Using machine learning with cameras as a low-cost substitute for LiDAR and RADAR sensors used in vehicle autonomy

Jack Sines — UP901354

School of Computing

Final Year Engineering Project

Word count: 12870

May 5, 2022

Abstract

This paper explores the feasibility of using stereo cameras as a replacement for LiDAR (light detection and ranging) and RADAR (radio detection and ranging) sensors in the context of vehicle autonomy.

A literature review discusses common sensors, computer vision methods, object detection methods, deep learning algorithms and depth estimation using different camera systems.

The project planning and implementation is explained in detail before the project is then tested and analysed.

The final artefact uses a YOLOv5s algorithm to detect objects in a frame and calculates the distance using the disparity between the images. Oriented FAST and rotated BRIEF (ORB) feature detection is used before the features are matched using brute force matching. The images are taken using a pair of stereo cameras mounted at a 16cm distance from each other. A simple graphical user interface (GUI) is used for testing and demonstration of the program.

This application achieves accurate results in object detection but the distance estimation results are inconsistent.

Table of Contents

List of Tables

List of Figures

Acknowledgements

The author would like to thank Jiacheng Tan for their feedback and support throughout the projects duration.

Chapter 1

Introduction

In 2019, there were 78,855 reported road traffic accidents in the UK. Of those, 66% were due to "driver/rider error or reaction".

Autonomous vehicles aim to reduce the number of road accidents by limiting human input and instead having software control the vehicle (or specific functions of the vehicle). The National Highway Traffic Safety Administration (n.d.) (NHTSA) has suggested a five-part model to classify different levels of automation – this is widely used in the area:

- Level 0: Zero autonomy; the driver performs all driving tasks.
- Level 1: Vehicle is controlled by the driver, but some driving assist features may be included in the vehicle design.
- Level 2: Vehicle has combined automated functions, like acceleration and steering, but the driver must remain engaged with the driving task and monitor the environment at all times.
- Level 3: Driver is a necessity, but is not required to monitor the environment. The driver must be ready to take control of the vehicle at all times with notice.
- Level 4: The vehicle is capable of performing all driving functions under

certain conditions. The driver may have the option to control the vehicle.

- Level 5: The vehicle is capable of performing all driving functions under all conditions. The driver may have the option to control the vehicle.

(National Highway Traffic Safety Administration, n.d.)

There are 3 sensors that are commonly used for vehicle autonomy: LiDAR (Light Detection and Ranging), RADAR (Radio Detection and Ranging) and camera – each of these has their strengths and weaknesses and are used for applications that range from blind spot detection to full vehicle autonomy (self driving).

One of the issues from a consumer standpoint is the cost that LiDAR and RADAR add to the price of the vehicle. Shchetko (2014) noted that "LIDAR systems on top of many of today's AVs cost $30,000 to $85,000 each". While this price has decreased drastically, with "Luminar [offering] autonomous driving-grade LiDARs for under US$1000" (Christoph Domke & Quentin Potts, 2020), it is still more expensive than a camera would be. RADAR is not as expensive, but "varies from $50 to $200" (Intellias Automotive, 2018). Despite this, many manufacturers are still using LiDAR and RADAR in their vehicles.

1.1 Aims and objectives

The overall aim of this project is to find out if cameras can be used as a suitable replacement for RADAR or LiDAR to reduce vehicle cost with a minimal impact on accuracy.

As part of meeting this aim and reaching a conclusion, a piece of software is created that takes footage from stereo cameras, identifies objects and calculates the distance to these objects. By doing this, one of the main purposes of LiDAR and RADAR in autonomous vehicles is covered.

This gives us the following aims for the artefact:

- Detect vehicles in front of the cameras
- Estimate the distance to the vehicle ahead with a good degree of accuracy
- Warn when the distance goes under a threshold
- Analyse the process and results to come to a conclusion on the suitability of the technology to replace functions of LiDAR and RADAR

Additionally, current literature is reviewed to form an understanding of the common sensors used in vehicles, object detection methods and distance calculation.

Once the technology and methods to be used are decided, development methodologies are compared before requirements, design, implementation and testing are detailed.

The artefact is evaluated against requirements before a conclusion is drawn on the title of the report.

1.2 Methods and results

The cameras are calibrated using images taken of a chessboard of a known size - these are used to calculate intrinsics and extrinincs which are used to alter images taken on the cameras. This method works well for removing distortion from the images and aligning pixels in the y axis - both of these things being done to improve and simplify the disparity calculation.

A YOLOv5s detector is used to detect objects and return their bounding box, confidence score and object type. The object detector produces impressive results.

Oriented FAST and rotated BRIEF (ORB) feature detection is used on both images taken before the features are matched using brute force matching.

Distance is then calculated using distance = (baseline * focal length) / disparity. The distance results are disappointing as they are inconsistent - this is discussed in further detail in section 8.

A Graphical User Interface (GUI) is used to allow for easy testing and demonstration of the program.

1.3 Legal, ethical, professional and social issues

During this project I do not plan on gathering any user data – if this changes, however, then I will ensure that I comply with the General Data Protection Regulation (GDPR) and the Data Protection Act when collecting information.

I will be taking dashcam footage to use for testing the software. In the UK, there are no laws against filming with a dashcam provided it is installed safely in the car. The footage will be used for testing and then deleted once the project has been completed.

1.4 Limitations

One of the factors that could limit the projects progress is the authors knowledge of the technology. While I do have experience with Python and computer vision, machine learning and stereo cameras are areas that will have to be researched throughout the project.

Hardware is another limitation, as I am unable to purchase neural network focused hardware which would typically be used for vehicle autonomy. The project is carried out on the available hardware, however this may come at the cost of processing time.

Chapter 2

Literature Review

This literature review discusses and compares the common sensors in vehicle autonomy, looking at the advantages and disadvantages of each in terms of resolution, range, cost, signal strength, etc.

Traditional and machine learning (ML) methods of object detection are then discussed, with examples of popular methods given.

Depth estimation with monocular and stereo cameras is then covered, with detail on camera calibration, rectification and correspondence searching.

Lastly, as part of the conclusion, a ML model is chosen from a comparison of popular models.

2.1 Technologies of vehicle detection/tracking

Three sensors are commonly used for environment perception in autonomous vehicles: Light Detection and Ranging (LiDAR), Radio Detection and Ranging (RADAR) and optical camera – each of these has their strengths and weaknesses.

LiDAR systems have many uses including mapping shorelines, mapping the ocean floor, surveying for construction, etc. More recently, autonomous vehicles have been using this technology to create depth maps of their environments.

LiDAR is hugely popular with the autonomous vehicles currently being developed, with Cruise, Waymo and Hyundai being a few examples of the many companies implementing this technology.

RADAR is currently used for blind spot detection, cruise control, semi-autonomous obstacle detection and braking functions (Mohammed et al., 2020).

2.1.1 A comparison of sensors

Active and Passive sensors

Active systems involve sending signals to the target and measuring the time of flight (the needed time of a signal to transmit and be reflected by the object). LiDAR and RADAR are active systems.

One of the main issues with the active method is that the sensors can pick up echoes from previous signals, which can cause an inaccurate reading.

Passive systems do not send out signals and instead only receive information – cameras are the most common form of passive systems.

LiDAR

According to the National Oceanic and Atmospheric Administration, LiDAR "is a remote sensing method that uses light in the form of a pulsed laser to measure ranges" (NOAA, 2021). These sensors send out a pulse of infrared light and then detect when it has reflected and reached the sensor again - the time of flight can be used to calculate distance and create a 3D point cloud of an environment.

LiDAR has the benefit of being able to see long distances in a high resolution when weather conditions are ideal. Luminar will offer a sensor that has a maximum range of 600 meters, with a range of 250 meters at less than 10% reflectivity - this sensor also has a resolution of up to 300 points per square degree, a field of view of 120 degrees and is able to detect other vehicles at up to 250 meters away (Luminar, n.d.).

However, LiDAR is more expensive than both camera and RADAR - with Luminar predicting a price of between 500 - 1000 USD per vehicle, depending on production volume (Ohnsman, 2021).

RADAR

RADAR works similarly to LiDAR but instead uses radio waves. It has the benefit of radio waves not being affected by poor weather conditions. The resolution is poor when compared to LiDAR or camera, meaning it does not provide as precise information.

Cameras

Cameras are cheaper than LiDAR and have a higher resolution. They are able to interpret colour information (crucial for recognising traffic lights, road signs, etc.) which can help with segmenting the image and isolating objects.

They require higher computing power and are sensitive to bad weather, such as heavy rain. Cameras typically have a lower viewing range than both LiDAR and RADAR. The main disadvantage of cameras in the context of vehicle autonomy is that they do not provide depth information - we must use other methods for this.

2.1.2 Traditional object detection methods

For the purposes of this paper, traditional methods are those that do not rely heavily on machine learning (ML). These methods tend to utilise image seg-

mentation methods such as thresholding-based, edge-based and clustering-based, which aim to divide the image into homogeneous regions.

Abdulateef and Salman (2021) cover and compare these image segmentation techniques well in their paper.

Thresholding-based segmentation

Thresholding-based segmentation looks for a change in the grayscale values of an image – an intensity threshold is set and then any pixel that has an intensity of higher than this threshold will be set to 1 (white) and anything below will be set to 0 (black).

As it uses grey scale images, it deals with less data than other methods and generally "simplifies the recognition operation" (Abdulateef & Salman, 2021). However, due to working with less data it can produce less detailed results and is often sensitive to noise.

Local thresholding can be used to set a threshold value on a per pixel level by looking at the intensity of the surrounding pixels. Park et al. (2021) uses local thresholding to achieve an error rate of 10.9% in one test and 12.5% in another.

This method can be useful when needing to separate the background and foreground (Abdulateef & Salman, 2021).

