Robust Multi-Object Tracking for an Autonomous Vehicle Testbed

Michael Finn

Supervisors: Dr. Tim French
Dr. Ghulam Mubashar Hassan

School of Electrical, Electronic and Computer Engineering
University of Western Australia

This dissertation is submitted for the degree of
Master of Professional Engineering

Word count: 7818

21st October 2018

Abstract

In 2017, a team of UWA students developed a hardware-in-the-loop autonomous vehicle testbed to bridge the gap between simulation and real-world trials. The testbed was designed to be highly interactive and support a variety of dynamic traffic scenarios for testing and teaching purposes. To track the objects in the testbed, a vision system was created that used hue-based detection techniques to identify the state of each car. This method had several limitations, however, which prevented the testbed from supporting the full scope of its intended applications. The purpose of this thesis was to design an improved vision system that overcame the limitations of its predecessor. To achieve this, only tracking methods that preserved object state memory were explored, as it was determined that the stateless detection approach employed by the previous system was the primary source of its deficiencies. The research focused primarily on adapting well-established single-object tracking methods to a multi-object context, while also concentrating on less complex trackers to ensure that the limited computational resources of the testbed were not exceeded. Several of the resulting methods met the extended validation criteria and were found to be highly effective in terms of both performance and accuracy.

Table of contents

List of figures

List of tables

Section 1

Introduction

The advent of autonomous vehicles has stimulated a predominantly enthusiastic response from the global community, with most countries attempting to adapt their infrastructure as fast as possible to reap the rewards of the emerging technology [1]. The domain remains largely untested, however, and promises of speed and convenience battle public apprehension and moral concerns [2]. As real-world trials are costly [3], and virtual simulations can be too abstract to be intuitively appreciated, there arises a need for readily accessible testbeds that can bridge the gap between the prohibitively expensive and the purely theoretical.

One such testbed was developed by UWA students, with the view that it would facilitate learning and experimentation for all members of the public. In the testbed, ZenWheels Micro Cars are used in place of real cars (see Figure 1.1), and their driving area is observed by an overhead camera.

Figure 1.1: A ZenWheels Micro Car (with coin for scale).

The camera frames were analysed by bespoke computer vision software to provide state information to the virtual agents directing the cars. Since the Micro Cars are all identical apart from their colour, the vision software used a hue-based technique to determine their positions. The hue signature of each car would become distorted by the projected scene, however, and so background subtraction was used to filter out the projected hue layer. While this method was sufficient for tracking the cars under specific conditions, it meant that the testbed could only operate successfully with uniquely coloured cars and a known, static background. As the testbed was originally designed to support dynamic traffic

scenarios which require changing, unknown scenes, this method of state estimation was unsuitable and ultimately prevented it from achieving the full scope of its intended potential. The following research explored several hue-independent tracking methods in order to develop a method that was capable of supporting multiple identical objects simultaneously, in the presence of stochastic hue and illumination variance; dynamic backgrounds. As the tracking software had to run in real-time on a Raspberry Pi 3, the research emphasises simple, efficient methods over more powerful, but computationally expensive approaches.

Section 2 of this paper provides a review of the existing vision software, general object tracking and successful tracking approaches from literature. Section 3 details the goals of the project, and the specific requirements of the final system. Section 4 reports the research performed, methods explored and discoveries made. Section 5 presents the results of the experiments performed to test the efficacy of the chosen tracking methods. Section 6 uses the results from the previous section to analyse the extent to which each method met the system requirements. Section 7 details the conclusions of the project and examines the potential for future study.

Section 2

Literature Review

2.1 Existing Testbed Software

The field of computer vision describes a broad collection of processes that often draw from many different areas to achieve a final result. As such, there is typically no 'best' approach to a problem; solutions are highly context dependent and may work well in certain conditions and not at all in others [4]. In development of the autonomous vehicle testbed, Hurst explored several methods to detect the cars in real-time, including hue segmentation and saliency detection, before they found that the most effective approach was background subtraction followed by histogram comparison [5], demonstrated in Figure 2.1.

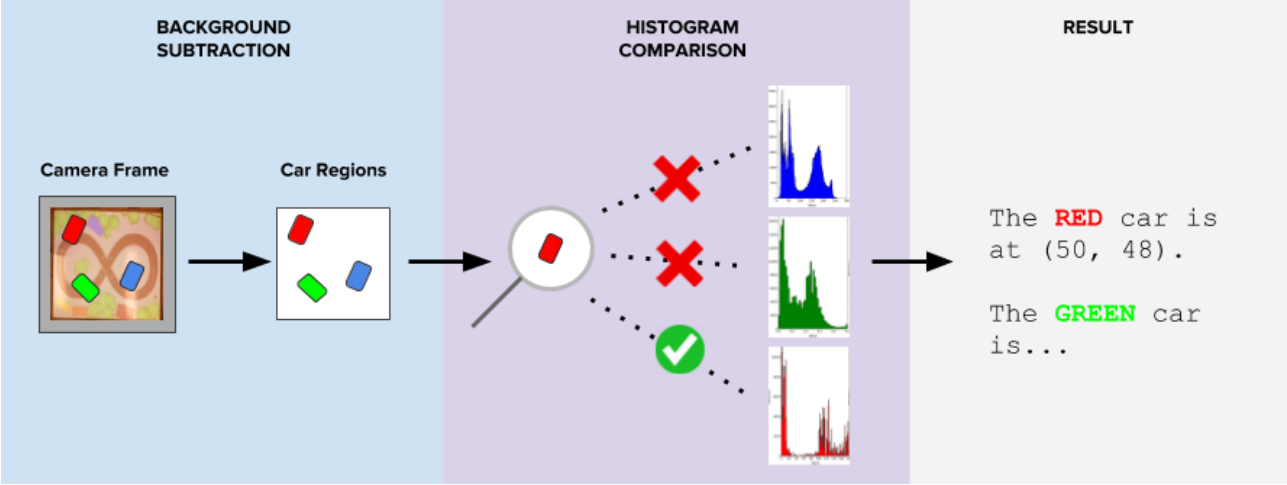

Figure 2.1: The method used by Hurst.

This methodology was resource-efficient but had several drawbacks, foremost among them that it required a static, predefined background, and that it was highly sensitive to minor changes in the cars' hues [5]. Since the testbed is designed to project an image on top of the cars, if the cars are driving in a coloured environment, their hues will certainly change. However, without being able to rely on hue,

there is nothing else that physically differentiates the cars, as their shapes and sizes are identical [6]. In this way, Hurst's vision software was limited by its focus on static object detection techniques, which only consider memoryless features [7].

2.2 General Object Tracking

Object tracking is a more advanced approach that seeks to follow an object over time, rather than considering data from a single instant or frame [8]. By following the object, it preserves the object's unique identity [8], which is especially applicable in this project where the objects are not physically differentiable to a reliable degree. It builds on object detection techniques, while also incorporating information regarding the object's spatio-temporal context, i.e. where and when it is [9]. The general process is to first select object features, such as edges, colour, optical flow etc. which differentiate the object from its background, use these features with object detection methods to locate the features in the image, then employ object tracking methods to identify the object [9]. Most approaches to tracking involve pre-defining an object of interest, then attempting to localise it in the following images, however there exist more general algorithms that can track unknown objects [10]. One of the main issues faced by tracking algorithms is object occlusion, where one object obstructs the view of another [9], however as the testbed uses an overhead camera the project will not be required to account for this. Another common issue is the correspondence problem, i.e. determining whether differences between frames are due to movements in the world or movements of the camera [9], but again the fixed nature of the camera eliminates this difficulty. Object detection, which seeks to localise an object in an image, is usually more resource-intensive than object tracking, which typically uses likelihood-based inference to predict the future position of the object [11]. To optimise a tracking system, detection may only be performed once every few frames, while tracking uses the gathered data to make an educated guess on every other frame [12].

