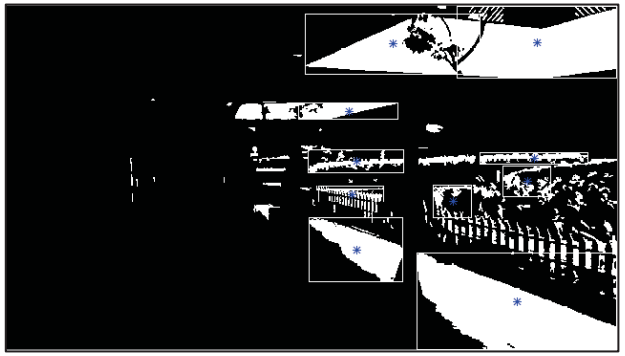


(a) White areas

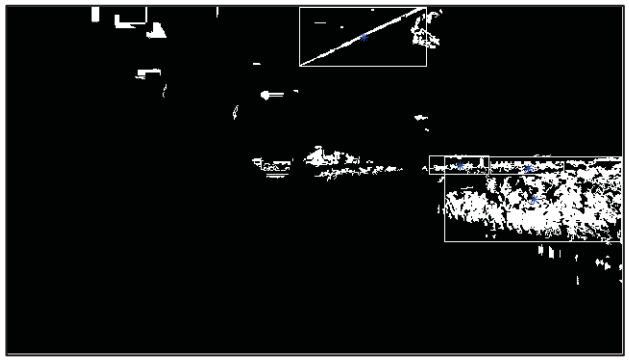

(b) Heuristics for SM1 sequence frame

Figure 4-34: Different colour areas for SM1 frame with the recognized object in frame (b) shown in purple (top marker)

**Object recognition for SM3 sequence frame**



(a) White areas



(b) Green areas


(c) Heuristics for SM3 sequence frame

Figure 4-35: Different colour areas for SM3 frame with the recognized object in frame (c) shown in purple (top marker)

Some error frames before heuristics were applied are shown in Figure 4-36. Incorrect labelling of walls and ceiling for indoor setting is shown. Frame on top shows incorrect labelling of roof. This is because heuristics was not programmed correctly. In a practical scene, the wall is between the ceiling and the floor. Heuristics combined with colour measurements detected all the white areas to be roof (ceiling) whereas, logically, the roof should exist at the top part of the frame. Similarly, the bottom frame has incorrect labelling of wall which should have been ceiling. This could be a wall if it stretched from the ceiling to the floor.

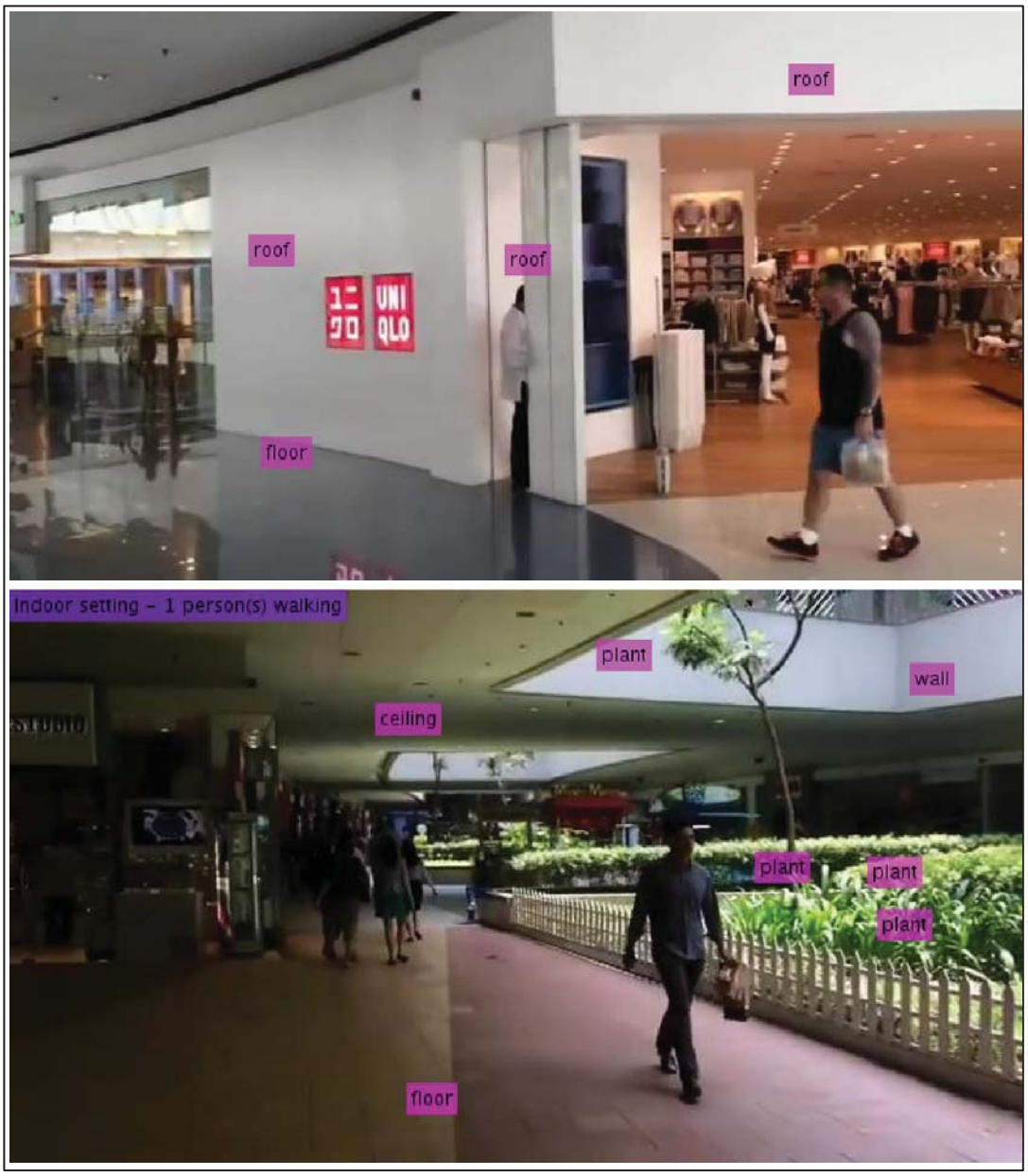
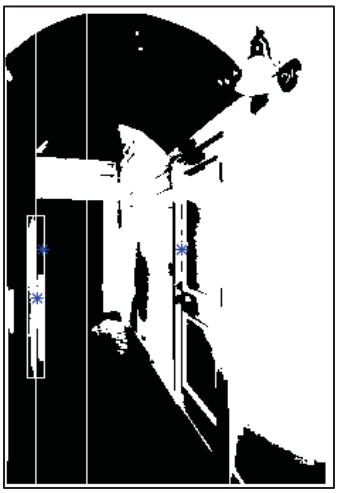