Edge-based segmentation

Edge-based segmentation finds edges in the image by using the gradients. There are methods that use the first order derivative (such as Sobel), second order derivative (such as Laplacian) and optimal edge detector (such as Canny).

This method can achieve good results – for example, Zaarane et al. (2020) use edge detection as part of the object detection in their paper.

Clustering-based segmentation

There are 3 main methods of region-based segmentation: growing, splitting and clustering. Of the 3, clustering is the most popular – it works by arranging the data into groups (clusters) of similar pixels based on size, colour, texture, etc. The K-Means algorithm is a popular unsupervised clustering algorithm that works by classifying "the input points of data into many groups produced on their inherent distance from one another" (Naik & Shah, 2014).

Clustering is quick and relatively simple, but the number of clusters to be found must be given.

2.1.3 Notable methods of traditional object detection

Viola-Jones detector

Published by Viola and Jones (2001), this method works by combining techniques such as Haar-like features, integral image, Adaboost and cascading classifier (Zaidi et al., 2021). Haar-like features (see figure 2.1 for an example) are searched for in the image. As each pixel has to be checked in this process, integral images (also known as summed-area tables) are used to keep the processing time down.

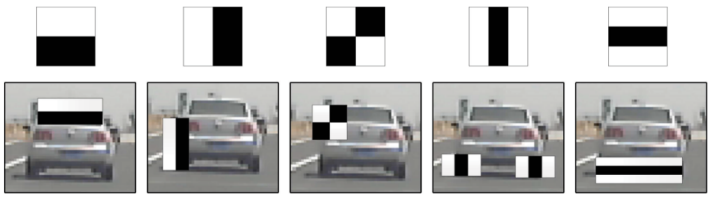


Figure 2.1: Demonstration of haar-like features applied to an image

Source: B. Wang et al. (2017)

A trained AdaBoost is used to pick out the features that are mostly likely to exist in the image of the many features that are identified.

This method is fast and efficient (Zaidi et al., 2021).

Histograms of Oriented Gradients Detector

Published by Dalal and Triggs (2005), the Histograms of Oriented Gradients (HOG) detector works under the idea that "local object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions" (Dalal & Triggs, 2005).

This method creates a feature table of the gradients and the orientation of the edges - this is then used to create a histogram of oriented gradients for parts of the image. The HOG detection method was proposed for pedestrian detection (Zaidi et al., 2021) and is still heavily used today (Mittal, 2020).

Deformable Part Model Detector

Felzenszwalb et al. (2010) published a method that used deformable part models in order to detect generic objects. In addition to this, it was also the first method to implement bounding box regression - a method for predicting the bounding box in an image that has since been used with the Region based convolutional neural network (R-CNN) algorithm.

2.1.4 Object detection using Deep Learning

ML is based on Neural Networks (NN), which aims to work like the human brain. Computing cells (comparable to neurons) each perform a simple operation and interact with each other to make a decision (O' Mahony et al., 2016). ML, as the name suggests, is about learning (with or without supervision) using a NN. Due to advances in device capabilities (e.g., computing power) this method has become widely researched - there are many ML algorithms that can reach a high accuracy - Microsoft's model (ResNet) won the 2016 ImageNet competition with 96.4% accuracy (Le, 2021).

When it comes to ML models, not only do we have to consider execution time but also training time. With these models, usually a higher accuracy means a longer training time and more training data. When choosing a model for any application, we must consider the data set, the accuracy we need and the training time we have.

One-stage and two-stage algorithms

Object detection methods can be divided into these two categories. Two-stage algorithms tend to create region proposals as their first stage, before then looking over these regions to detect objects. Examples of multi-stage algorithms include RCNN, Faster RCNN, FPN and G-RCNN. These tend to sacrifice execution time for accuracy.

One-stage algorithms miss the region proposal process and instead skip straight to detecting the objects in one pass of the NN. Examples of single-stage algorithms include YOLOv4, SDD, RetinaNet and Fast-D. These are generally faster but less accurate than two-stage algorithms.

2.1.5 Recent ML object detection algorithms

G-RCNN

Proposed by J. Wang and Hu (2021), this method (Gated Recurrent Convolution Neural Network - GRCNN) introduces gates to the RCNN method to "control the amount of context information inputting to the neurons" (J. Wang & Hu, 2021) - the goal of this is to make the receptive fields of neurons adaptive, which is more consistent with how neurons work biologically. The results of this are a model that outperforms RCNN for object recognition, text recognition and object detection with accuracy on par or higher than state of the art models.

YOLOR and YOLOv4

YOLO (You Only Look Once) was first developed by Redmon et al. (2016) and is based on a single Convolutional Neural Network (CNN) - it predicts bounding boxes and probabilities in one frame (hence the name) making it an extremely fast algorithm.

YOLOv4, builds upon YOLOv3, improving both average precision (AP) and frames per second (FPS) by 10% and 12% respectively (Bochkovskiy et al., 2020).

YOLOR (You Only Learn One Representation) was published by C.-Y. Wang et al. (2021). It builds upon YOLOv4-CSP by introducing the idea of implicit knowledge. Implicit knowledge is information that is learnt subconsciously - in the context of a ML model, implicit knowledge would be more detailed information (for example, a vehicle may be identified, but the implicit knowledge would be recognising the wheels, windows, etc). This method produces accuracy comparable with YOLOv4-P7, but with an interference speed that is 88% less.

2.2 Depth estimation with cameras

As discussed above, sensors such as LiDAR and RADAR are used to provide depth information.

Cameras do not provide depth information, so methods have to be used to calculate this from the information that they do provide.

2.2.1 Monocular systems

Monocular systems use only one camera. From this image, it is possible to estimate depth by looking at features such as gradients and changes in texture, similarly to how humans can estimate depth with one eye closed. These are often "very dependent on the assumptions imposed or specific training

scenarios that strongly limit the performance" (Arenado et al., 2014). Saxena et al. (2007) use a monocular system to estimate depth by using texture variations, texture gradients, and colour. One of the advantages of a monocular system over a stereo system is that the cost of the hardware is less. Additionally, monocular systems do not need to carry out the process of feature matching, a computationally expensive step that stereo systems use.

A disadvantage of a monocular system is that they rely on texture, lighting, etc., meaning they tend to be less accurate than a stereo system.

2.2.2 Stereo vision systems

Stereo vision systems use two cameras to capture images of an object from two different perspectives. Stereo vision is a well researched and well known method that is used for many applications - for example, rovers such as NASA's Perseverance (NASA, n.d.). One of the main advantages of a stereo vision system is that the stereoscopic images created can be used to calculate size and distance.

One of the disadvantages of using stereo cameras is that the process can be computationally expensive. Both initial camera calibration and matching the 2 images requires "considerable computational complexity" (Zhe et al., 2020), which leads to a longer execution time.

The stereoscopic images are created from the known relative camera locations and the disparity between the two images. The cameras are usually set up in a verged or non-verged configuration.

When creating stereoscopic images, we observe the displacement caused by the different positions of the cameras. This displacement is called parallax and is split into horizontal and vertical. Horizontal parallax is generally what is used to measure distance, size, etc., while vertical parallax can cause issues in these measurements. To avoid vertical parallax, most systems use a non-verged system where the optical axes are parallel to each other (see figure

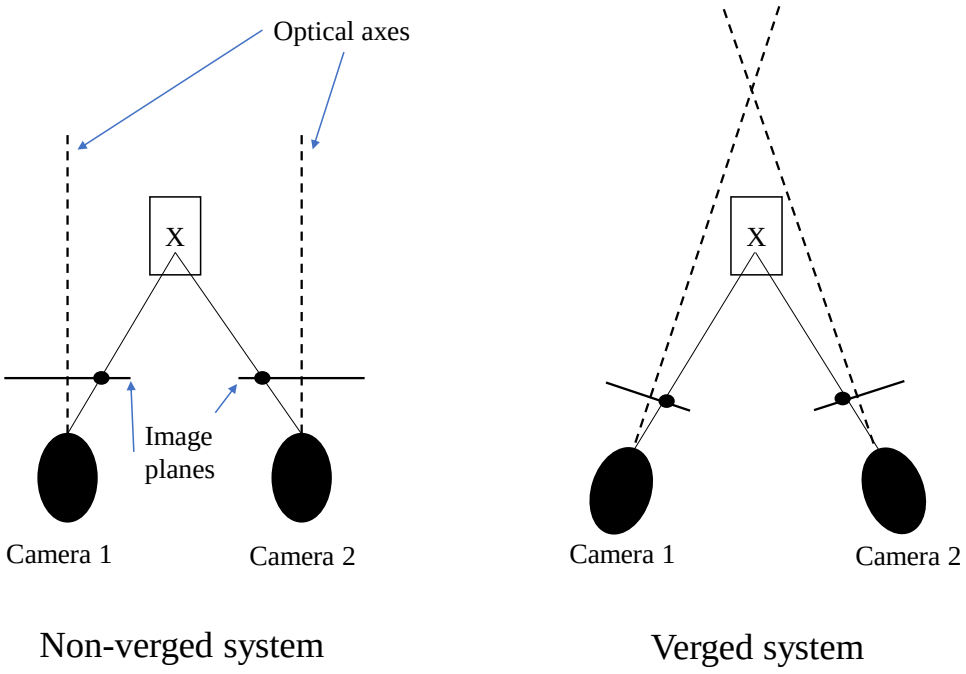


Figure 2.2: Non-verged and verged stereo cameras

2.2 for an example of verged and non-verged systems).

Woods et al. (2002) discuss image distortions in stereoscopic systems in high detail in their paper.