There are three primary categories of object tracking approaches: point-based, kernel-based and silhouette-based [9]. In point-based tracking, objects are represented as points, and tracking is performed by evaluating their state in terms of position and motion [13]. There are several assumptions that a point-based tracking system may use to reduce the search space during correspondence, the process of matching an object in one frame to an object in the next [8], thus increasing speed. These include assuming that an object's location in one frame will be nearby its location in the next, an object's velocity cannot exceed a certain upper limit or change too quickly, and that the object's shape does not deform over time [9]. These assumptions are all appropriate in the context of this project, as the cars will only be able to move in a predictable, limited manner, and their shapes are rigid. Point tracking is suitable for use with objects that occupy a small region in the frame, and is invariant to changes in illumination. However, it requires a separate mechanism to detect the objects per frame, and has difficulty with occlusion [9]. Kernel-based tracking methods involve defining the region of

the image occupied by an object of interest using a primitive object such as a rectangle or ellipse, and then using its motion model (trajectory) to predict its likely next position [14]. An advantage of kernel tracking is that object detection, which is relatively resource-intensive compared to tracking, only needs to be performed once when the tracking begins [11].
Silhouette tracking is similar to kernel tracking, but it is used when the object cannot easily be represented by primitive shapes, e.g. objects like hands and other body parts. It instead represents the object model using colour histograms, edges or contours, and attempts to locate a region in the frame that matches the model [8]. Like kernel tracking, object detection only needs to be performed once at the start of the process, but a comparative advantage is that silhouette tracking can handle a large variety of potentially non-rigid shapes as opposed to the primitive objects dealt with by its counterpart [8].

2.3 Approaches From Literature

As mentioned previously, many aspects of computer vision do not have perfect solutions, and object tracking is a good example of this. Selecting a tracking algorithm is not only influenced by an object's appearance, motion model and environment, but also the hardware capability of the system, as the tracking software will often be required to share system resources with other programs. In the case where execution speed is a priority, assumptions about the object's context can be made to reduce computation time [15].
The mean shift method is a simple kernel tracking approach that assumes an absence of object occlusion [16], an assumption that can be made in the context of this project. It tracks an object in a predefined region by calculating where the highest density of matching feature points (e.g. matching pixel hue values) in the image is located and shifting the kernel there [16]. This method is efficient, but cannot adapt to changes in object scale and rotation, and so CAMSHIFT (Continuously Adaptive Mean Shift) was developed to address these problems [17]. It first applies mean shift, then adjusts the size of the kernel and fits an oriented ellipse to the object until accuracy is maximised, and thus the new size and rotation of the object has been found - maintaining a similar degree of computational efficiency as its predecessor [17]. Kalman filtering is a point tracking method that assumes the future position of an object has a Gaussian distribution; its position is most likely to be where it was heading, with equally decreasing probability in any direction away from that point to account for noise [18]. It proposes an area that the object is likely to be in, and then adjusts the parameters of its model based on how close it was, and so the success of the tracking converges as it 'learns' the motion model of the object [18]. Proposing a limited search area leads to higher overall system efficiency, as the object detection algorithm does not have to search the entire image every frame, and instead can focus on the most likely area [18]. This method is highly relevant to the project, where the cars have constrained, predictable motion models. Kalman filtering is frequently combined with mean shift in situations

where object data may be frequently missing or noisy, as it will use the last known trajectory of the object to make a best guess [18].

An issue with Kalman filtering is that when tracking multiple similar objects, if two objects are close enough that their probability density fields intersect, the tracker may 'jump' to the wrong object [19]. In the case of the testbed, it is critical that cars can be distinguished even when they collide. The Condensation algorithm was created to solve the problem of tracking objects moving in a cluttered environment [19]. It does this in a similar way to Kalman filtering, but it assumes an arbitrary probability density function and morphs it over time using feedback [19]. It has been found to be accurate enough to track a single leaf on a tree blowing in the wind [20], implying that it would be sufficient to track small cars in a dynamic background.

Efforts have been made to analyse trackers in the context of Raspberry Pi hardware, but they typically resort to visual aids such as coloured markers [27], an approach that Hurst found to be ineffective for the testbed [5]. Törnberg [21] had success using a Raspberry Pi with CAMSHIFT and the Kanade-Lucas-Tomasi feature tracker (KLT), but as KLT relies on optical flow features, which assume smooth and predictable changes in successive frames [22], it will not be considered in this project.

Section 3

Design Process

3.1 Project Goals

The purpose of this project was to create a new vision system that overcame the limitations of its predecessor, thus enabling all intended functionality of the testbed. In order to achieve this goal, the vision system must:

- **Meet every previous requirement** (specified in Section 3.3). The system must be at least as effective in every way as the previous system to be deemed a success.
- **Support dynamic projected environments.** This will enable the use of changing, interactive traffic scenarios where many non-car agents, such as pedestrians, may be simulated and displayed alongside the real cars.
- **Be able to statically determine the orientation of cars.** There are several common scenarios where it is imperative to know the heading of a stationary vehicle, e.g. at an intersection or in a parking bay. This will both enable these scenarios and increase the overall robustness of the testbed.

3.2 System Redesign

The base testbed consists of a 1.2x0.9m melamine sheet as its driving surface, a projector that superimposes a 2D scene over this area, and a top-view camera suspended from a gantry in the center. Originally, the testbed employed a single Raspberry Pi 3 (RPi3) computer at its core, which performed every task concurrently. This meant that the vision software's already limited resources were further reduced by the overhead associated with communicating with vehicles, running agent scripts, and managing the simulated world. Both the hardware and software of the testbed were redesigned to better meet the goals specified in 3.1. To ensure that the new vision software had the greatest amount

of available resources possible, a second RPi3 was added to the testbed so that all non-vision related tasks could be offloaded. Figure 3.1 shows the upgraded testbed, with the 'Tracker' RPi3 running the vision software, and the 'Controller' RPi3 handling all other tasks, communicating via Ethernet.

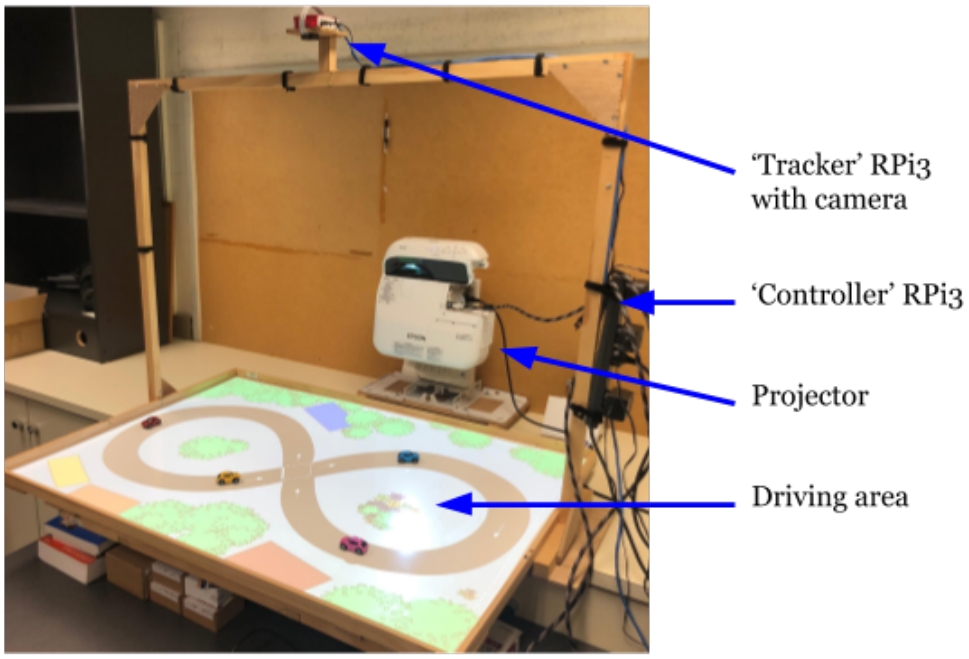

Figure 3.1: The upgraded testbed.