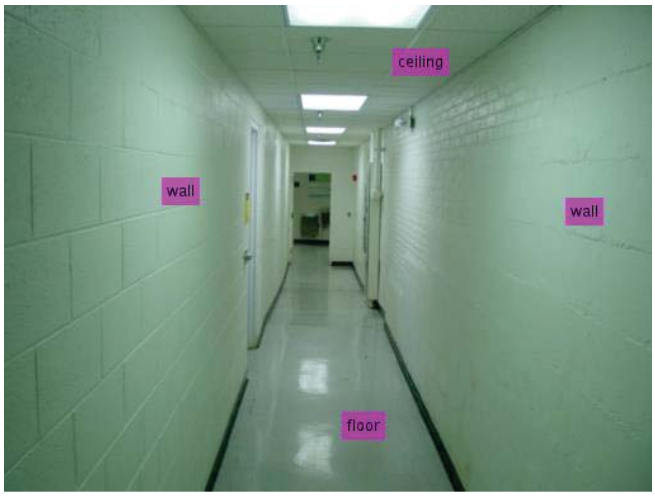

Figure 4-36: Erroneous heuristic implementation for SM1 and SM3 sequence frames

The two frames in Figure 4-36 show errors regarding correct detection of walls and ceiling in the scene. This happened due to incorrect programming of heuristics. Hence, some test images displayed in Figure 4-37, Figure 4-38 and Figure 4-39 were used to train heuristics combined with colour measurement to correctly detect basic structures of an indoor scene. Generally, if the images do not involve any tilt or rotation, the ceiling would appear at the top, followed by the walls and then the floor. We as humans, use our past experience and awareness to help us comprehend an indoor setting which must have a ceiling or some sort of shelter. Therefore, using these test images of an indoor setting, heuristics was trained to detect the ceiling, walls and the floor for scenarios if any one of them is not detected by colour, heuristics would help fill in the gaps.