If the focal length and baseline of the cameras is known, and the cameras are parallel, a simple calculation of depth is $depth = \frac{baseline * focallength}{disparity}$

Figure 2.3 shows a typical example of a stereo camera system. The baseline, b, represents the distance between the 2 cameras. The height, h, is the distance that we aim to calculate. The angles θ and σ can be calculated by calculating the disparity between the two images. As $\theta + \sigma + a = 180$, we can also calculate a easily using:

$$a = 180 - \theta - \sigma \tag{2.1}$$

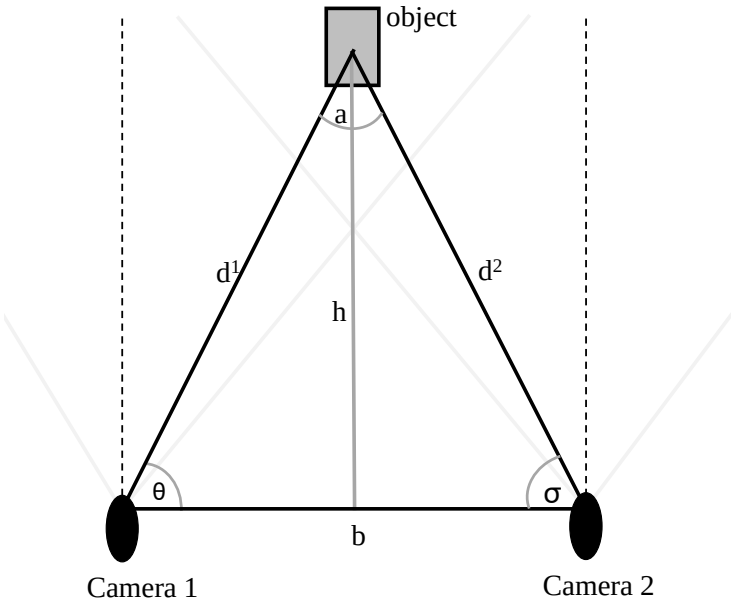


Figure 2.3: Traditional stereo example

The law of sines gives us:

$$\frac{d_1}{Sin(\sigma)} = \frac{d_2}{Sin(\theta)} = \frac{b}{Sin(a)} \tag{2.2}$$

$$d_1 = \frac{bSin(\sigma)}{Sin(a)} \tag{2.3}$$

$$d_2 = \frac{bSin(\theta)}{Sin(a)} \tag{2.4}$$

Treating figure 2.3 as 2 separate triangles split by h, we can use trigonometric functions to calculate:

$$Sin(\theta) = \frac{h}{d_1} \tag{2.5}$$

$$h = d_1 Sin(\theta) \tag{2.6}$$

$$Sin(\sigma) = \frac{h}{d_2} \tag{2.7}$$

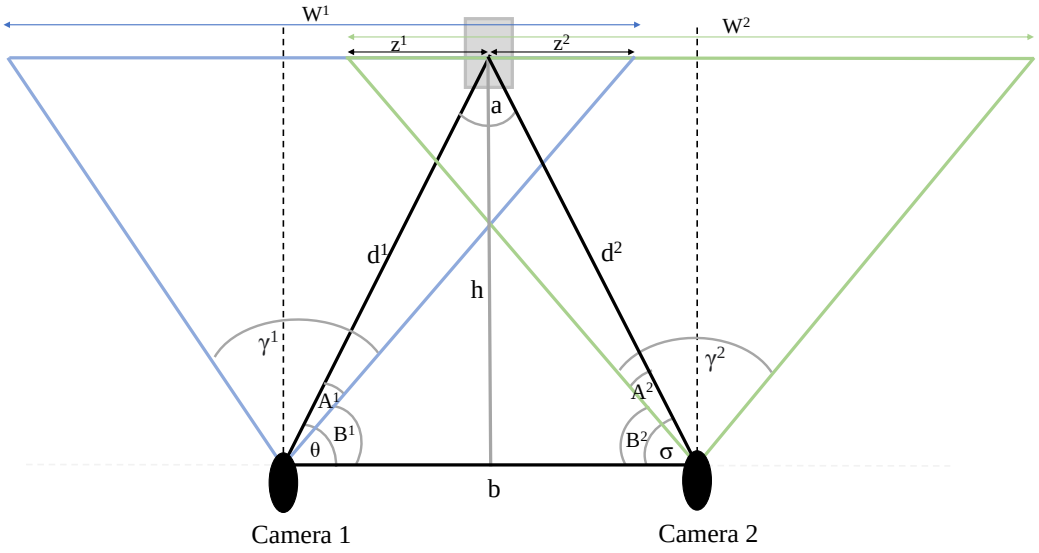


Figure 2.4: A closer look at a stereo example

$$h = d_2 Sin(\sigma) \tag{2.8}$$

Substituting equation 2.3 into 2.6 we can calculate:

$$h = \frac{bSin(\sigma)Sin(\theta)}{Sin(a)} \tag{2.9}$$

Calculating the disparity requires looking at figure 2.3 in more detail - see figure 2.4.

γ_1 and γ_2 represent the field of view (FOV) of the cameras. B_1 and B_2 show the angle between the baseline and the FOV of the cameras - these angles are equal on either side of each camera and can be calculated using $B = \frac{180 - \gamma}{2}$

W_1 and W_2 represent the size of the images captured by the cameras (horizontally and in pixels). Additionally, Z_1 and Z_2 represent the size of the crossover between the cameras. Notice that $\theta = A_1 + B_1$ and $\sigma = A_2 + B_2$.

A_1 and A_2 can be calculated by multiplying the location of the object in the

frame (Z_1 and Z_2 - these are in pixels) by the angle that one pixel is equal to.

To calculate this angle (P), we must divide the FOV of the camera (γ) by the number of pixels across the frame (W).

$$P = \frac{\gamma}{W} \tag{2.10}$$

So multiplying the location of the object by the angle per pixel gives us:

$$\theta = Z_1 \frac{\gamma_1}{W_1} + B_1 \tag{2.11}$$

$$\sigma = Z_2 \frac{\gamma_2}{W_2} + B_2 \tag{2.12}$$

Filling equations 2.12, 2.11 and 2.1 into 2.9, we get:

$$H = \frac{bSin(Z_2 \frac{\gamma_2}{W_2} + B_2)Sin(Z_1 \frac{\gamma_1}{W_1} + B_1)}{Sin(180 - (Z_1 \frac{\gamma_1}{W_1} + B_1) - (Z_2 \frac{\gamma_2}{W_2} + B_2))} \tag{2.13}$$

The above shows how to calculate depth using a stereo camera system.

2.2.3 Camera Calibration

Calibration is an important step which involves taking images of a known object and from these calculating the camera intrinsics, which is then used to remove distortion from the image (figure 2.6).

Knowing the characteristics of the camera(s) is vital if carrying out a process such as calculating distance/size from images. They are used to account for distortion of the image caused by the camera lens. These characteristics, such as focal length and lens distortion, are called camera intrinsics. The two types of distortion faced by lenses are radial and tangential. Radial distortion can come in the form of barrel distortion or pincushion distortion (figure 2.5), both caused by the light rays bending more at the lens edges

than the centre. Tangential distortion occurs when the picture screen or sensor is at an angle with respect to the lens (Dulari Bhatt, 2021).

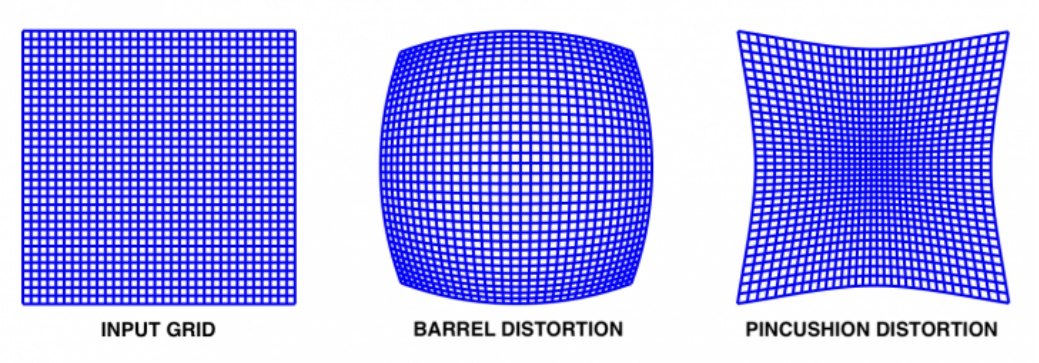


Figure 2.5: Radial distortion examples

Source: Dulari Bhatt (2021)

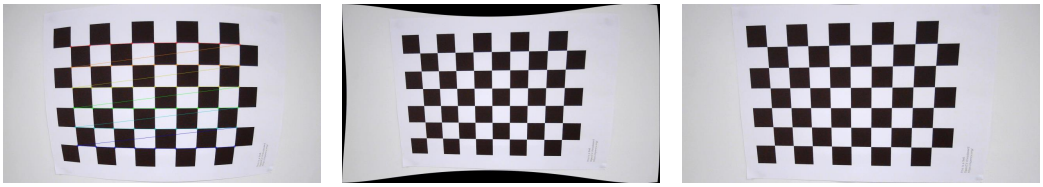


Figure 2.6: The process of undistorting an image

2.2.4 Rectification

While distortion is removed from the images using the camera intrinsics, rectification uses the camera extrinsics. The extrinsics represent the camera's location in 3D space - it includes a rotation matrix and a translation matrix. Both of these matrices are used to align the y axis in both images, so that during processes such as correspondence searching, each pixel will be found in the same column in both images, simplifying the matching process (figure 2.7).