The previous system's control flow was simply:

1. Connect to the cars.
2. while not finished:
   (a) Display the scene.
   (b) Analyse camera frame to determine car states.
   (c) Send commands to the cars.

This process was found to be inadequate for the new system for several reasons. The first and foremost among these was that since colour could no longer be used for car identification, a new method of assigning identity had to be established. This necessitated the addition of the 'Identification' stage, where the Controller sends an ordered list of the names of the cars it wishes to track, and the Tracker then assigns these names to the unknown cars from left to right at the beginning of the simulation.
Another issue was that the previous software could not accurately map the positions of cars from camera space (pixels) to world space (metres), and instead used a heuristically-defined scaling factor to estimate the correct values. To solve this, the 'Calibration' stage was added, where the Controller

displays a known calibration image, and the Tracker analyses its perspective of the image to produce a homography filter. All future car positions will be passed through this filter to give their corresponding world positions with a higher degree of accuracy. This stage also allows the Tracker tolerance to errors in camera position and rotation, as can be seen in Figure 3.2. The red box represents the accuracy of naive, uncalibrated positions, and the green box represents the filtered accuracy.

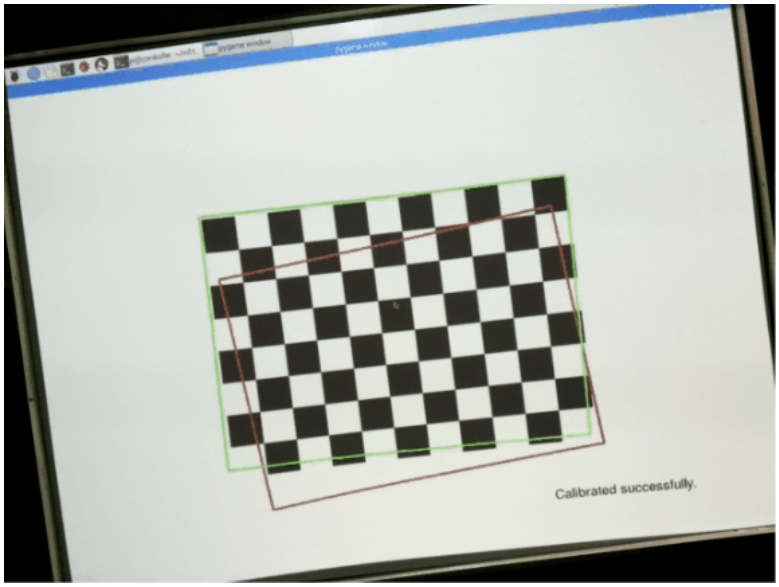

Figure 3.2: Demonstration of the calibration stage accounting for abnormal camera placement.

The final reason that the previous control flow was inadequate is that as the testbed now operated using two RPi3s, there needed to be a communication protocol for handshaking and managing the transmission of data. The new control flow is shown by Figure 3.3.

3.3 Vision System Requirements

Table 3.1 specifies the design requirements for the vision system. As this project seeks to create a system that is categorically superior to the system developed by Hurst, it must at least meet all of his validation criteria, and thus requirements R1-R8, those originally specified by Hurst, have been included. Two new requirements have also been added: the system must be able to track and differentiate without relying on car hues (R9), and it must be able to determine the orientation of cars that are stationary (R10).

Hurst argued that the system's ability to track cars in a changing, unknown environment was a non-mandatory requirement [5], however this ability is essential to the testbed supporting dynamic projected scenes, and therefore requirement R8 has been made mandatory.

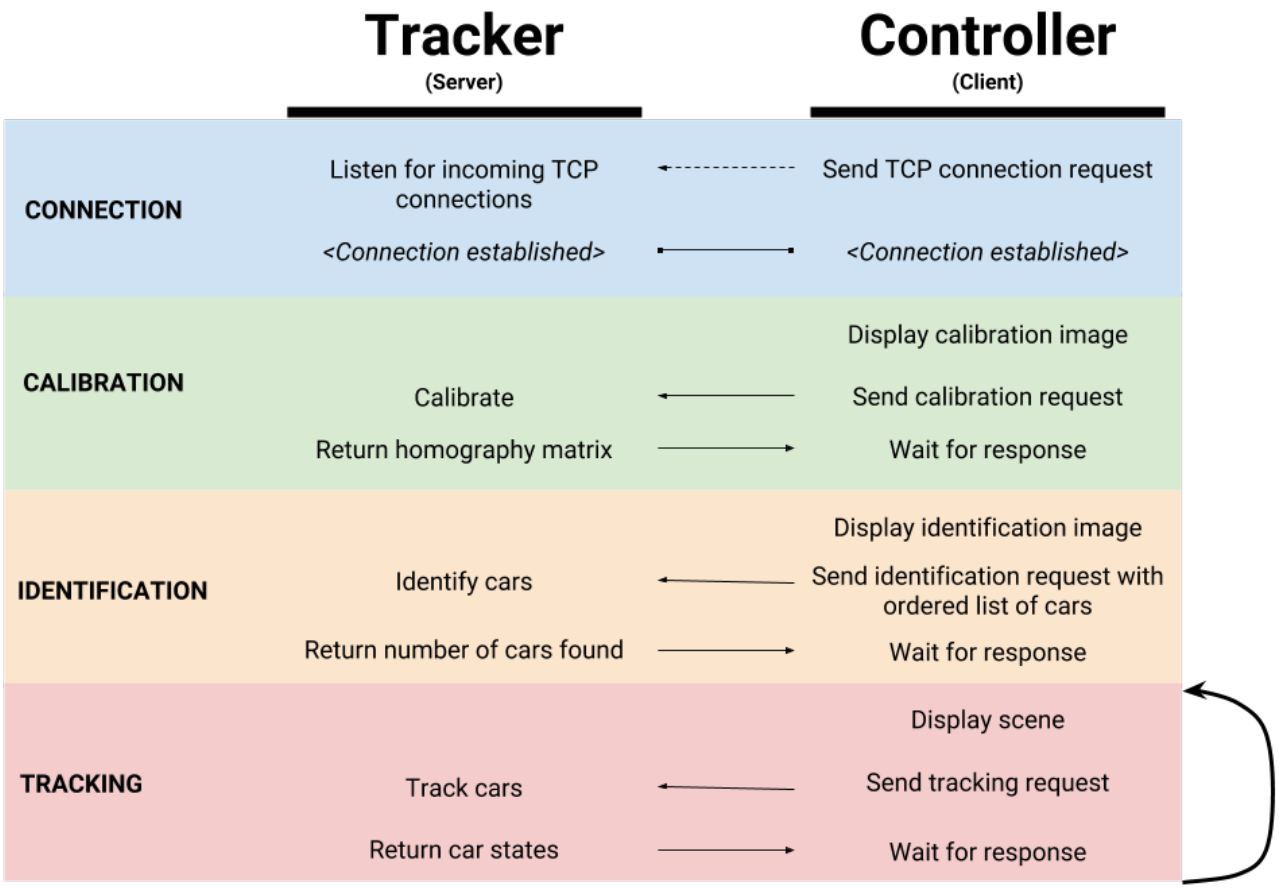

Figure 3.3: The new control flow for the testbed.

Table 3.1: Requirements for the new vision system.