(a) White areas for image 2



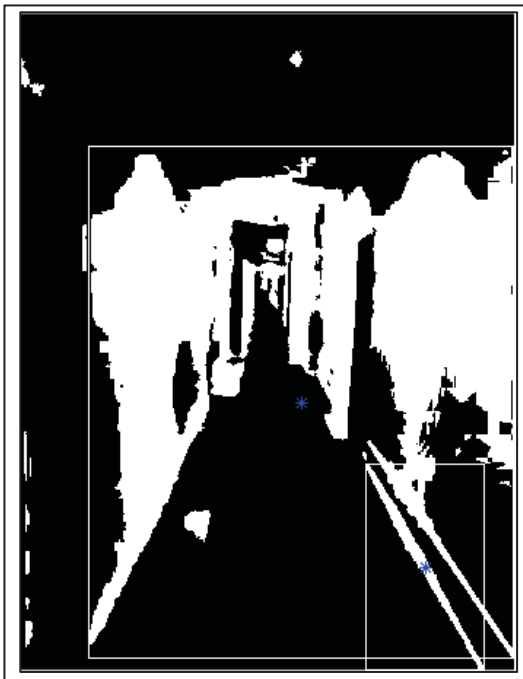
(b) White areas for image 2



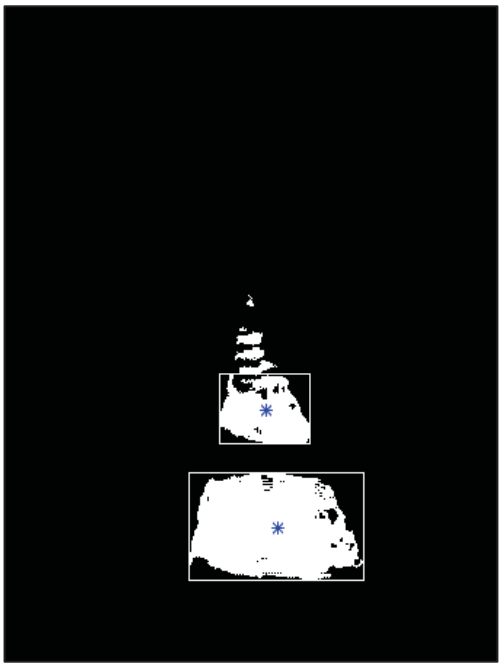

(c) Heuristics for image 1


(d) Heuristics for image 2

Figure 4-37: Heuristics for test images 1 and 2



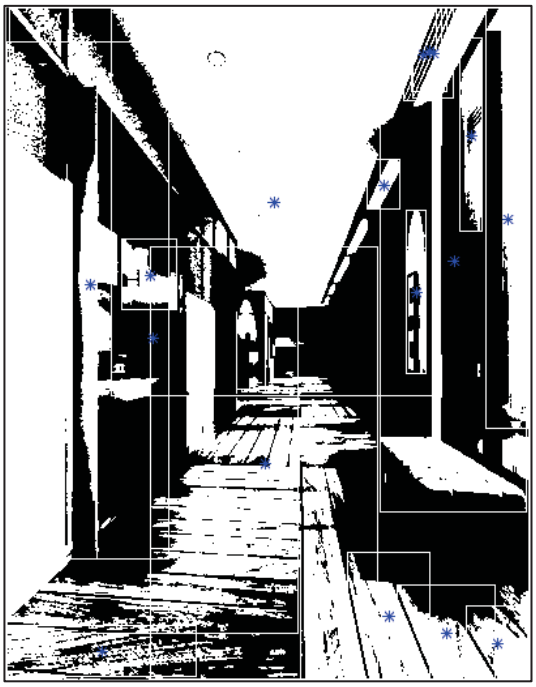
(a) White areas



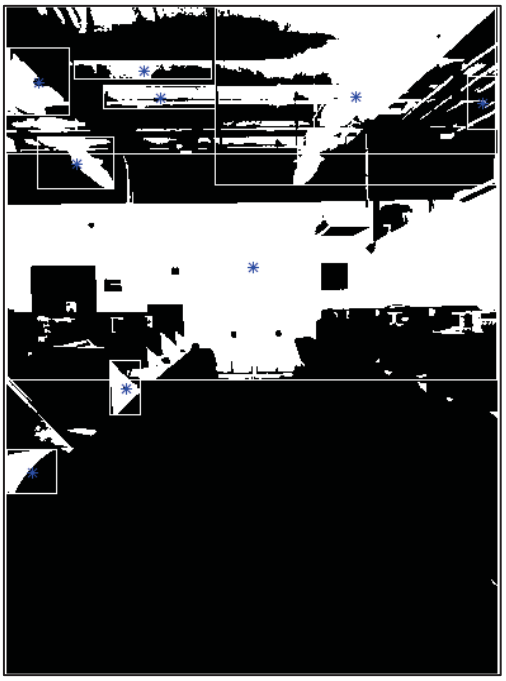
(b) Red areas


(c) Heuristics for image 3

Figure 4-38: Heuristics for test image 3

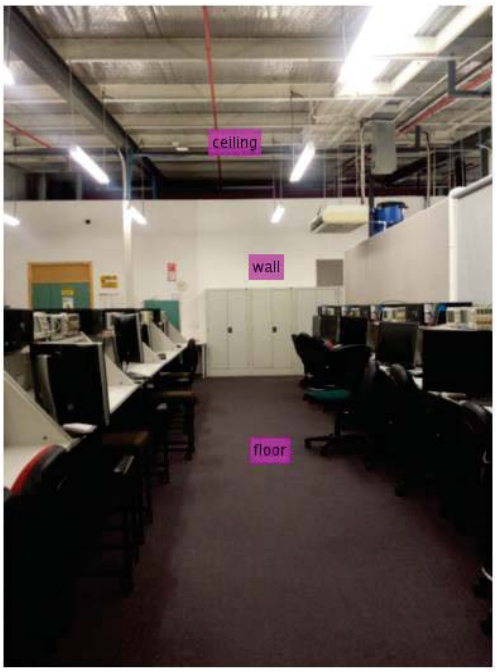


(a) White areas for image 4



(b) White areas for image 5


(c) Heuristics for image 4



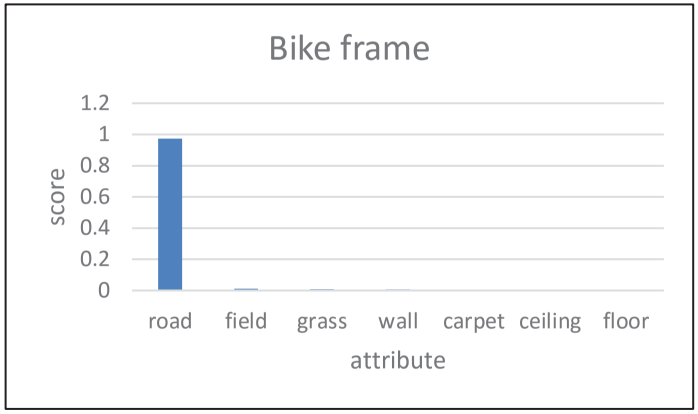

(d) Heuristics for image 5

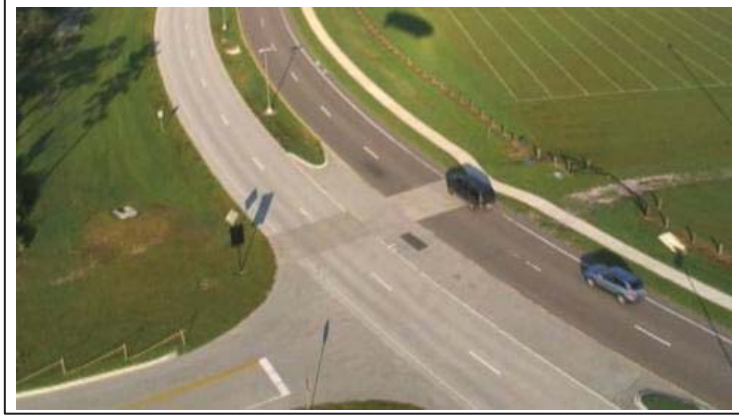
Figure 4-39: Heuristics for test images 4 and 5

## 4.3 Scene recognition using TensorFlow

The frames that have been used to test scene recognition using TensorFlow are the same frames that are used to test the colour measurements in Sections 4.1 and 4.2. Three types of results are shown in this section. Firstly, the test image is displayed. Secondly, a table depicting the attributes that were recognized in the scene using TensorFlow, along with the associated recognition score. These attributes comprised of over 80 images per attribute that were trained on TensorFlow. And lastly, a bar graph showing the attributes and their score.

| Bike frame | |
|---|---|
| attribute | score |
| road | 0.9732 |
| field | 0.01155 |
| grass | 0.00696 |
| wall | 0.00469 |
| carpet | 0.00197 |
| ceiling | 0.00095 |
| floor | 0.00068 |



| Aerial Seq | |
|---|---|
| attribute | score |
| road | 0.53614 |
| field | 0.3429 |
| ceiling | 0.04114 |
| grass | 0.03548 |
| floor | 0.02888 |
| wall | 0.01136 |
| carpet | 0.0041 |