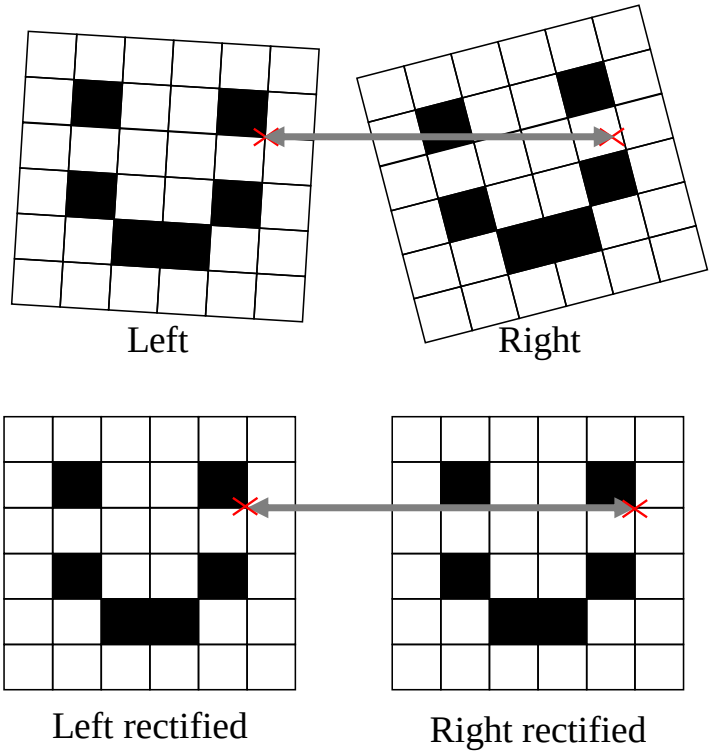


Figure 2.7: An example of stereo image rectification

2.2.5 Correspondence searching

One of the main steps when using stereo vision is point matching between the 2 images. This is the process of finding the same part of image A in image B – you can then calculate the disparity between the 2 images, which is later used to find the distance to the object. There are 2 main classes of algorithms for this, feature-based and region-based.

Feature-based matching

The aim of these algorithms is to find correspondence by matching sets of features, which are usually measured using vectors. This method tends to be efficient, however finding the sets of features can be difficult.

Feature-based matching tends to produce sparse disparity maps with a high accuracy (if the features are clear enough).

Region-based matching

This method takes a region of an image and attempts to match it with the other - this is generally done by minimising or maximising the region using method such as minimising the sum of the squared difference (SSD) or maximising the normalised cross-correlation (NCC).

Region-based is easier to implement but can be more sensitive to changes in lighting and texture.

2.3 Conclusion

Stereo cameras have been chosen as this application requires that the distance estimation be as accurate as possible and, while mono approaches can be accurate, a stereo approach should provide a more robust, reliable and accurate system.

ML has been chosen for object detection due to the low computing time of single-stage algorithms and the high accuracy they can provide. From the research carried out, the extra accuracy provided seems worth the additional time cost these methods have over traditional methods.

ORB feature detector was chosen due to its performance and speed, matching Scale Invariant Feature Transform (SIFT) in performance while being almost twice as fast (Tyagi, 2019). Additionally, ORB is free to use, unlike SIFT and Speeded-Up Robust Features (SURF), which are patented.

2.3.1 Choosing a model

There are many datasets available for training models to detect vehicles, such as KITTI, Oxford RobotCar, Cityscapes, etc. As access to reliable, labelled datasets is not an issue and supervised learning tends to be more accurate (Delua, 2021), a labelled dataset will be chosen.

The dataset chosen for this project is Berkeley Deep Drive (Yu et al., 2020)

- this was chosen as it provides over 100,000 labelled images at varying times of day.

The main factors when choosing a ML model are accuracy, speed, computing power and training time. As this application is real-time, speed will be prioritised. However, a high level of accuracy is needed to avoid any false readings - balancing speed and accuracy will be crucial.

Training time is not much of a concern - 70,000 images are used for training, however if hundreds of thousands or even millions of images were used, this would be more of a concern. Despite this, it will still be a consideration in case of further development of this project.

Zaidi et al. (2021) provide the comparison of object detectors shown in figure 2.8 in their paper.

Model	Year	Backbone	Size	$AP_{[0.5:0.95]}$	$AP_{0.5}$	FPS
R-CNN*	2014	AlexNet	224	-	58.50%	∼0.02
SPP-Net*	2015	ZF-5	Variable	-	59.20%	∼0.23
Fast R-CNN*	2015	VGG-16	Variable	-	65.70%	∼0.43
Faster R-CNN*	2016	VGG-16	600	-	67.00%	5
R-FCN	2016	ResNet-101	600	31.50%	53.20%	∼3
FPN	2017	ResNet-101	800	36.20%	59.10%	5
Mask R-CNN	2018	ResNeXt-101-FPN	800	39.80%	62.30%	5
DetectoRS	2020	ResNeXt-101	1333	53.30%	71.60%	∼4
YOLO*	2015	(Modified) GoogLeNet	448	-	57.90%	45
SSD	2016	VGG-16	300	23.20%	41.20%	46
YOLOv2	2016	DarkNet-19	352	21.60%	44.00%	81
RetinaNet	2018	ResNet-101-FPN	400	31.90%	49.50%	12
YOLOv3	2018	DarkNet-53	320	28.20%	51.50%	45
CenterNet	2019	Hourglass-104	512	42.10%	61.10%	7.8
EfficientDet-D2	2020	Efficient-B2	768	43.00%	62.30%	41.7
YOLOv4	2020	CSPDarkNet-53	512	43.00%	64.90%	31
Swin-L	2021	HTC++	-	57.70%	-	-

[a]Models marked with * are compared on PASCAL VOC 2012, while others on MS COCO.Rows colored gray are real-time detectors (>30 FPS).

Figure 2.8: A comparison of object detectors on the COCO and Pascal VOC datasets

Source: Zaidi et al. (2021)

$AP_{0.5}$ being the average precision with an Intersection Over Union (IOU) of 0.5. The IOU is equal to the area of overlap (between the bounding box prediction and the ground truth) divided by the area of union of the two boxes.

Removing any model with an $AP_{0.5}$ of less than 50% (lower accuracy) and an FPS of less than 30 (lower speed), gives us table 2.1

Model	$AP_{0.5}$	FPS
YOLO	57.90%	45
YOLOv3	51.50%	45
EfficientDet-D2	62.30%	41.7
YOLOv4	64.90%	31

Table 2.1: Object detectors with $AP_{0.5}$ larger than 0.5 and FPS larger than 30

Source: Zaidi et al. (2021)

Nepal and Eslamiat (2022) find that YOLOv5 achieves a slightly higher precision than YOLOv4 (0.707 and 0.69) but has a slightly lower FPS (58.82 compared with 59).

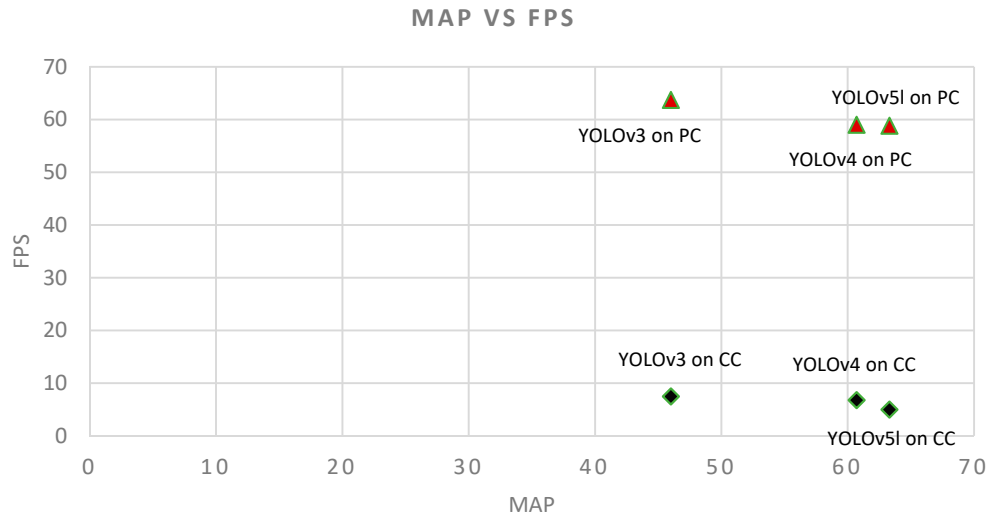


Figure 2.9: MAP vs FPS for object detectors on PC and Companion Computer (CC)

Source: Nepal and Eslamiat (2022)

Figure 2.9 shows the results that Nepal and Eslamiat obtained from their tests - they find YOLOv5l most suitable for their application.

Ge et al. (2021) introduce YOLOX in their paper. Starting with YOLOv3 as a baseline, the three main features of YOLOX are decoupled head, it is anchor-free and it has a new label assigning method. Table 2.2 shows their results when comparing YOLOX to YOLOv5 on a Tesla V100 at 640x640 and with a batch size of 1.

Model	AP(%)	Parameters	GFLOPs	Latency
YOLOv5-S	36.7	7.3 M	17.1	8.7 ms
YOLOX-S	39.6 (+2.9)	9.0 M	26.8	9.8 ms
YOLOv5-M	44.5	21.4 M	51.4	11.1 ms
YOLOX-M	46.4 (+1.9)	25.3 M	73.8	12.3 ms
YOLOv5-L	48.2	47.1 M	115.6	13.7 ms
YOLOX-L	50.0(+1.8)	54.2 M	155.6	14.5 ms
YOLOv5-X	50.4	87.8 M	219.0	16.0 ms
YOLOX-X	51.2 (+0.8)	99.1 M	281.9	17.3 ms

Table 2.2: Comparison of YOLOX to YOLOv5

Source: Ge et al. (2021)

From table 2.2 we can see that YOLOX scales both the AP and the latency. Compared to YOLOv5-S, YOLOX improves AP by 2.9% but also increases latency by 1.1ms.

Ge et al. (2021) also gives a comparison of popular models trained on the COCO data set which can be seen in table 2.3.

YOLOv5-M achieves a higher FPS but lower precision than YOLOv4-CSP (90.1 vs 73 and 44.5% vs 47.5%). YOLOv5-M gains 17.1 frames per second at a cost of only 3% precision.