ID	Description	Rationale	Priority
R1	Real-time operation	Vision data must be processed at a rate that is consistent and timely enough to be useful to agents.	Mandatory
R2	Accurate classification	Cars must be classified correctly, minimising the proportion of false negatives (detection failures), but more importantly the number of false positives (misclassifications), as it is generally preferable to miss a detection than provide agents with false data.	Mandatory
R3	Accurate state estimation	The degree of noise associated with car state information (position, orientation, velocity) must be small enough that it does not reduce the usefulness of the data.	Mandatory
R4	Occlusion tolerant	As the testbed is designed with interaction in mind, cars may become occluded during operation. The vision system must therefore be robust to infrequent, momentary occlusions.	Mandatory
R5	Car orientation independent	The cars will change state at runtime and thus the vision system must be able to process car images in any possible orientation.	Mandatory
R6	Tolerant of lighting variation	As the testbed was designed to be portable, it may be moved to areas with different lighting (incandescent, fluorescent etc.), and this must not prevent the normal operation of the vision software. It may be assumed, however, that all such environments will be well lit.	1
R7	High car speed	The speed of the cars should not be limited by the capabilities of the vision system, however high object velocity can pose a challenge to tracking.	2
R8	Background independent	The testbed should be able to display a myriad of dynamic projected scenes without the vision system being affected.	Mandatory
R9	Car hue independent	Cars should be able to be tracked independent of their apparent hue features, which a) will change in different scenes and b) may not be unique to the trial; cars with the same colour should be able to operate simultaneously.	Mandatory
R10	Static orientation determination	A car should not have to be moving to determine its orientation - many common traffic scenarios involve stationary vehicles.	Mandatory

3.4 Evaluation Framework

The effectiveness of the developed tracking methods will be assessed using the following validation criteria shown in Table 3.2. The criteria for requirements R1-R8 are the same as those specified by Hurst, with the following changes:

- R1 must be able to track all 7 cars at at least 5 frames per second. This is down from the 8 cars specified by Hurst, as the Raspberry Pi 3 can only support 7 simultaneously active Bluetooth connections [23].
- Hurst argued that position estimates were required to be accurate to within 2mm, as this was comparable to what other similar projects had achieved [5, 24, 25]. A more lenient threshold of 20mm will be used, as a) the aforementioned methods used detection-only approaches that are incompatible with the new goals and b) experimentation suggests that position estimations need only to be within half a car length to be useful to the testbed.
- The validation criteria for R5, R6 and R8 have been altered to focus on tracking accuracy instead of the overall 'performance' of the system. This was done because R1 is a mandatory requirement that sets a 5fps threshold, so no further validation of tracking speed is necessary.

The criteria for requirement R9 will be fulfilled if the vision system can support fleets of cars with duplicate hues, and cars without easily discernible hue values (black/white cars), without affecting the overall system's accuracy or speed. The criteria for requirement R10 will be fulfilled if the vision system is able to accurately assess the bearing of every car in the scene from a single frame.

Table 3.2: Validation criteria for the system requirements.

ID	Description	Validation Criteria
R1	Real-time operation	Must be able to track up to 7 cars simultaneously at a speed of at least 5 frames per second.
R2	Accurate classification	The true positive detection rate for each car is >= 95%, and the false positive detection rate >= 0.5%.
R3	Accurate state estimation	Position estimates are accurate to within 20mm, orientation to within 10°, velocity measurement noise is <5mm/s.
R4	Occlusion tolerant	Continues to track position accurately or correctly reports failed detection when car is occluded, re-commences tracking within 1 sec of car returning.
R5	Car orientation independent	Tracking accuracy is unaffected by car orientation.
R6	Tolerant of lighting variation	Tracking accuracy is unaffected by reasonable changes in lighting conditions.
R7	High car speed	Successfully tracks cars moving at up to 0.3m/s, experimentally determined to be the top speed of the Micro Cars.
R8	Background independent	Tracking accuracy is unaffected by reasonable changes in the projected driving environment.
R9	Car hue independent	Can track identical cars and cars without detectable hue values.
R10	Static orientation determination	Can determine the orientation of stationary cars.

Section 4

Design Outputs

To meet the project requirements, three problems had to be solved:

1. Detection of likely car regions in a dynamic background.
2. Orientation determination of stationary vehicles.
3. Tracking (potentially identical) cars over time to preserve their unique identities.

The following section will detail the process taken to solve each of these problems.

4.1 Detection

In the previous software, car detection was achieved by subtracting an image of the known background from a frame, which would result in a difference image highlighting the regions which did not match, i.e. the car regions. To meet requirement R8, the new vision software was required to support projected backgrounds which could change in unknown ways at runtime, and so background subtraction was not a compatible detection method.
To solve the detection problem, two properties unique to the context of the testbed were leveraged:

I. The projector illumination raises the pixel intensity of semi-reflective surfaces. This means that the white melamine driving area has notably higher intensity values when viewed from the overhead camera.

II. The shape and size of cars is always constant, with the only transformations being 2D position and rotation changes.

The chosen detection method takes advantage of Property I by taking the raw frame and passing it through a global thresholding algorithm, which produces a binary mask with white pixels representing real (non-projected) objects. The threshold value was determined experimentally, with 110 found to be

the minimum at which projected areas are no longer visible. The morphological operations of dilation and closing are then applied to the mask, in order to fill any holes in the car regions. The next step takes advantage of Property II, by using a blob detecting algorithm on the filled car region image. The blob detector parameters are tuned to the general size and shape of the car regions, and this allows it to filter out any white regions that are unlikely to belong to cars. Figure 4.1 demonstrates each stage of this detection process.