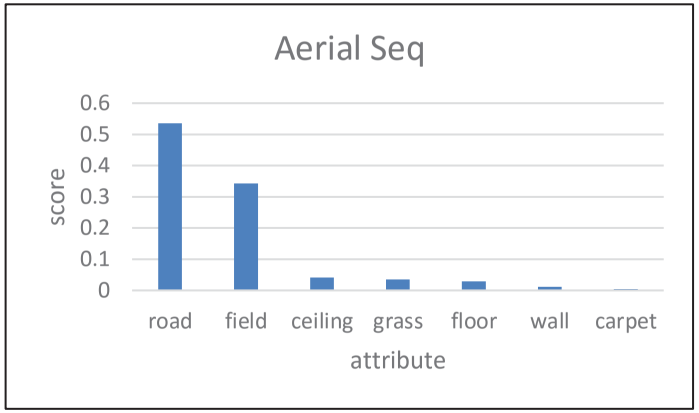
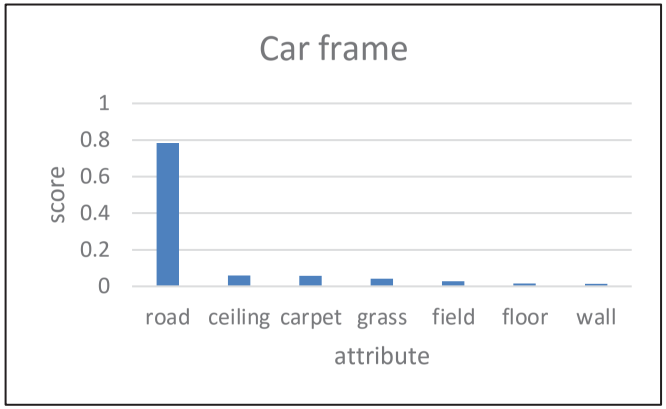



Figure 4-40: Bike frame (top) and Aerial sequence (bottom), showing score for each attribute found in the frames using TensorFlow

| Car frame | |
|---|---|
| attribute | score |
| road | 0.78415 |
| ceiling | 0.05901 |
| carpet | 0.05818 |
| grass | 0.04181 |
| field | 0.02758 |
| floor | 0.01495 |
| wall | 0.01431 |




| SM1 | |
|---|---|
| attribute | score |
| ceiling | 0.43803 |
| wall | 0.37861 |
| carpet | 0.10401 |
| floor | 0.05949 |
| field | 0.00918 |
| road | 0.00648 |
| grass | 0.0042 |

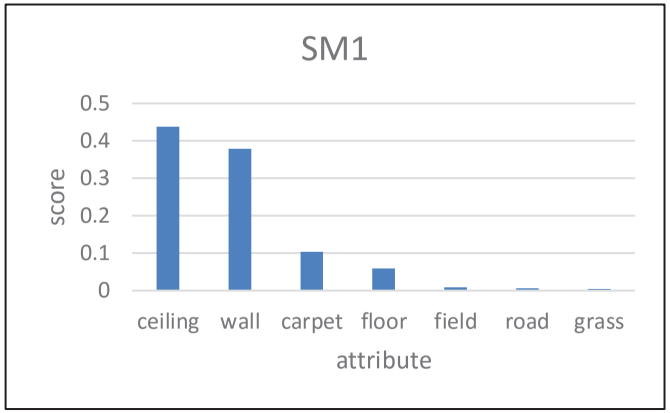
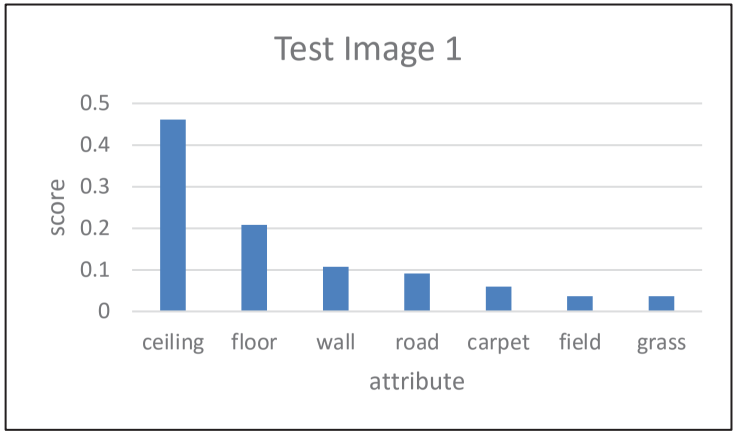



Figure 4-41: Car frame (top) and SM1 sequence (bottom), showing score for each attribute found in the frames using TensorFlow

| Test Image 1 | |
|---|---|
| attribute | score |
| ceiling | 0.46083 |
| floor | 0.20782 |
| wall | 0.10752 |
| road | 0.09116 |
| carpet | 0.05999 |
| field | 0.03637 |
| grass | 0.03631 |




| Test Image 2 | |
|---|---|
| attribute | score |
| wall | 0.47894 |
| floor | 0.18616 |
| ceiling | 0.1758 |
| carpet | 0.14015 |
| road | 0.00707 |
| field | 0.00607 |
| grass | 0.00581 |

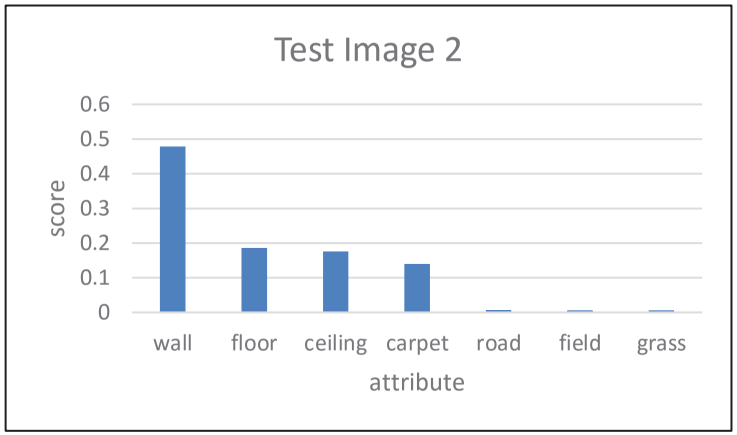



Figure 4-42 Test image 1 frame (top) and Test image 2 (bottom), showing score for each attribute found in the images using TensorFlow

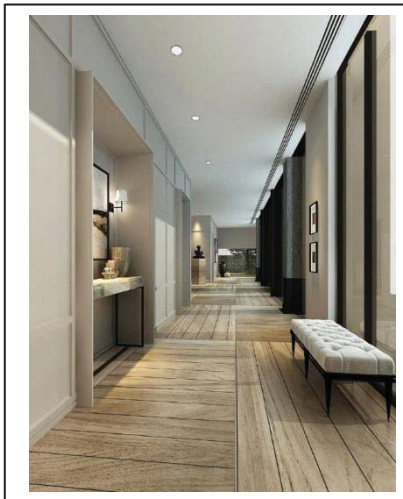
| Test Image 3 | |
|---|---|
| attribute | score |
| ceiling | 0.3937 |
| floor | 0.37248 |
| wall | 0.11033 |
| carpet | 0.10152 |
| grass | 0.00787 |
| road | 0.00759 |
| field | 0.00651 |