YOLOv5 will be used for this application. As discussed above, it strikes a good balance between FPS and accuracy while also keeping latency time down.

Method	Backbone	Size	FPS	AP (%)	AP_{50}
YOLOv3 + ASFF	Darknet-53	608	45.5	42.4	63.0
YOLOv3 + ASFF	Darknet-53	800	29.4	43.9	64.1
EfficientDet-D0	Efficient-B0	512	98.0	33.8	52.2
EfficientDet-D1	Efficient-B1	640	74.1	39.6	58.6
EfficientDet-D2	Efficient-B2	768	56.5	43.0	62.3
EfficientDet-D3	Efficient-B3	896	34.5	45.8	65.0
PP-YOLOv2	ResNet50-vd-dcn	640	68.9	49.5	68.2
PP-YOLOv2	ResNet101-vd-dcn	640	50.3	50.3	69.0
YOLOv4	CSPDarknet-53	608	62.0	43.5	65.7
YOLOv4-CSP	Modified CSP	640	73.0	47.5	66.2
YOLOv3-ultralytics	Darknet-53	640	95.2	44.3	64.6
YOLOv5-M	Modified CSP v5	640	90.1	44.5	63.1
YOLOv5-L	Modified CSP v5	640	73.0	48.2	66.9
YOLOv5-X	Modified CSP v5	640	62.5	50.4	68.8
YOLOX-DarkNet53	Darknet-53	640	90.1	47.4	67.3
YOLOX-M	Modified CSP v5	640	81.3	46.4	65.4
YOLOX-L	Modified CSP v5	640	69.0	50.0	68.5
YOLOX-X	Modified CSP v5	640	57.8	51.2	69.6

Table 2.3: Comparison of object detectors

Source: Ge et al. (2021)

Ultralytics provides a comparison of the YOLOv5 models trained on the COCO dataset as shown in table 2.4.

Looking at mean average precision with an intersection over union of 0.5 ($mAP_{0.5}$), we see the largest increase between YOLOv5n (nano) and YOLOv5s (small), going from 45.7 to 56.8 (11.1 difference).

Looking at Speed V100 (Tesla V100), we see nano and small have similar results with 6.3ms and 6.4ms respectively, while YOLOv5m (medium) jumps to 8.2ms.

Model	size (pixels)	$mAP_{0.5:0.95}$	$mAP_{0.5}$	Speed CPU	Speed V100	params	FLOPs
YOLOv5n	640	28.0	45.7	45 ms	6.3 ms	1.9 M	4.5
YOLOv5s	640	37.4	56.8	98 ms	6.4 ms	7.2 M	16.5
YOLOv5m	640	45.4	64.1	224 ms	8.2 ms	21.2 M	49.0
YOLOv5l	640	49.0	67.3	430 ms	10.1 ms	46.5 M	109.1
YOLOv5x	640	50.7	68.9	766 ms	12.1 ms	86.7 M	205.7

Table 2.4: YOLO model comparison - trained on the COCO data set

Source: Ultralytics (2020)

For floating point operations per second (FLOPs), there is an increase of 12 from nano to small and 32.5 from small to medium.

From everything discussed above, keeping precision, speed and performance in mind, YOLOv5s increases precision by the largest amount while increasing speed and performance by the smallest amount. The jump from YOLOv5s to YOLOv5m may increase precision by 7.3, but this is at the cost of speed and performance. Due to this, YOLOv5s will be used for this application.

Chapter 3

Methodology

3.1 Iterative method

This method splits the development process into iterations, where simple versions of the requirements are added before reviewing and then building on this.

Advantages

- Gives working functionality early in the process
- Regular feedback and testing

Disadvantages

- New requirements can be added but not ideal
- Not suitable for smaller projects

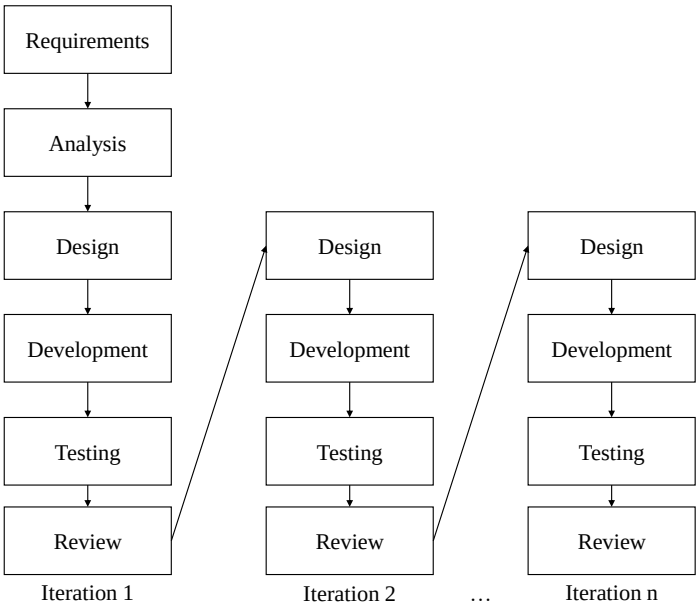


Figure 3.1: Iterative model diagram

3.2 Incremental method

This method breaks the development process into increments, with each increment building on the working product produced by the previous increment. For example, completing one feature of the software before moving on to the next.

Advantages

- Software produced quickly
- Regular testing and feedback
- Requirements can be easily added if needed

Disadvantages

- Requires good planning
- Has a higher cost than waterfall

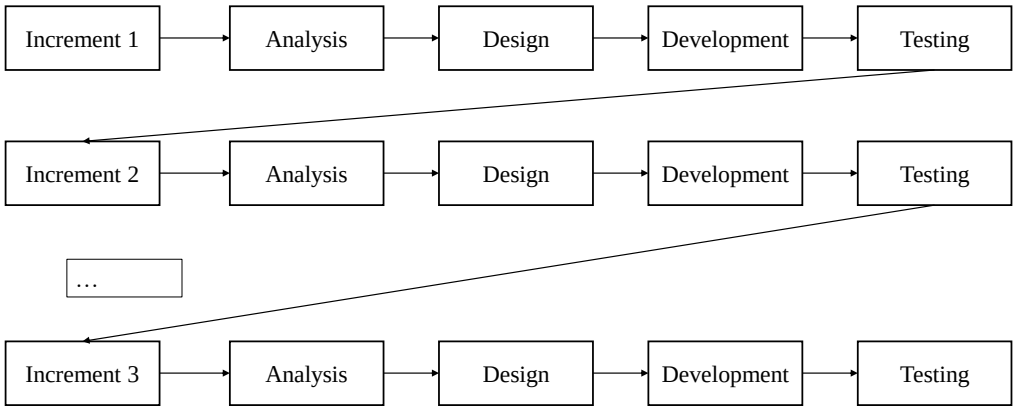


Figure 3.2: Incremental model diagram

3.3 SCRUM method

The term 'scrum' in this context was originally used by Takeuchi and Nonaka (1986). Ken Schwaber and Jeff Sutherland worked together to develop this method and presented a paper on it in 1997 (Schwaber, 1997).

Scrum is an iterative method that uses "a time box approach where development cycles known as Sprints take no more than 4 weeks" (Despa, 2014) - feedback is given at the end of each sprint to guide the next sprint. In addition to this, 15 minute daily meetings take place (daily scrum) to plan for that day. Every member of the team is involved in the organisation process but the project is overseen by the Scrum Master.

Advantages

- Divides projects into more manageable sections
- Progress tracked daily through daily meetings
- Generally produces a product quickly
- Regular feedback

Disadvantages

- Usually requires a well trained, experienced team
- Can be difficult to make cost estimates (Despa, 2014)

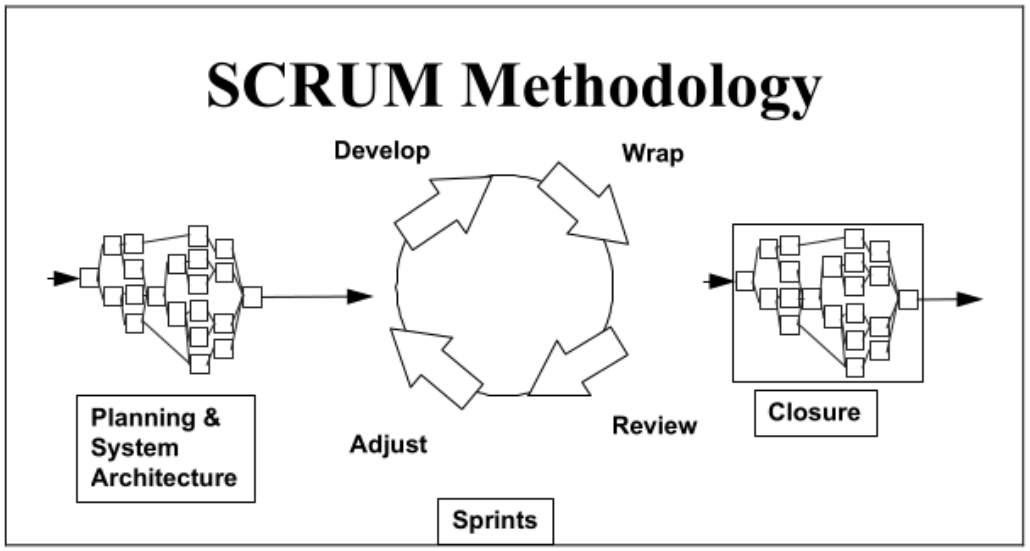


Figure 3.3: SCRUM model diagram

Source: Schwaber (1997)

3.4 Waterfall method

First detailed by Winston Royce in 1970, this linear process is one of the oldest software development models. This method is "predictable and values rigorous software planning and architecture" (Despa, 2014).