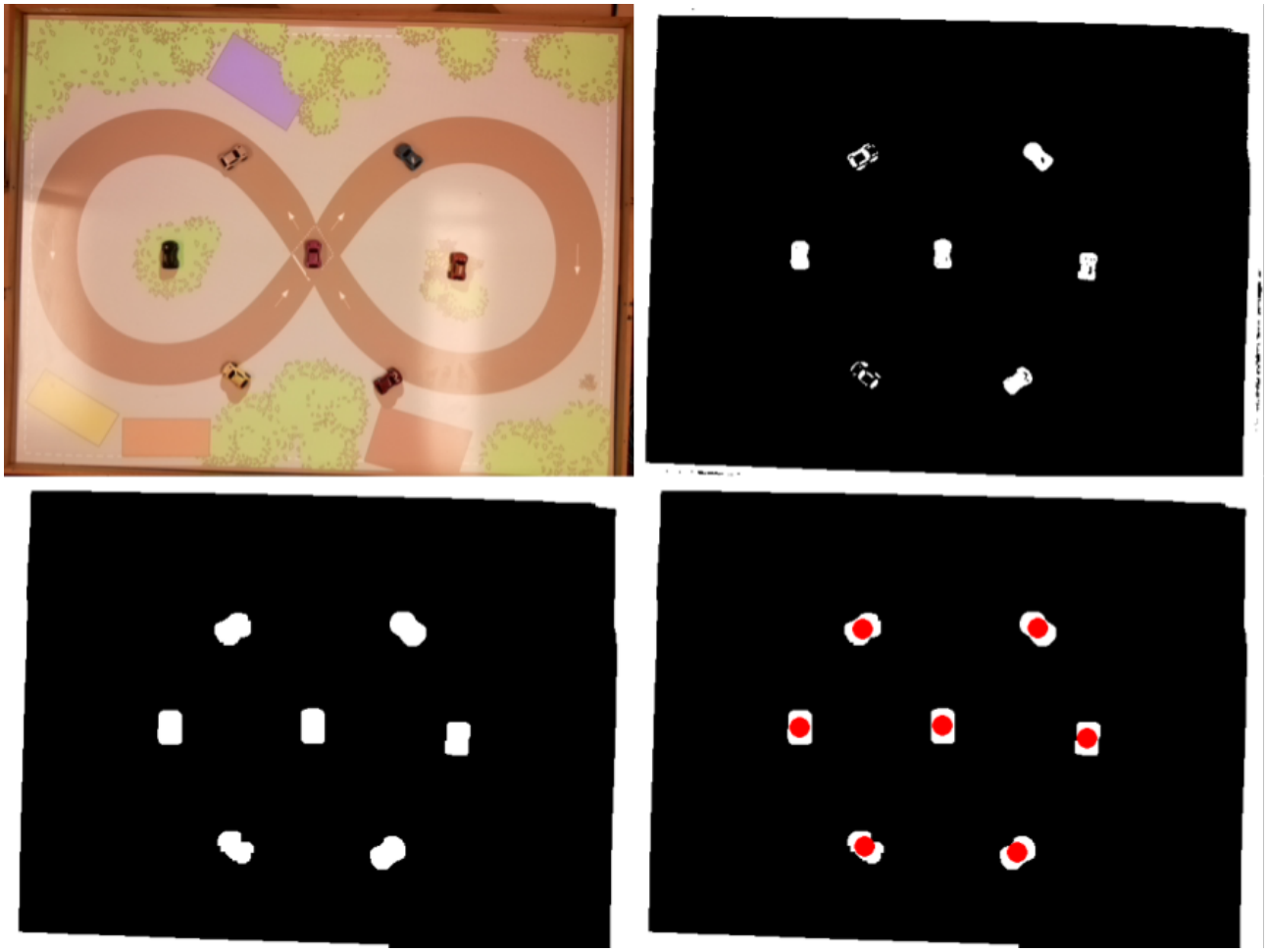

Figure 4.1: Raw image (top left), image after global thresholding (top right), image after morphological operations (bottom left), image with cars detected (bottom right).

While this detection method was sufficient for most general driving scenarios, an issue arose when two or more cars collided with each other. In this case, their individual blob profiles would merge to form a ‘superblob’, as shown in Figure 4.2.

Figure 4.2: Two cars on a collision course (left), the formation of a superblob (right).

As superblobs are never the same shape or size of individual cars, they are filtered out by the blob detector, and all cars within the blob mass go undetected. This issue was resolved by a simple two-step process. Given a superblob:

1. Fit a line to the region.
2. Draw a black line perpendicular to the first line at the centroid of the region.

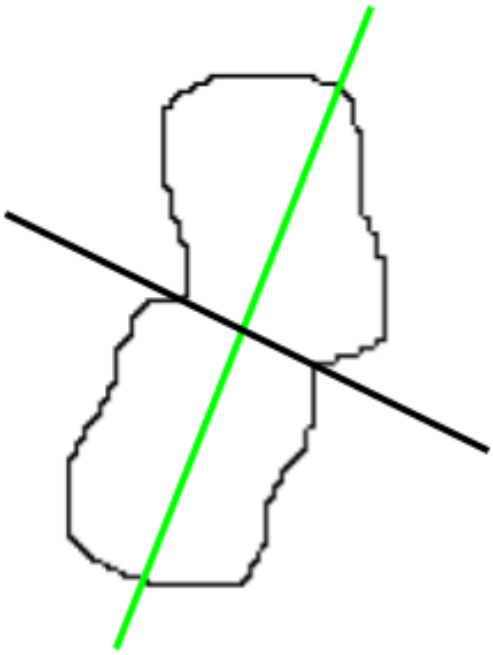

Figure 4.3: The superblob division technique.

The black line forms a dividing wall that separates the blob into two separate regions, which the blob detector can then correctly label as cars. This technique can be extended to collisions of more than two cars, but the growing complexity of the problem coupled with the relative unlikeliness of a 3+ car collision occurring meant that this was too computationally expensive to justify its inclusion in the final system.

4.2 Static Orientation Determination

To determine the orientation of cars at runtime, the previous system simply calculated the angle between a past position and the current position and inferred that this must be the approximate heading of the car. This method was chosen both because it is computationally inexpensive enough to be run every frame, and also because the contours of the cars have rotational symmetry, making them impossible to orient without relying on other salient features. As mentioned before, the primary drawback to this approach is that a car's orientation cannot be determined unless it moves, and a car's agent cannot safely choose to move without knowing which direction it will go. In order to determine a car's orientation from a single frame, without relying on trajectory information, the following algorithm was developed:

1. Given a car region, fit an oriented bounding box to its contour.
2. Halve the width of the bounding box, and bisect the box at its centroid.
3. Count the number of black pixels in each half.
4. The angle between the centroid of the blob and the centroid of the half with the highest count is the angle of the car.

This method takes advantage of a property of the Micro Cars, in that they have a prominent black windscreen at the front of their bodies. This algorithm is demonstrated in Figure 4.4.

A case where this algorithm fails is with black Micro Cars, as the number of black pixels in each half is equivalent. This necessitated the use of the CLAHE algorithm [26], a form of histogram equalisation that, used in this context, grants the ability to distinguish between different shades of black. Figure 4.5 demonstrates the effectiveness of this algorithm to segregate the windscreen section from the body of the car, at which point the static orientation determination algorithm may be applied.

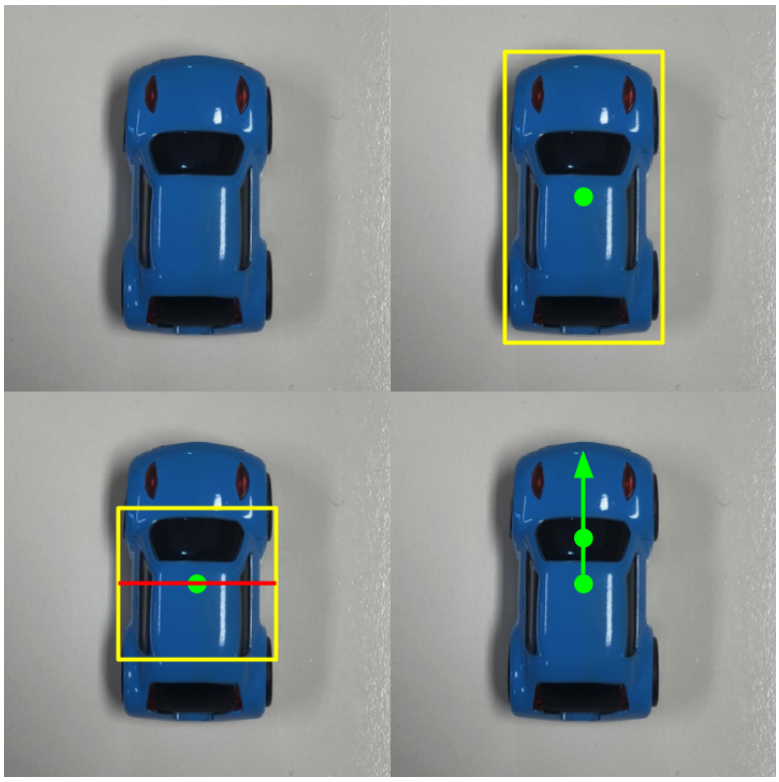

Figure 4.4: A car region (top left), fitted with an oriented bounding box (top right), halved and bisected (bottom left), resulting orientation (bottom right).

Figure 4.5: A black car (left), processed with CLAHE (right).

The use of CLAHE makes the overall orientation determination process relatively computationally expensive, but it only needs to be used in three scenarios: for each car during the initial identification stage (see Section 3.2), when a car's velocity drops to zero, and when a car's tracking is lost and regained. In all other situations, the dynamic method used by the previous vision system is more appropriate, and the vision system should alternate between each method on an as-needed basis.

4.3 Tracking

The previous vision system did not use any tracking method, as it did not require one - car identities were inherently bound to their unique colours, and thus could be resolved by detection (histogram comparison) alone. For the new vision system to be able to determine car states in dynamic environments, car hues can no longer be leveraged, as they will no longer be constant, and so a tracking method must be used. Three types of tracking were explored:

- Kalman filtering, a point-based tracking method,
- The Condensation algorithm, a silhouette-based method,
- Three kernel-based methods - CSRT, KCF and MOSSE.