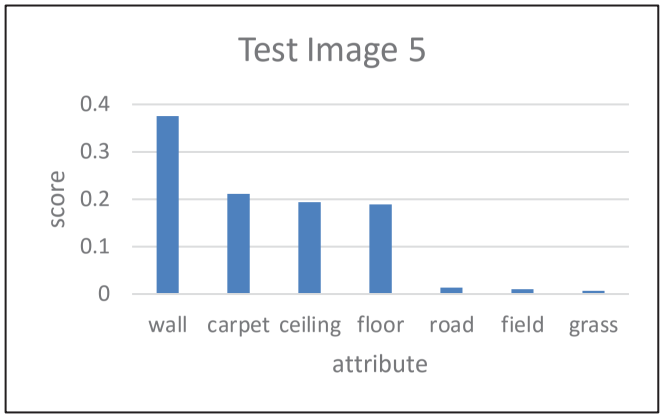
| Test Image 4 | |
|---|---|
| attribute | score |
| ceiling | 0.58921 |
| floor | 0.16647 |
| wall | 0.1033 |
| carpet | 0.09505 |
| field | 0.02173 |
| road | 0.01385 |
| grass | 0.01038 |

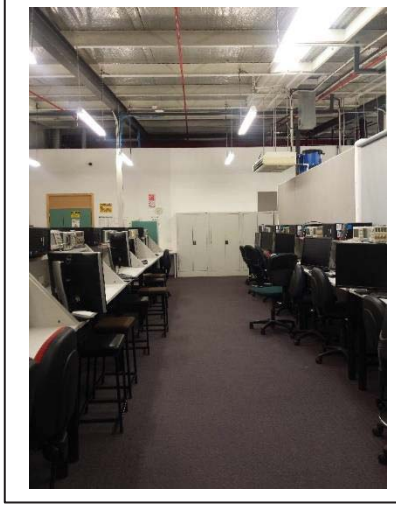


Figure 4-43: Test image 3 frame (top) and Test image 4 (bottom), showing score for each attribute found in the images using TensorFlow



| Test Image 5 | |
|---|---|
| attribute | score |
| wall | 0.37529 |
| carpet | 0.21108 |
| ceiling | 0.19408 |
| floor | 0.18931 |
| road | 0.01372 |
| field | 0.01002 |
| grass | 0.0065 |



| Test Image 6 | |
|---|---|
| attribute | score |
| ceiling | 0.41511 |
| carpet | 0.34113 |
| wall | 0.14703 |
| field | 0.03567 |
| road | 0.02696 |
| grass | 0.01858 |
| floor | 0.01552 |

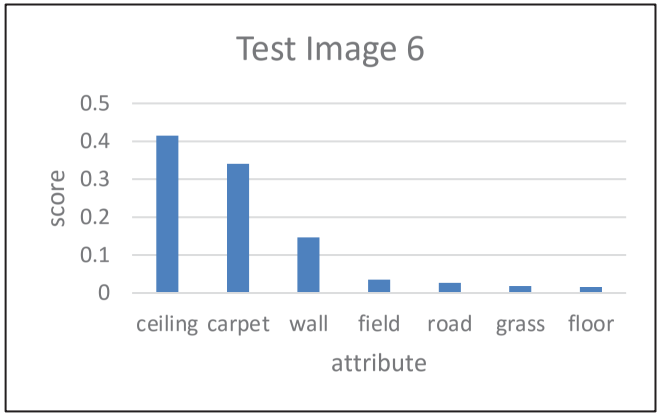



Figure 4-44: Test image 5 (top) and Test image 6 (bottom), showing score for each attribute found in the images using TensorFlow

Comparing the results obtained for colour measurements and TensorFlow:

- Computation time for TensorFlow takes an average of 2.5 seconds compared to 0.5 seconds to 1.2 seconds taken to detect colours using thresholding of HSV images.
- In total of 690 images were trained with 100 images for each attribute comprising of walls, ceiling, floor, road, grass or field, etc. with different brightness levels and viewpoints. The training images influence computation time so if there are more images, more time will be taken.
- Once the images are trained on TensorFlow, re-training them is not required. Whereas, with colour measurements, for every image or frame, the HSV conversion and thresholding process needs to be reiterated.
- Results shown in Section 4.3 reveal that TensorFlow is invariant to changes in pixel intensity levels and colour contrast as long as the trained images cater for each and every scenario, thereby increasing the efficiency of recognition. However, to attain the efficiencies, more images per attribute need to be trained but that may influence computation time for recognition of each attribute.

## 4.4 Summary of Results & Comparison with Literature

The proposed algorithm has produced some promising results. The main attributes of the results achieved in this research work are as follows:

1. The clustering algorithm groups feature points into accurate clusters 100% of the time by computing the *intraSearchRadius* and *centroid_disp_limit* for each pair of test frames. Attained moving object detection efficiency of 95%. This includes the quality the detection of object (either part of whole).
2. Achieved the highest object recognition efficiency of 97% in non-dynamic scenes
3. The method for object detection is not memory intensive because it uses two frames at a time to detect moving objects and does not involve any background modelling or motion compensation. In addition to that, no databases have been used so memory consumption has been kept to minimum.
4. The algorithm has no constraint for object detection on smooth or dynamic background. Results show successful detection of objects on both smooth and cluttered backgrounds.
5. The algorithm has been tested for real-world applications with dynamic backgrounds and also performs well in low contrast scenes.

Referring to object detection and recognition efficiencies shown in Table 4-1 that displays the performance of the proposed methods and Table 2-1 that displays the performance of results by other researchers, it can be seen that the performance of methods that are proposed in this thesis have outperformed the performance of methods that have been used in the past. The results presented in this thesis have also been compared with the results obtained by other researchers. The results are quantified in general areas of object detection and recognition such as complexity of algorithm, memory consumption, efficiency, frame size, complexity of image used and detection or recognition in indoor/outdoor scenes.