Advantages

- Straightforward and simple
- With a non-complex project with well known requirements, it means the stages are set out and easy to follow, giving a good plan

Disadvantages

- Time consuming and not flexible

- A linear process - you do not move back in stages and you cannot move forward with a stage unless the current one is complete
- Does not work well for time sensitive projects or projects where the requirements may change

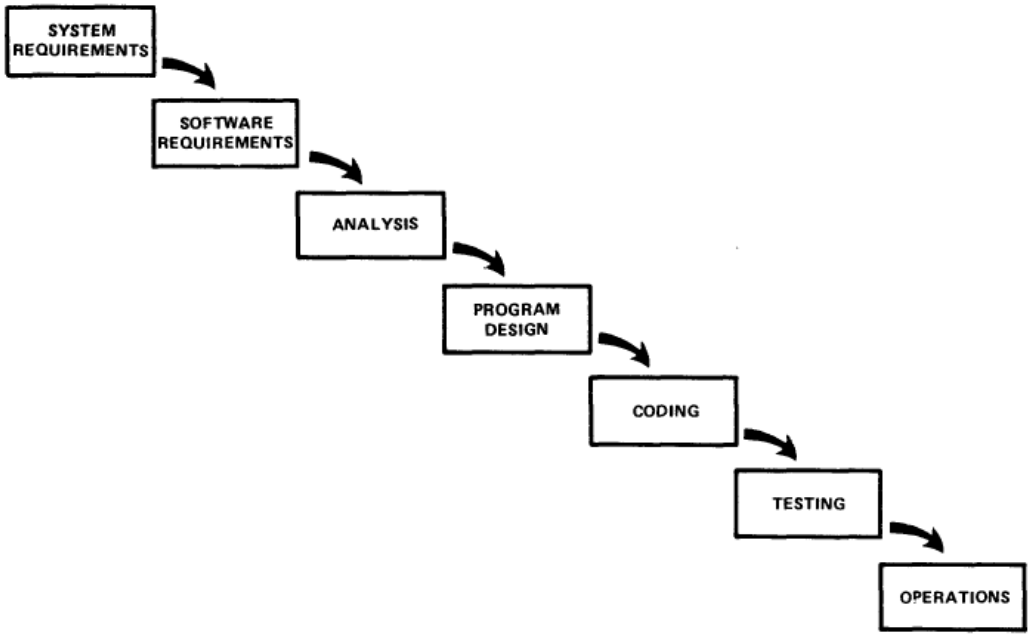


Figure 3.4: Waterfall model diagram

Source: Royce (1970)

3.5 Chosen method

I have decided to choose the incremental approach for this project. This is due to the regular testing and feedback it provides as well as the ability to easily add requirements if needed. An iterative approach was considered, however it is less suitable for smaller projects and, while it is possible to add extra requirements during development, this is easier with an incremental approach. SCRUM was not chosen as it is more suitable for a development team and does not lend itself to individual development as well. Waterfall

was not chosen due to the rigid structure - further requirements would be difficult to add.

Chapter 4

Software Requirements

The requirements for this project have been acquired from the literature review and through analysis of automotive RADAR and LiDAR sensors. As we are testing the feasibility of cameras replacing these sensors, many of the non-functional requirements come from the statistics of these sensors.

4.1 Software overview

This software is being developed to test the thesis that a camera can provide a similar result to LiDAR and RADAR at a lower cost - this would be with the purpose of bringing down manufacturing costs (savings that could be passed to the consumer) as well as reducing the number of computer chips needed during a time of shortage.

Table 4.1 recaps on the advantages and disadvantages gathered from the literature review.

	Advantages	Disadvantages
LiDAR	- High resolution - High range (some models detecting vehicles at up to 250m) - Option of 360 degree sensors - Low interference when multiple sensors close	- Expensive to purchase and maintain (especially for 360 degree sensors) - Some smaller objects can be undetected due to size - Poor discrimination of contrast from wet to dry surfaces - Draws more power than RADAR - Influenced by weather conditions
RADAR	- Mostly unaffected by poor weather conditions - More affordable than LiDAR	- RADAR attenuation (weakening of signal due to rain, etc.) - Low resolution - Interference between multiple sensors can be an issue

Table 4.1: Advantages and disadvantages of LiDAR and RADAR

4.2 Audience and intended use

This project focuses on vehicle autonomy applications - a fully developed product would be used in vehicles to provide data that the various autonomous functions use. It focuses mainly on vehicle recognition, but will be built in a way that allows for potential further development. Therefore, the main audience for this product are vehicle manufacturers. While a user interface is not necessary for the final product (as it is intended to be integrated with other systems), one is being added for testing and demonstration purposes.

4.3 Assumptions

- Cameras are fitted behind the windscreen of the vehicle
- Cameras are fixed relative to each other at a consistent baseline
- Target audience are knowledgeable about object detection and stereo vision

4.4 Functional requirements

The MoSCoW method is used to rate our requirements in terms of importance. Each requirement is given a rating: Must, Should, Could or Would. Dai Clegg created this method for prioritising tasks during development (Pro-

ductPlan, n.d.).

4.4.1 F1 - Take an input of stereo images / videos

MoSCoW: Must

Stereo images/videos will be used for distance calculation.

4.4.2 F2 - Have the option to take an input of mono images/videos

MoSCoW: Should

Ensuring object detection works from a single mono input will help the robustness of the program, demonstrating that the object detection part of the program still functions in the occurrence of a fault with one of the two cameras. Distance estimation would not be possible in this scenario as there is no disparity to calculate.

This could also be helpful when integrating with other solutions which may only require the object detection functionality.

4.4.3 F3 - Have the option of using images taken from different cameras

MoSCoW: Could

This would serve as a good method of demonstrating the object detection function.

For this, the images will not be able to be altered to remove distortion as the camera intrinsics will not be known.

4.4.4 F4 - Detect and isolate key objects such as vehicles, traffic signs, etc.

MoSCoW: Must

This is key information that can be used for many applications if the software is integrated with other systems.

This information is also used in the distance calculation process.

4.4.5 F5 - Calibrate the camera to adjust the images for distortion

MoSCoW: Should

The cameras being used are affected by barrel distortion from the lens. Removing this should give a more accurate result when using pixel disparity to measure distance.

4.4.6 F6 - Calibrate the stereo cameras to keep matching points on a similar y axis

MoSCoW: Should

The two cameras may not be perfectly aligned - this can be due to how they are set up or how they are manufactured. Aligning the y axis for matching points will make the distance calculation easier and more accurate.

4.4.7 F7 - Calculate distance to detected objects

MoSCoW: Must

Calculating a distance value is a key feature of RADAR and LiDAR - without this our solution could not compare to these sensors.

4.4.8 F8 - Have a simple, informative user interface

MoSCoW: Should

The user interface serves as a method of demonstrating and testing the functionality easily and conveniently.

4.4.9 F9 - Process images/videos in real-time

MoSCoW: Should

This program will have to work in real-time so that it can provide information for changing situations.

As this is a showcase of the software, it should be able to do this, however, it is also something that has the potential to be added during further development.

Video will be processed without any pre-processing to show the ability of the system.

4.5 Non-functional requirements

4.5.1 NF1 - Detect objects at a distance of at least 200m

Blickfeld (n.d.) offers a long range automotive LiDAR sensor with a detection range of 200m. Their shorter range LiDAR sensor reaches 100-150m.

Luminar (n.d.) advertises a top range for object and vehicle detection of 250m.

4.5.2 NF2 - Detect objects with a high rate of precision

Figure 2.9 from section 2.3.1 shows a comparison of real-time object detectors. We will aim for at least 40% mean average precision based on this graph.

4.5.3 NF3 - Process images/frames at a speed faster than the average human reaction time

The processing time must be faster than a humans visual reaction time if it is to be relied on for autonomous functions. Stanisław Jurecki et al. (2017) found that the reaction time during a complex road incident ranged from 0.45s to 1.4s.

Jain et al. (2015), in a study on 120 medical students, found the average visual reaction time to be 247.60ms (+=18.54 sd).

Our processing time should be less than 1.4 seconds maximum and ideally close to 200ms.

4.5.4 NF4 - Produce a data stream of equal or less than a similar LiDAR system

Robert Bielby states that LiDAR produces data streams ranging from 20-100Mbit/s (Chris Mellor, 2020). The less data that has to be stored, the less that has to be spent on storage hardware.

4.5.5 NF5 - Have a total sensor cost of less than LiDAR or RADAR sensors

From our literature review we know that Luminar (n.d.) predicts a cost of £380-£761 ($500-$1000) for LiDAR per vehicle. RADAR costs range from £38-£152 ($50-$200) (Intellias Automotive, 2018).

4.5.6 NF6 - Calculate the camera matrix to be used for various functions

The camera matrix is used to calculate extrinsics, undistort images and provide the camera focal length.

4.5.7 NF7 - Calculate the camera extrinsics for the stereo calibration

The camera extrinsics are used when calibrating the stereo images.

4.5.8 NF8 - Use a trained YOLOv5s algorithm for object detection

This is a fast, accurate method of object detection provided you have access to training data.

4.5.9 NF9 - Well structured and commented code

The final product would be integrated with many other pieces of software - this would allow for easy integration with this other software.

4.5.10 NF10 - Use the image disparity to calculate the distance to objects

Distance = (camera baseline * focal length) / disparity

4.5.11 NF11 - Calculate distance to within an accuracy of 1cm

Luminar (n.d.) advertises a range precision of 1cm for their automotive LiDAR sensors.

Chapter 5

Design

5.1 Timeline

The deadline for this project is the 6th May 2022, which gives a limited time frame for the implementation. This has been considered when making decisions throughout the development process.

5.2 System architecture

Figure 5.1 shows a high level view of the functions of the system and how they interact.