4.3.1 Kalman Filtering

Kalman filtering is well-established method for estimating a real trajectory from noisy or incomplete data [18]. It maintains a movement model for each tracked object that it uses to predict its position and velocity at each successive timestep. Whenever it receives a new measurement (from the detection phase), it compares it with its prediction and corrects its internal model for that object. This method typically works best for objects that move in a continuous and predictable fashion, such as thrown objects [18]. Kalman filtering was explored because of its ability to interpolate an object's position smoothly given a set of data, thus potentially reducing the impact of an imperfect detector. It was incorporated into the vision system in the following manner.
A Kalman filter is maintained for each active car, and at each timestep it makes a prediction for its car's position at the next timestep. The detection method specified in Section 4.1 runs as soon as a new camera frame is available, and returns a list of unlabelled car positions. Each of these positions is allocated to each filter based on Euclidean distance; the filter with a predicted position closest to a detected position obtains that position, and then this continues among the remaining filters until no detected positions remain, as shown in Figure 4.6.

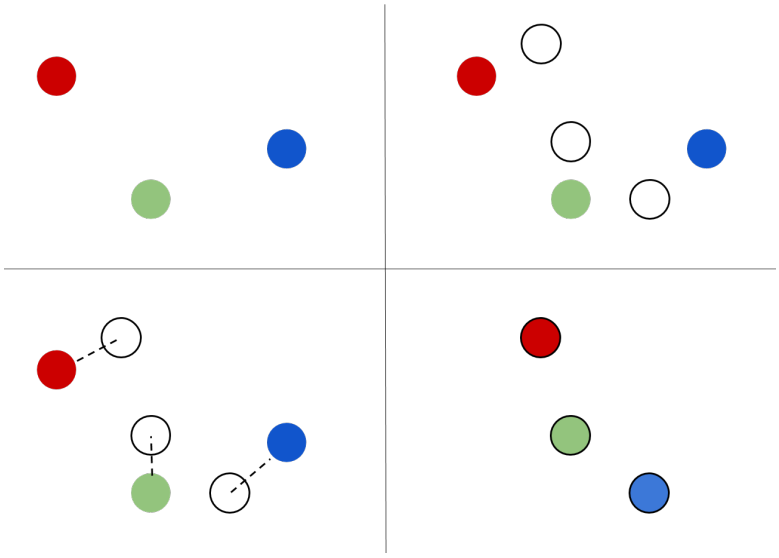

Figure 4.6: Associating predictions with known detections by minimising Euclidean distance.

The Kalman filtering aspect of this method is relatively cheap compared to the detection component, which has constant performance independent of the number of cars in the scene, and thus many cars may be added without affecting the overall processing speed. Additionally, the rate at which detection is run can be tweaked to improve performance, and it was found that even when running detection only once per second, the tracking of the cars was maintained at an accuracy level sufficient for normal use. A major drawback of this method is that it was designed to track singular, isolated objects such as rockets [27], and as such it is not natively suited to a multi-object environment. If a car's displacement between frames is large, whether due to a low detection rate or a high car speed, identities may become ambiguous and thus the heuristic for assigning detections to filters may work incorrectly. In these cases identity swapping becomes a problem, but as long as the detection rate is high enough this generally does not occur.

4.3.2 Condensation Algorithm

The Condensation algorithm was also explored due to its apparent ability to tightly track objects in the presence of significant background clutter. Given an initial template, it maintains a population of particles that each represent a particular configuration of the template. At each timestep, the particles are given a score based on how well their template configuration fits the given image region, and then are re-distributed with respect to the previous timestep's scores and a stochastic element. This results in each car being represented by a probability distribution, with the peak of the distribution representing the most likely state.

The algorithm was adapted for use in the testbed in the following manner. A contour template was created, and the following configuration model was used:

$$M_t = (x_t, y_t, \theta_t, s_t)^T$$

Where x_t, y_t represent the position of the template in pixel coordinates, θ_t is its 2D rotation, and s_t is its scaling factor. At each timestep t, the template is transformed by M_t over a frame that has been processed with the Canny edge detector [28]. The number of matching pixels is counted, and a score is given according to the function:

$$\text{score} = \left(\frac{\text{count}}{\text{maxCount}}\right)^{-4}$$

This function is designed to give much higher weighting to counts that are closer to the maximum, and thus reduce the influence of clearly incorrect particles on the next timestep's distribution. As this method is particle-based, it is well-suited to parallelisation and can be adapted to take advantage of the quad-core RPi3 architecture [23] for greater performance. Additionally, the method naturally tracks object orientation as well as position, and so, theoretically, no additional resources need to be allocated to orientation determination.

In practice this does not work for the cars, however, due to the problem mentioned in Section 4.2 - car contours are rotationally symmetrical. This causes the Condensation algorithm to frequently flip the contour template 180°, due to the fact that both polar directions give identical scores as shown in Figure 4.7.

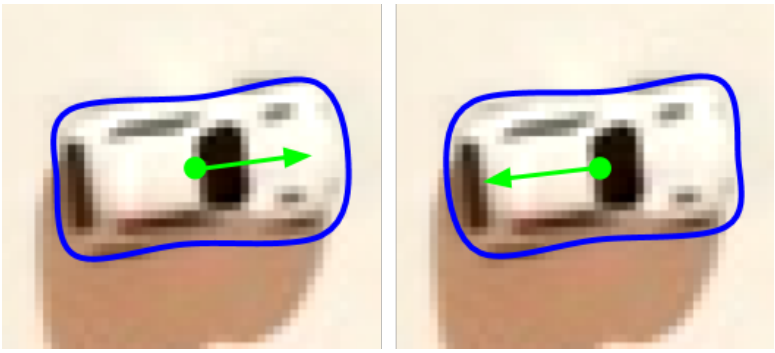

Figure 4.7: A high scoring match (left), the same configuration flipped 180° with the same score (right).

To prevent these flips, a constraint can be placed on the rotation element of the configuration model limiting the amount that it can rotate between timesteps, but this makes it less reactive when cars turn sharply.

Another downside of this algorithm is that it requires several hundred particles (>300) per car to be accurate, which means that it scales to larger car fleets very poorly.

4.3.3 Correlation Filters

Tracking methods based on correlation filters (CFs) follow the same underlying principle: given an initial image region as a template, search the image for an area that the template best matches, and adapt the template over time to better match the tracked region. Three CF-based methods were chosen for examination - CSRT [29], KCF [30] and MOSSE [31]. The foremost reason that these methods were chosen is that they take advantage of the Fast Fourier Transform to perform rapid computations in the frequency domain and thus they typically exhibit much higher speeds than other types of tracking [31]. Another advantage is that they require only a single detection at the beginning of a trial, in order to initialise the position and size of the region to be tracked, after which detection is only required if the target is lost.

These methods were integrated into the vision system in the following way. At the end of the identification stage (see Section 3.2), a CF tracker is created for each car, and intialised with a bounding box at the car's position. Bounding boxes are set as squares slightly wider than the length of a car, thus ensuring that the entire car is always present in the initialisation region, independent of its orientation. At every timestep, each tracker is provided with the most recent camera frame, and it returns either the position of the new bounding box or a flag indicating that tracking was lost. In the latter case, detection is performed and the Euclidean distance method described in Section 4.3.1 is used to resolve car identities, but with only those trackers who had lost their targets being added to the resolution pool.

A limitation of CF methods is that they are unable to track objects travelling at high velocities in the camera's perspective. This is due to the search behaviour of the methods only exploring regions within the immediate vicinity of the last known position [33].