Efficiency and speed of computation depends not only on the algorithm but also on the type of images that are used. Especially for object detection, smooth or plain background would detect moving objects at a much faster rate compared to images with dynamic backgrounds. The size of the images should also be taken into account, as larger images will require longer processing times than smaller frames. One of the strong candidates [22] that was chosen in the literature review, used test

images of size 320x240 which is half the size of the frames that were used to test the proposed algorithm in this thesis. Their constraint was the detection of objects on smooth surfaces whereas, the proposed algorithm can successfully detect objects on plane or smooth backgrounds such as soccer (Figure 4-21), shooting (Figure 4-20) and target (Figure 4-17) frames. Also, research conducted in [40] was constraint to using images with dynamic background as their detection algorithm resulted in inaccurate results when using smooth background. The average efficiency for object detection in [22] was 85%, using sharp images with objects of interest in focus. Research conducted for moving object detection in, resulted in approximately 95% efficiency. The test images of size 320x155, their limitation was to use image with smooth background which eliminates the need for background computation and thus, it would not require a complex algorithm for processing such frames. Other approaches that relate to detecting moving objects on moving background such as [26], [27] and [28] have achieved 64.7%, 65.4% and 90% efficiency in detecting moving objects. However, [28] was constraint to small changes in motion of the object of interest which is not always the case in a real world application. Majority of the research works for object detection mentioned above, are programmed for outdoor applications. However, there was not enough research done for object detection in an indoor setting.

Evaluating the results for object recognition obtained in this research does not involve use of any databases or complex searching algorithms. This has been replaced by case-based reasoning to integrate heuristics and colour measurements. The perceptive approach to object recognition has achieved an efficiency of over 97% by perceiving the context of the environment solely based on colour measurements and shape of the detected object. Heuristics assist in interpreting what each colour in a certain part of the image could be. The processing time for recognition of objects range between 0.4 seconds to 1.2 seconds. Object recognition has also been approached in many ways, achieving different efficiencies such as object recognition approach in [39, 43, 45, 46] achieving an recognition accuracy of 95%, 68.4%, 92.8% and 66.6% respectively. Keeping in mind that recognition was done using trained images which makes recognition of objects much easier. Research [39] producing the an efficiency of 95% which takes between 10 to 20 seconds per image to process. Hence, a 2% increase in efficiency for object recognition using the proposed algorithm has been achieved.

# Chapter 5 Conclusion and future work

This thesis presented a novel approach to object recognition and also an improved clustering algorithm for detection of moving objects in the scene, on a non-static background. An extensive literature review indicated feeble areas, and gaps in research were identified. Four strong candidates emerged from the literature that were taken into consideration for further investigation. Main contributions of this research were:

1. An optical flow computation method using cross correlation at SURF feature points, resulting in computation faster than other optical flow methods, such as pyramidal LK and exhaustive block search.
2. Improved clustering algorithm with an adaptive search radius scheme.
3. New object recognition algorithm based on colour measurements and heuristics.

Referring to Figure 5-1, the areas highlighted in yellow depict where this research fits in the sphere of object detection and recognition. These highlighted areas display a new method of context awareness for object recognition, and an improved method for optical flow and clustering. These supplement the recognition of objects by adding context of the environment. This is unlike any other recognition algorithm that has been reviewed in the literature.

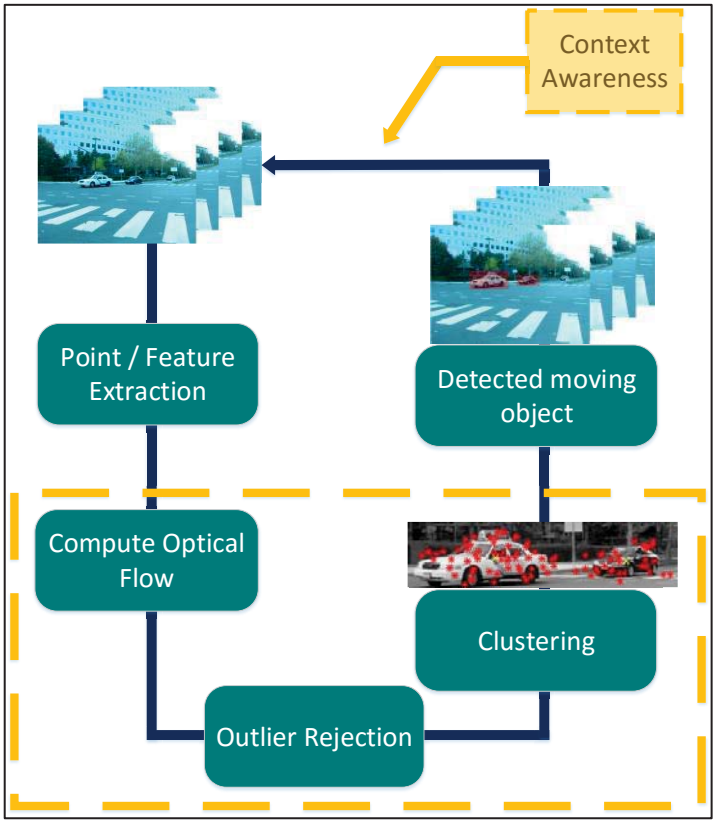



Figure 5-1: Conceptual diagram highlighting the new and improved areas

The proposed algorithm was tested for various indoor and outdoor scenes. The test results have shown the following:

- Efficiency of 95% was achieved for the detection of moving objects in a non-stationary background, with average computation time of 10-17 seconds.
- The new proposed method for object recognition resulted in recognition efficiency of 97%, with computation time of 0.5-1.2 seconds.
- The improved clustering algorithm performs accurately with variety of patterns of feature points scattered around the image, using its adaptive search radius without the user having to specify the total number of clusters.
- An accurate detection and recognition of objects without the use of computationally intensive techniques have been achieved.
- Colour measurements with thresholding performed better than TensorFlow. On average of 100 images per attribute were trained on TensorFlow for experimentation purposes. However, the scene recognition efficiency using TensorFlow as shown in Section 4.3 could produce better results if more images were trained for each attribute.

The efficiencies and computation times that were obtained in this thesis are for the setup used for this thesis and these are not conclusive.