The system can be broken down into 3 main steps: camera calibration, object detection and distance calculation. It has been designed in this way so that each of the 3 steps can be carried out individually as easily as possible. These steps are tied together using a GUI.

The camera calibration step is vital if accurate measurements are to be calculated using stereo images - it removes lens distortion and rectifies the images. For mono images, this step is not as important due to the fact that rectification is not needed and the object detection can still work well even with some

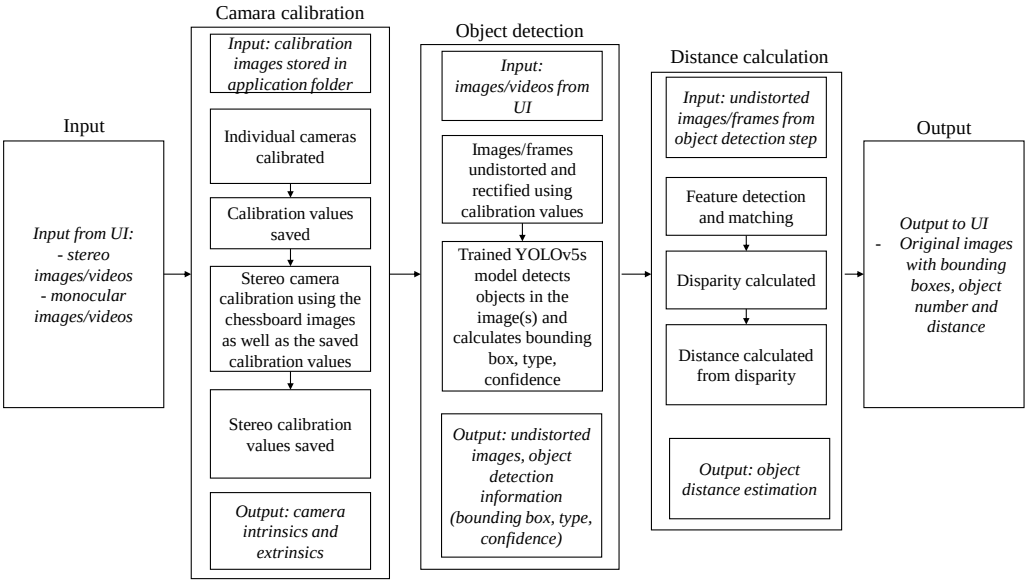


Figure 5.1: Function decomposition diagram

lens distortion. Calibration values are saved into the program files for later use - the program checks for these saved values before running the object detector, meaning calibration only has to be completed once. These values are saved as .CSV files instead of being stored as variables as they must be saved even while the program is closed (to avoid calibrating each time it is opened).

The object detection step provides the bounding box coordinates, object type (car, pedestrian, etc.) and a confidence score - these are all used later in the program. Images are undistorted using the calibration values calculated in the previous step and then passed to the detector. From the detector results, an object (of class 'detected') is made for each object detected in an image and stores each objects data - objects are created to simplify the program. A list of all objects in an image can then be generated - useful for feature matching later on.

For distance calculation, the two lists of objects (one for each image) are compared and an ORB feature detector is used to match the objects up and

remove any objects without a match - this is because any object that is only present in one image cannot have the disparity and distance calculated. While the list matching is not vital (as both images are likely to have the same number of objects), it is an important measure added to make the program more robust. Disparity is also calculated between each object pair at this point. Multiple methods of calculating disparity were used, but ORB in combination with brute force matching was found to produce the best results.

Justification on the use of stereo cameras, ML model and feature detector is specified in section 2.3.

5.3 Training attributes

Attributes that must be given when training our model include:

- Image size: depending on the size chosen, provided images will have to either be upscaled or downscaled.
  - Upscaling an image involves padding it - the model must learn that the padded area has no impact on the result. Upscaling can cause slower training times.
  - Downscaling an image means there is less information for the model to analyse - as a result it can lead to less accurate results. Downscaling can also lead to more generalised results.
- Batch size: this defines the number of samples that will be propagated through the network in one iteration. A smaller batch size tends to increase training speed. Generally, a large batch size causes degradation when it comes to generalisation - this is known as the "generalisation gap" phenomenon (Hoffer et al., 2018). However, Hoffer et al. (2018) suggest that the generalisation gap is related to the amount of updates and not the batch size. Batch size can be classified into:

  - Batch Gradient Descent (batch size = training data size)
  - Stochastic Gradient Descent (batch size = 1)
  - Mini-batch gradient descent (1 <batch size <training data size) (Brownlee, 2018)

- Epoch number: the number of passes through the data set. Iterating through the data set multiple times increases accuracy from the model (up to a point), but heavily impacts training time.
- Data location: the location of the YAML file that specifies the data location, as well as the classes of objects in the data set.
- Weight location: either use pre-trained weights or start from scratch. Pre-trained weights are often recommended for smaller data sets.

5.4 Language and development environment

Our chosen ML model, YOLOv5, is written in the Pytorch framework. Pytorch is an open source ML framework developed primarily by Facebook's AI research lab and initially released in 2016.

Many languages are now viable for ML applications, however for this project I have chosen Python due to the simplicity, flexibility and the number of libraries available. According to the 2021 Stack Overflow Developer Survey (Stack Overflow, 2021), 48.24% of developers use Python - putting it in third place behind HTML/CSS with 56.07% and JavaScript with 64.96%. Additionally, according to Karczewski (2020), 57% of data scientists use python and 33% prefer it over other languages. Scala, Java and R were also considered. Python has the advantage of being simple and consistent, however the main advantage is the number of libraries available. Libraries such as TensorFlow, NumPy and OpenCV give developers more convenient ways of implementing certain methods.

Additionally, Anaconda is being used to simplify package management. Ana-

conda was chosen as "in the data science sector, Anaconda is an industry staple" (Rohit Sharma, 2021).

5.5 User interface

The user interface has been designed keeping in mind Shneiderman's rules as well as Nielsen's heuristics. The aim is to create a simple, informative interface which showcases the functionality of the program. It should be easy to navigate, with a consistent layout and high contrast in the colours. To make it easy to navigate and read, there should only be one window open for the application at any one time - this includes error messages.

5.5.1 Opening Screen (1.0)

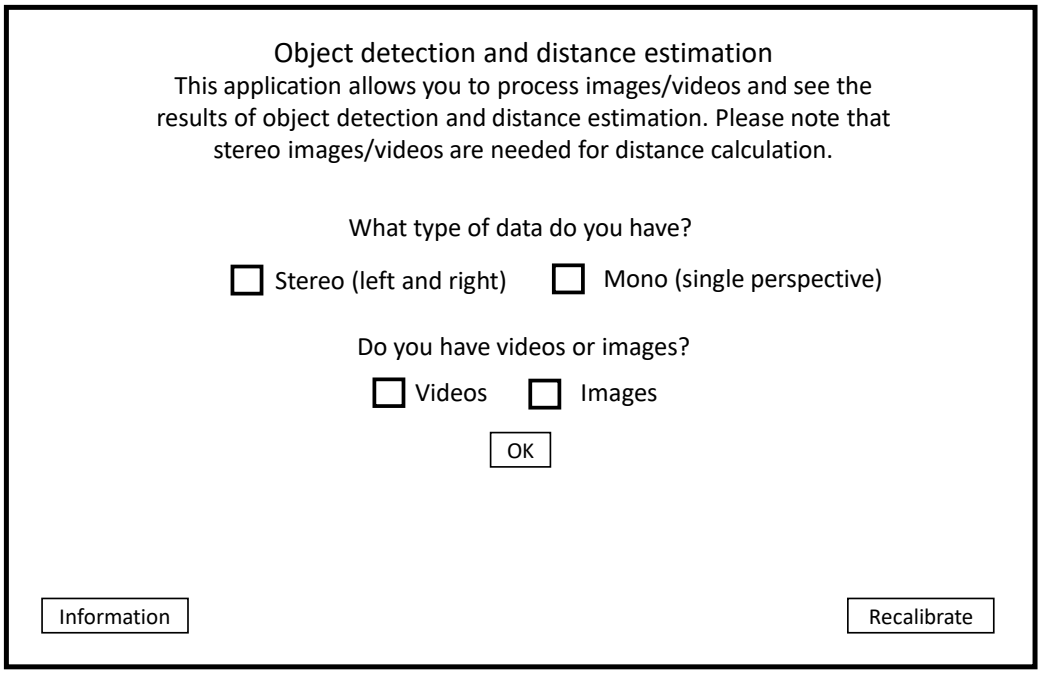


Figure 5.2: Screen 1.0 - the home screen

This starting screen is what the user is greeted with when loading the application. It includes a brief description of the application, with more infor-

mation available by clicking the 'Information' button at the bottom left of the screen. The 'Recalibrate' takes the user to the calibration screen - this should be used in the event that the camera hardware changes. To take the user to the correct screen, there are 2 questions on the type of data that they must answer. Once both questions are answered, the OK button takes the user to the appropriate screen.

5.5.2 Information Screen (1.1)

The information screen provides further details on the 3 main functions of the application. From here, the user can select 'Home' or 'Back' to go to the home screen.