Another issue with all CF methods is that they do not incorporate information regarding the tracked region's surroundings; they are focused solely on the target, and having limited information about the tracking context can cause the target to drift out of the center of the region [31]. During the project it was found that smaller bounding boxes were more resistant to drift, but also caused tracking to be lost at lower car speeds.

Section 5

Results

To assess the efficacy of each tracking algorithm to meet the project requirements, four experiments were designed that encompassed the full scope of validation criteria outlined in Section 3.4. All algorithms were implemented in Python using the OpenCV computer vision library. Hurst's algorithm was implemented in C++, which generally gives superior runtime performance over interpreted languages like Python [32]. His algorithm was therefore re-implemented in Python so that the relative performance differences could be assessed fairly.

5.1 Computational Performance

The purpose of this experiment was to determine which algorithms met requirement R1. In accordance with Hurst's methodology, three trials of 1,000 frames were conducted for one to seven cars in order to assess the average speed of each tracking method, with Hurst's algorithm used as a baseline.

Table 5.1: Average computational performance comparison for each tracking method (frames per second, 1 d.p.). Values in grey indicate where the method fell below the 5 fps requirement.

	Number of Cars						
Tracking Method	1	2	3	4	5	6	7
Baseline	5.0	5.0	5.0	5.0	5.0	5.0	5.0
Kalman Filtering	5.9	5.9	5.9	6.0	5.9	5.9	5.9
Condensation Algorithm	0.8	0.4	0.3	0.2	0.2	0.1	0.1
CSRT	5.0	2.5	1.8	1.3	1.0	0.9	0.7
KCF	20.3	12.0	8.7	6.0	5.0	4.3	3.7
MOSSE	30.0	30.0	30.0	30.0	30.0	30.0	30.0

Table 5.1 shows that only Kalman Filtering and MOSSE were able to consistently exceed the baseline, and therefore were the only methods to meet requirement R1. For this reason, all other methods will be excluded from further testing.

5.2 Classification and Estimation Accuracy

This experiment was designed to assess the degree to which methods met requirements R2 and R3. It involved 6,000 frames showing all seven cars under varying lighting conditions, and both the classification accuracy and state estimation accuracy were assessed.

Table 5.2: Proportion of true positives and false negatives as a percentage of total frames.

Tracking Method	True Positive %	False Negative %
Kalman Filtering	0.875	0.076
MOSSE	1.000	0.003

Table 5.3: Position determination accuracy based on average distance from car centroid.

Tracking Method	Position Error (mm)
Kalman Filtering	5.59
MOSSE	8.34

The MOSSE method was observed to exhibit larger position deviations than Kalman Filtering, largely due to the target drift phenomenon described in Section 4.3.3, which caused cars to move slightly to one side of their tracking regions after some time had elapsed. This was especially observed during car collisions, in which case a small section of one car would appear in the other car's tracking region and exert some influence over the region's trajectory.

5.3 Velocity Tolerance

This experiment was performed to assess the ability of trackers at different car velocities (requirement R7). It involves a single car travelling at a constant speed on a white background. A trial of 500 frames was performed for each of the four speed levels. The proportion of tracking losses per trial is used to determine the level of success. A tracking loss is deemed to have occurred when either the predicted location is equal or greater to one car length away from the actual car centroid, or the tracker explicitly admits failure and requests redetection.

Table 5.4: Tracking losses at different car velocities.

	% Top Speed			
Tracking Method	25%	50%	75%	100%
Kalman Filtering	0	0	5	34
MOSSE	0	0	0	4

5.4 Static Orientation Determination Accuracy

The final experiment did not test individual trackers but rather the efficacy of the static orientation estimation method described in Section 4.2, as it is used in conjunction with all tracking methods. Images of each colour car were taken, rotating the car in 5° intervals for a total of 72 images per colour, thus ensuring complete coverage of each car's representations.

Table 5.5: Static orientation determination accuracy based on average difference in estimated and actual orientation.

Car Colour	**Orientation Error (degrees)**
Black	5.4
Orange	4.7
Pink	5
Green	5.5
Yellow	4.2
White	4.3
Blue	4.7
Red	4.4

Section 6

Discussion

As shown in Section 5.1, only two tracking methods, Kalman Filtering and MOSSE, were able to exceed the 5 fps minimum in all cases. KCF exhibited good performance for small fleets, but this exponentially dropped off as more cars were added. It can also be observed that the processing speeds of both Kalman Filtering and MOSSE appeared to be almost constant, irrespective of the size of tracked fleet. In the case of Kalman Filtering, this is due to the fact that the detection stage of the method is orders of magnitude more computationally expensive than the tracking stage, and as detection is performed once for any number of cars, adding more cars does not cause a noticeable impact. In the case of MOSSE, however, its 30 fps limit was imposed by the camera, as this was its set capture rate. MOSSE has been observed to track at up to 450 fps on standard hardware [30], and thus it is fair to assume that it could manage at least a quarter of that on an RPi3. The significant performance gap between MOSSE and the other correlation filter techniques can be ascribed to the much simpler sum of squared error correlation method used by MOSSE [31]. This results in theoretically lower accuracy than its counterparts, however this was not observed in the context of the testbed, most likely because of the relatively simple 2D transformations to which the cars are constrained.

The results from Section 5.2 indicate that both methods were able to meet requirement R2 convincingly, with a very high true positive rate and low false negative rate. MOSSE scored an almost perfect classification rate, with only a few immediately resolved tracking losses across the entire experiment. The observed estimation error for both methods was also well below the threshold specified by requirement R3. Additionally, the experiment found that altering the projected hue had no observable effect on the tracking ability of either method, nor did moderate lighting changes. Cars with duplicate hues were also used successfully, and these observations indicate that both methods successfully meet requirements R6, R8 and R9.

As only the Condensation algorithm incorporated orientation information into its tracking process, it was found to be the only tracker to fail requirement R5. In the case of the other trackers, they had no way of internally representing orientation and thus it had no bearing on their overall accuracies. Despite correlation filters generally having difficulty with tracking at high velocities [31], MOSSE

was observed to only lose tracking a few times at top speed. In contrast, the Kalman Filtering method began to lose tracking at an increasing rate after the car reached 75% of its top speed, to the point where tracking became near impossible at top speed. This result appears to be due to Kalman Filtering operating at a framerate five times lower than that of MOSSE, as processing frames slower causes the car displacement to be proportionally greater between tracking timesteps, and thus outside of the tracker's search area. As such, only MOSSE was able to meet requirement R7.

The results from the orientation determination experiment (Section 5.4) seemed to be consistent across all car colours, with an average error of approximately 5°. As 10° was the specified threshold, the static orientation estimation method meets requirement R10 and is thus appropriate for use as a component of the overall vision system.

The remaining requirement, R4, was observed to have been met implicitly by the tracking loss protocol used by all trackers, where failures are immediately followed by a detection step that re-initialises the lost tracker.

Table 6.1 details the performance of each method against the validation criteria. It can be seen that both the Kalman Filtering and MOSSE methods met all mandatory criteria. As MOSSE also met all optional criteria and exhibited a significantly superior processing speed, it appears to be the most appropriate choice for the testbed.

Table 6.1: Method validation results. Entries marked with '-' indicate that the requirement was not tested due to failing a mandatory requirement.