For future work, a revised version of colour measurement algorithm would be required to detect different shades of colours which becomes slightly challenging to achieve using thresholding. Furthermore, the results for object recognition as stated in Table 4-1, were programmed for specific scenarios and could have been biased. Therefore, in the future, a general and more flexible approach can be instated. For faster computation and testing in the real world, the algorithm would need to be deployed in the field, on a dedicated graphics processing unit (GPU). One of the applications could be with colour and frame, and robots like Junior [67] where only static objects in the scene are regarded. But what if the object is non-static? It can be observed from Section 4.3 that, when using Tensorflow, scene recognition still does not improve much when compared to colour measurements. The proposed algorithm was still able to detect parts of the scene such as ceilings, walls, grass and roads at a faster rate without the memory intensive database storing technique.

# References


[1] R. Szeliski, *Computer vision: algorithms and applications*: Springer Science & Business Media, 2010.

[2] B. Dataset. Available: http://lmb.informatik.uni-freiburg.de/resources/datasets/

[3] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, *et al.*, "Towards robust automatic traffic scene analysis in real-time," in *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, 1994, pp. 126-131.

[4] Q. Zang and R. Klette, "Object classification and tracking in video surveillance," in *International Conference on Computer Analysis of Images and Patterns*, 2003, pp. 198-205.

[5] K. Patwardhan, G. Sapiro, and V. Morellas, "Robust foreground detection in video using pixel layers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, pp. 746-751, 2008.

[6] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, pp. 780-785, 1997.

[7] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, 1999.

[8] A. Elgammal, R. Duraiswami, and L. S. Davis, "Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking," *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, pp. 1499-1504, 2003.

[9] M. Seki, H. Fujiwara, and K. Sumi, "A robust background subtraction method for changing background," in *Proceedings Fifth IEEE Workshop on Applications of Computer Vision*, 2000, pp. 207-213.

[10] S. Huwer and H. Niemann, "Adaptive change detection for real-time surveillance applications," in *Visual Surveillance, 2000. Proceedings. Third IEEE International Workshop on*, 2000, pp. 37-46.

[11] L. Li, W. Huang, I. Y. Gu, and Q. Tian, "Foreground object detection in changing background based on color co-occurrence statistics," in *Applications of Computer Vision, 2002.(WACV 2002). Proceedings. Sixth IEEE Workshop on*, 2002, pp. 269-274.

[12] S. C. Pohlig, "Spatial-temporal detection of electro-optic moving targets," *IEEE transactions on aerospace and electronic systems*, vol. 31, pp. 608-616, 1995.

[13] Y. Sheikh, O. Javed, and T. Kanade, "Background subtraction for freely moving cameras," in *Computer Vision, 2009 IEEE 12th International Conference on*, 2009, pp. 1219-1225.

[14] X. Ye, J. Yang, X. Sun, K. Li, C. Hou, and Y. Wang, "Foreground–Background Separation From Video Clips via Motion-Assisted Matrix Restoration," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, pp. 1721-1734, 2015.

[15] E. Hayman and J.-O. Eklundh, "Statistical background subtraction for a mobile observer," in *ICCV*, 2003.

[16] A. Fathi, X. Ren, and J. M. Rehg, "Learning to recognize objects in egocentric activities," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference On*, 2011, pp. 3281-3288.
[17] Y. J. Lee, "Foreground focus: Finding meaningful features in unlabeled images," 2008.
[18] N. Y. Khan, B. McCane, and G. Wyvill, "SIFT and SURF performance evaluation against various image deformations on benchmark dataset," in *Digital Image Computing Techniques and Applications (DICTA), 2011 International Conference on*, 2011, pp. 501-506.
[19] A. Bugeau and P. Pérez, "Detection and segmentation of moving objects in highly dynamic scenes," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1-8.
[20] A. Viswanath, R. K. Behera, V. Senthamilarasu, and K. Kutty, "Background Modelling from a Moving Camera," *Procedia Computer Science*, vol. 58, pp. 289-296, 2015.
[21] Y. Jin, L. Tao, H. Di, N. I. Rao, and G. Xu, "Background modeling from a free-moving camera by multi-layer homography algorithm," in *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, 2008, pp. 1572-1575.
[22] J. Kim, X. Wang, H. Wang, C. Zhu, and D. Kim, "Fast moving object detection with non-stationary background," *Multimedia tools and applications*, vol. 67, pp. 311-335, 2013.
[23] K. Lal and K. M. Arif, "Feature extraction for moving object detection in a non-stationary background," in *Mechatronic and Embedded Systems and Applications (MESA), 2016 12th IEEE/ASME International Conference on*, 2016, pp. 1-6.
[24] M. P. Patel and S. K. Parmar, "Moving object detection with moving background using optic flow," in *Recent Advances and Innovations in Engineering (ICRAIE), 2014*, 2014, pp. 1-6.
[25] S.-W. Sun, Y.-C. F. Wang, F. Huang, and H.-Y. M. Liao, "Moving foreground object detection via robust SIFT trajectories," *Journal of Visual Communication and Image Representation*, vol. 24, pp. 232-243, 2013.
[26] Y. Lu, X. Xu, Y. Dai, and B. Zheng, "An Adaptive Object Detection Scope Algorithm Based on SIFT," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2012 4th International Conference on*, 2012, pp. 100-103.
[27] H. Uemura, S. Ishikawa, and K. Mikolajczyk, "Feature Tracking and Motion Compensation for Action Recognition," in *BMVC*, 2008, pp. 1-10.
[28] C.-C. Wong, W.-C. Siu, S. Barnes, and P. Jennings, "Low relative speed moving vehicle detection using motion vectors and generic line features," in *2015 IEEE International Conference on Consumer Electronics (ICCE)*, 2015, pp. 208-209.
[29] C. Rabe, U. Franke, and S. Gehrig, "Fast detection of moving objects in complex scenarios," in *2007 IEEE Intelligent Vehicles Symposium*, 2007, pp. 398-403.
[30] Y. Chen, R. hua Zhang, and L. Shang, "A novel method of object detection from a moving camera based on image matching and frame coupling," *PloS one*, vol. 9, p. e109809, 2014.
[31] K. Yun and J. Y. Choi, "Robust and fast moving object detection in a non-stationary camera via foreground probability based sampling," in *Image*

*Processing (ICIP), 2015 IEEE International Conference on*, 2015, pp. 4897-4901.

[32] Y. Zhao, M. Casares, and S. Velipasalar, "Continuous background update and object detection with non-static cameras," in *Advanced Video and Signal Based Surveillance, 2008. AVSS'08. IEEE Fifth International Conference on*, 2008, pp. 309-316.