	Method				
Requirement	Kalman	Condensation	CSRT	KCF	MOSSE
R1	Yes	No	No	Partial	Yes+
R2	Yes	-	-	-	Yes
R3	Yes	-	-	-	Yes
R4	Yes	No	Yes	Yes	Yes
R5	Yes	No	Yes	Yes	Yes
R6	Yes	-	-	-	Yes
R7	No	-	-	-	Yes
R8	Yes	-	-	-	Yes
R9	Yes	Yes	Yes	Yes	Yes
R10	Yes	Yes	Yes	Yes	Yes

Section 7

Conclusion

This project successfully developed a multi-object tracking system with superior capabilities to its predecessor and has thus expanded the utility of the autonomous vehicle testbed to its intended scope. The installation of this vision system will allow the use of dynamic, interactive traffic scenarios, and facilitate several new opportunities for the research and development of the ongoing testbed project.
The research explored five existing object tracking algorithms to assess their efficacy in following several miniature cars using the limited computational resources of a Raspberry Pi 3, in a variety of environmental conditions. Two of the algorithms emerged as being suitable for this task, (Kalman Filtering and the MOSSE), and three were found to be deficient in terms of processing speed (the Condensation Algorithm, CSRT and KCF). While both of the surviving algorithms could potentially be used in the testbed, MOSSE was able to accurately track the cars at their top speeds, as well as running at over five times faster than its nearest competitor, and for this reason it was selected for use in the final vision system. In addition to assessing existing trackers, a simple and effective static position and orientation estimation method was developed, and it is possible that future researchers may benefit from applying these techniques in their own low-cost vision projects.
The final vision system is comprised of an initial detection phase to localise the cars, followed by indefinite MOSSE tracking. Further detections are performed only if a car's tracking is lost, and following this the regular tracking phase resumes. Car orientation is determined dynamically if its velocity exceeds 0.01m/s, otherwise static orientation estimation is performed.
A limitation of this system is that since tracking relies on historical data, cars cannot be removed from and replaced in the driving area at runtime, as in the previous system, although this is not a critical feature of the testbed. The valuable additions of support for dynamic projected backgrounds and the ability to use any car independent of its colour outweigh the loss and indicate that, overall, this project has been a complete success.

7.1 Future Work

As mentioned before, MOSSE's processing speed was unaffected by large car fleets, and the data suggests that it could easily support a fleet twice the size without falling below the 5 fps minimum. This implies that there is an abundance of free computational resources during peak operation, and thus in the future the two Raspberry Pi 3 computers used in the testbed could potentially be reduced to only one by taking advantage of multiprocessing on the quad-core architecture [23].
The primary drawback of MOSSE is that it is susceptible to drifting and losing some target precision over time. There has been recent work undertaken into the development of context-aware correlation filter tracking, where instead of only considering the target region, the tracker also analyses the region's immediate surroundings, and this has been shown to address the problem of target drift without significantly affecting the tracking speed [33]. This emerging technology could potentially upgrade the capabilities of the testbed while still maintaining compatibility with the vision system, and therefore it is a recommended area for future exploration.

References

[1] Fagnant, D.J. and Kockelman, K., 2015. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77, pp.167-181.

[2] Bonnefon, J.F., Shariff, A. and Rahwan, I., 2016. The social dilemma of autonomous vehicles. *Science*, 352(6293), pp.1573-1576.

[3] Reuters. 2018. *U.S. spending plan include $100 million for autonomous cars research, testing*. [ONLINE] Available at: https://www.reuters.com/article/us-autos-selfdriving-congress/u-s-spending-plan-include-100-million-for-autonomous-cars-research-testing-idUSKBN1GY074. [Accessed 4 October 2018].

[4] A., D., 2002. *Computer Vision: A Modern Approach.* Prentice Hall.

[5] Hurst, A., 2018. Computer Vision for a Real-Time Physical Traffic Testbed.

[6] ZenWheels. 2018. *ZenWheels Micro Car.* [ONLINE] Available at: http://zenwheels.com/zenwheelsmicro-car-37.html. [Accessed 4 October 2018].

[7] Cheng, M.M., Jiang, H. and Li, J., 2014. Salient Object Detection: A Survey. *arXiv, 1411*(5878).

[8] Yilmaz, A., Javed, O. and Shah, M., 2006. Object tracking: A survey. *Acm computing surveys (CSUR), 38* (4), p.13.

[9] Ågren, S., 2017. Object tracking methods and their areas of application: A meta-analysis: A thorough review and summary of commonly used object tracking methods.

[10] Mlích, J., 2009. Feature Point based object tracking: AMI training program report.

[11] Learn OpenCV. 2018. *Object Tracking using OpenCV (C++/Python).* [ONLINE] Available at: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/. [Accessed 5 October 2018].

[12] Janků, P., Koplik, K., Dulík, T. and Szabo, I., 2016. Comparison of tracking algorithms implemented in OpenCV. In *MATEC Web of Conferences 20th International Conference on Circuits, Systems, Communications and Computers (CSCC 2016).* EDP Sciences.

[13] Primet, M. and Moisan, L., 2012. Point tracking: an a-contrario approach.

[14] Comaniciu, D., Ramesh, V. and Meer, P., 2003. Kernel-based object tracking. *IEEE Transactions on pattern analysis and machine intelligence, 25* (5), pp.564-577.

[15] Grabner, H., Matas, J., Van Gool, L. and Cattin, P., 2010, June. Tracking the invisible: Learning where the object might be. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (pp. 1285-1292). IEEE.

[16] Comaniciu, D. and Meer, P., 2002. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence, 24* (5), pp.603-619.

[17] Bradski, G.R., 1998. Computer vision face tracking for use in a perceptual user interface.

[18] Taylor, L.E., Mirdanies, M. and Saputra, R.P., 2016. Optimized Object Tracking Technique Using Kalman Filter. *Mechatronics, Electrical Power & Vehicular Technology, 7* (1).

[19] Blake, A. and Isard, M., 1997. The condensation algorithm-conditional density propagation and applications to visual tracking. In *Advances in Neural Information Processing Systems* (pp. 361-367).

[20] Isard, M., 2018. *The Condensation Algorithm Home Page.* [ONLINE] Available at: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL COPIES/ISARD1/condensation.html. [Accessed 10 October 2018].

[21] Törnberg, I., 2016. Real time object tracking on Raspberry Pi 2.

[22] Tomasi, C. and Kanade, T., 1991. Detection and tracking of point features.

[23] Raspberry Pi. 2018. *Raspberry Pi 3 Model B.* [ONLINE] Available at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed 20 September 2018].

[24] Gonzalez, M., Huang, X., Martinez, D.S.H., Hsieh, C.H., Huang, Y.R., Irvine, B., Short, M.B. and Bertozzi, A.L., 2011. A Third Generation Micro-vehicle Testbed for Cooperative Control and Sensing Strategies. In *ICINCO* (2) (pp. 14-20).

[25] Brezak, M., Petrović, I. and Ivanjko, E., 2008. Robust and accurate global vision system for real time tracking of multiple mobile robots. *Robotics and Autonomous Systems, 56*(3), pp.213-230.

[26] Reza, A.M., 2004. Realization of the contrast limited adaptive histogram equalization (CLAHE) for real-time image enhancement. *Journal of VLSI signal processing systems for signal, image and video technology, 38*(1), pp.35-44.

[27] Lefferts, E.J., Markley, F.L. and Shuster, M.D., 1982. Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics, 5*(5), pp.417-429.

[28] Canny, J., 1986. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6), pp.679-698.

[29] Lukezic, A., Vojir, T., Zajc, L.C., Matas, J. and Kristan, M., 2017, July. Discriminative Correlation Filter with Channel and Spatial Reliability. In *CVPR* (Vol. 6, p. 8).

[30] Henriques, J.F., Caseiro, R., Martins, P. and Batista, J., 2015. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 37*(3), pp.583-596.

[31] Bolme, D.S., Beveridge, J.R., Draper, B.A. and Lui, Y.M., 2010, June. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (pp. 2544-2550). IEEE.

[32] Sanner, M.F., 1999. Python: a programming language for software integration and development. *J Mol Graph Model, 17*(1), pp.57-61.

[33] Mueller, M., Smith, N. and Ghanem, B., 2017, July. Context-aware correlation filter tracking. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Vol. 2, No. 3, p. 6).