[33] G. Tzanidou, I. Zafar, and E. A. Edirisinghe, "Carried object detection in videos using color information," *IEEE Transactions on Information Forensics and Security*, vol. 8, pp. 1620-1631, 2013.

[34] M. Lin and X. Xu, "Multiple vehicle visual tracking from a moving vehicle," in *Intelligent Systems Design and Applications, 2006. ISDA'06. Sixth International Conference on*, 2006, pp. 373-378.

[35] T. Chen, R. Wang, B. Dai, D. Liu, and J. Song, "Likelihood-field-model-based dynamic vehicle detection and tracking for self-driving," *IEEE Transactions on Intelligent Transportation Systems,* vol. 17, pp. 3142-3158, 2016.

[36] T. B. Moeslund and E. Granum, "A survey of computer vision-based human motion capture," *Computer vision and image understanding,* vol. 81, pp. 231-268, 2001.

[37] X. Li, W. Hu, C. Shen, Z. Zhang, A. Dick, and A. V. D. Hengel, "A survey of appearance models in visual object tracking," *ACM transactions on Intelligent Systems and Technology (TIST),* vol. 4, p. 58, 2013.

[38] L. Ballan, M. Bertini, A. Del Bimbo, L. Seidenari, and G. Serra, "Event detection and recognition for semantic annotation of video," *Multimedia Tools and Applications,* vol. 51, pp. 279-302, 2011.

[39] A. Diplaros, T. Gevers, and I. Patras, "Color-shape context for object recognition."

[40] Y. Jie, C. Xiaomin, G. Pengfei, and X. Zhonglong, "A new traffic light detection and recognition algorithm for electronic travel aid," in *Intelligent Control and Information Processing (ICICIP), 2013 Fourth International Conference on*, 2013, pp. 644-648.

[41] M. Liang, M. Yuan, X. Hu, J. Li, and H. Liu, "Traffic sign detection by ROI extraction and histogram features-based recognition," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, 2013, pp. 1-8.

[42] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin, "Context-based vision system for place and object recognition," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, 2003, pp. 273-280.

[43] A. Rabinovich, A. Vedaldi, C. Galleguillos, E. Wiewiora, and S. Belongie, "Objects in Context," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1-8.

[44] C. L. Zitnick, J. Sun, R. Szeliski, and S. Winder, "Object instance recognition using triplets of feature symbols," 2007.

[45] A. Uçar, Y. Demir, and C. Güzeliş, "Moving towards in object recognition with deep learning for autonomous driving applications," in *2016 International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)*, 2016, pp. 1-5.

[46] N. M. Ali, S. W. Jun, M. S. Karis, M. M. Ghazaly, and M. S. M. Aras, "Object classification and recognition using Bag-of-Words (BoW) model," in *Signal Processing & Its Applications (CSPA), 2016 IEEE 12th International Colloquium on*, 2016, pp. 216-220.

[47] J. W. Howarth, H. H. Bakker, and R. C. Flemmer, "Feature-based Object Recognition," in *Autonomous Robots and Agents, 2009. ICARA 2009. 4th International Conference on*, 2009, pp. 375-379.
[48] D.-N. Ta, W.-C. Chen, N. Gelfand, and K. Pulli, "Surftrac: Efficient tracking and continuous object recognition using local feature descriptors," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009, pp. 2937-2944.
[49] T. S. Y.Ramadevi, B.Poornima, B.Kalyani "Segmentation and Object Recognition using Edge Detection Techniques," *International Journal of Computer Science & Information Technology*, vol. 2, pp. 153-162, 2010.
[50] M. Prasad, A. Zisserman, A. Fitzgibbon, M. P. Kumar, and P. H. Torr, "Learning class-specific edges for object detection and segmentation," in *Computer Vision, Graphics and Image Processing*, ed: Springer, 2006, pp. 94-105.
[51] K. Sekiyama, K. Watanabe, M. Rizki, and T. Fukuda, "Event recognition using object-motion context," in *Micro-NanoMechatronics and Human Science (MHS), 2010 International Symposium on*, 2010, pp. 477-482.
[52] H. Dheemanth, "Object Recognition Using Eigen Values," *International Journal of Innovative Research and Development|| ISSN 2278–0211*, vol. 3, 2014.
[53] M. Vidal-Naquet and S. Ullman, "Object Recognition with Informative Features and Linear Classification."
[54] E. Jauregi, E. Lazkano, and B. Sierra, "Object recognition using region detection and feature extraction."
[55] R. G. Wijnhoven and P. de With, "Experiments with patch-based object classification," in *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, 2007, pp. 105-110.
[56] A. Guan and Z. Peng, "An Improved Approach for Vehicles Detection and Tracking," in *ITS Telecommunications Proceedings, 2006 6th International Conference on*, 2006, pp. 306-309.
[57] K. T. Hong, J. H. Shim, and Y. I. Cho, "Moving Objects Tracking Method using Spatial Projection in Intelligent Video Traffic Surveillance System," *Journal of Korean Institute of Intelligent Systems*, vol. 25, pp. 35-41, 2015.
[58] Y. Tang, C. Zhang, R. Gu, P. Li, and B. Yang, "Vehicle detection and recognition for intelligent traffic surveillance system," *Multimedia tools and applications*, pp. 1-16, 2015.
[59] C. Harris and M. Stephens, "A combined corner and edge detector," 1988.
[60] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91-110, 2004.
[61] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, *et al.*, "A comparison of affine region detectors," *International journal of computer vision*, vol. 65, pp. 43-72, 2005.
[62] Y. Lee, K. Wampler, G. Bernstein, J. Popović, and Z. Popović, "Motion fields for interactive character locomotion," in *ACM Transactions on Graphics (TOG)*, 2010, p. 138.
[63] C. Rabe, U. Franke, and R. Koch, "Dense 3D motion field estimation from a moving observer in real time," in *Smart Mobile In-Vehicle Systems*, ed: Springer, 2014, pp. 19-34.
[64] S. J. Russell and P. Norvig, "Artificial Intelligence (A Modern Approach)," ed: Prentice Hall, 2010.

[65] (31 March). *GoogleBlog*. Available: https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html
[66] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467,* 2016.
[67] R. Flemmer. (1 April). *Junior Robot*. Available: juniorenterprises.co.nz