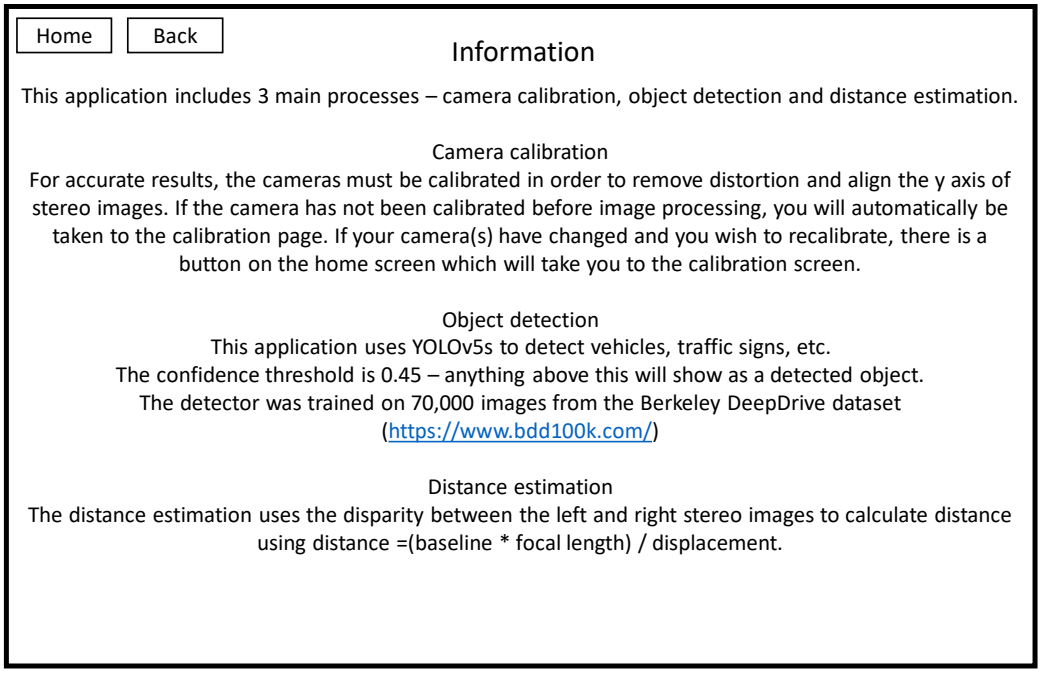


Figure 5.3: Screen 1.1 - the information screen

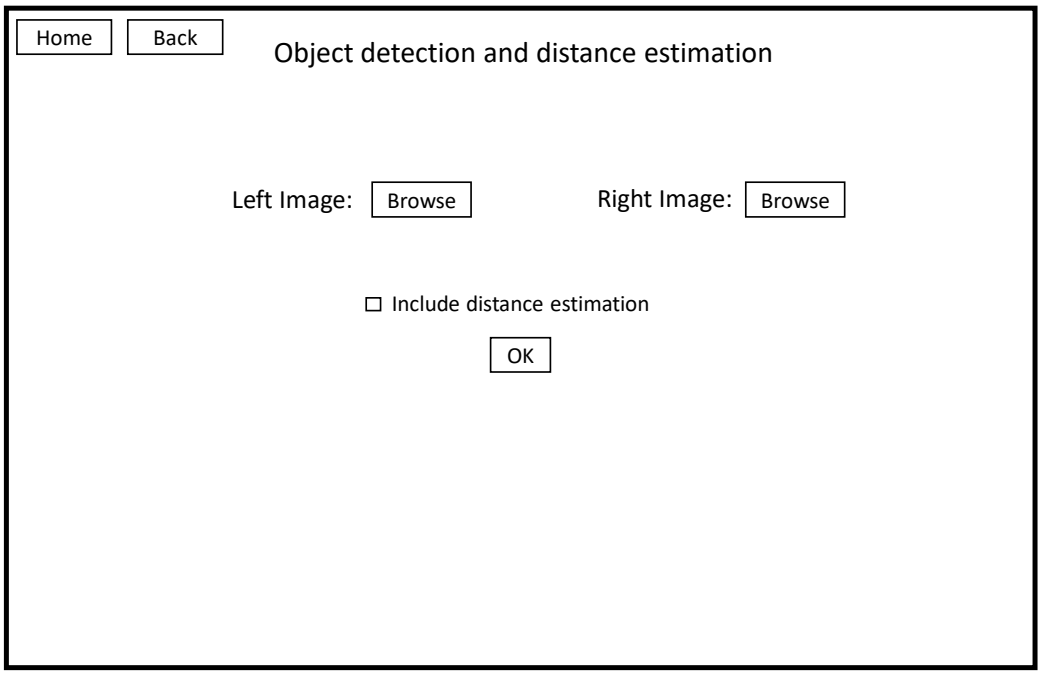


Figure 5.4: Screen 2.0 - the stereo image processing screen

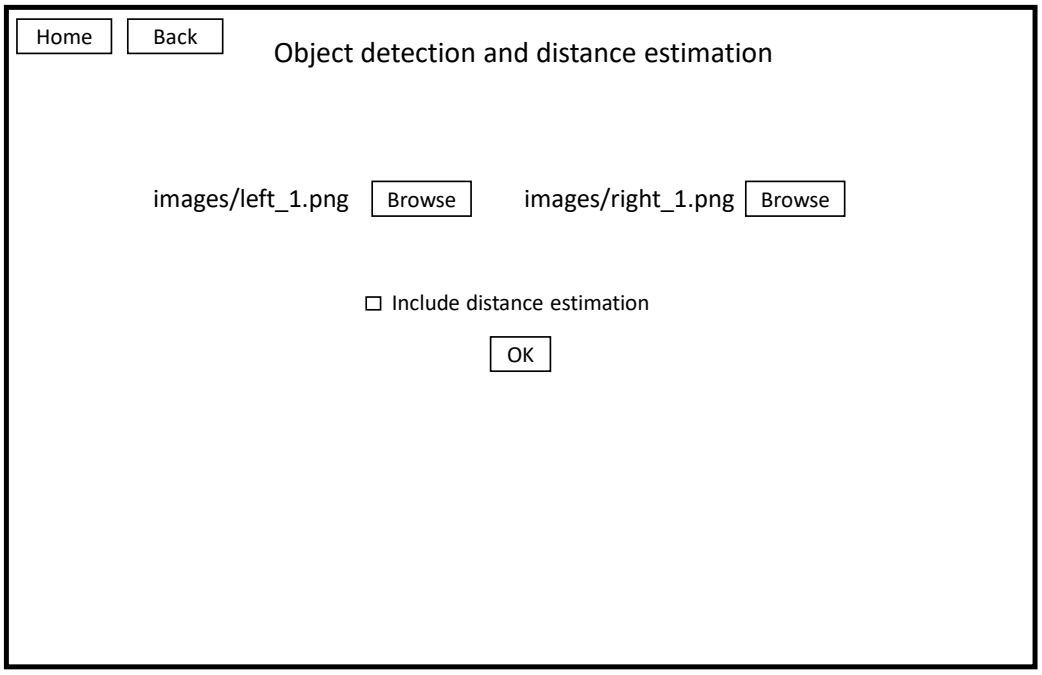
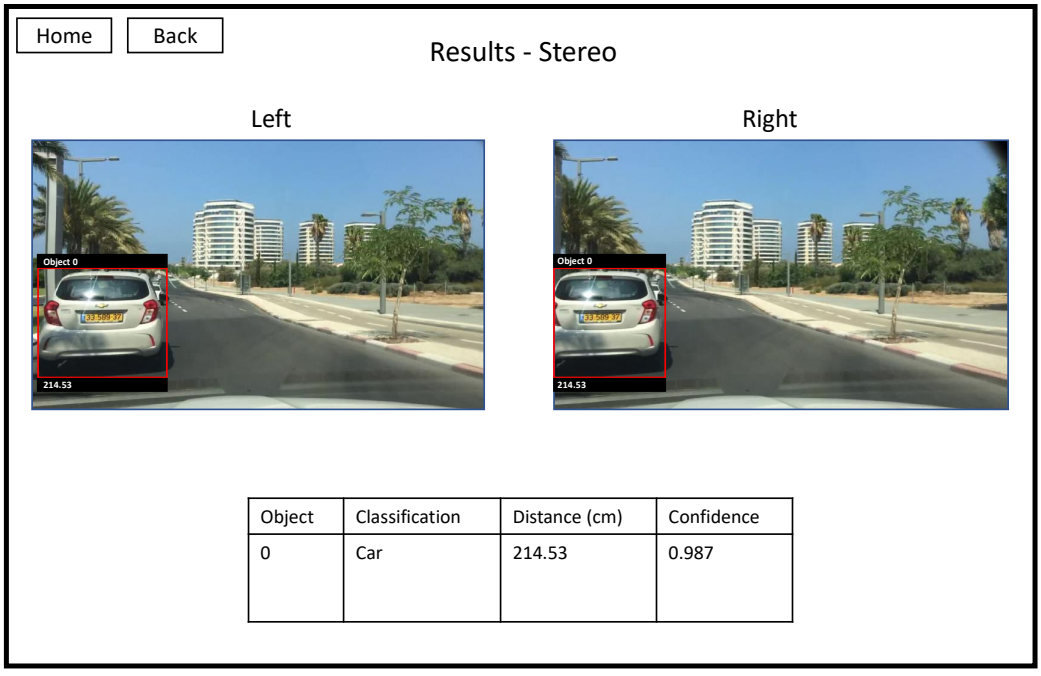


Figure 5.5: Screen 2.0 - images uploaded

5.5.3 Stereo image processing screen (2.0)

This screen will be shown if the user presses the options 'stereo' and 'images' in figure 5.2 - it is for processing stereo images with or without distance estimation, which can be toggled on or off via the tick box. Once an image has been uploaded via one of the two buttons available, visual confirmation with appear (the filepath of the image - see figure 5.5). Once both images have been uploaded, the user can select 'OK'. They are taken to screen 2.1 (figure 5.6).

5.5.4 Stereo image results screen (2.1) / Stereo video results screen (2.2)



Object	Classification	Distance (cm)	Confidence
0	Car	214.53	0.987

Figure 5.6: Screen 2.1 and 2.2 - the stereo results screen

Source: Car image from Yu et al. (2020)

This screen shows the objects detected in both images - the bounding box is surrounded by text (white text on black background) with the object number

(object 0, for example) and the distance. Additionally, the information is displayed in a table at the bottom of the page.

The video results screen is similar, but refreshes for each frame of the video to update the image shown and the table of objects.

5.5.5 Mono processing screen (3.0)

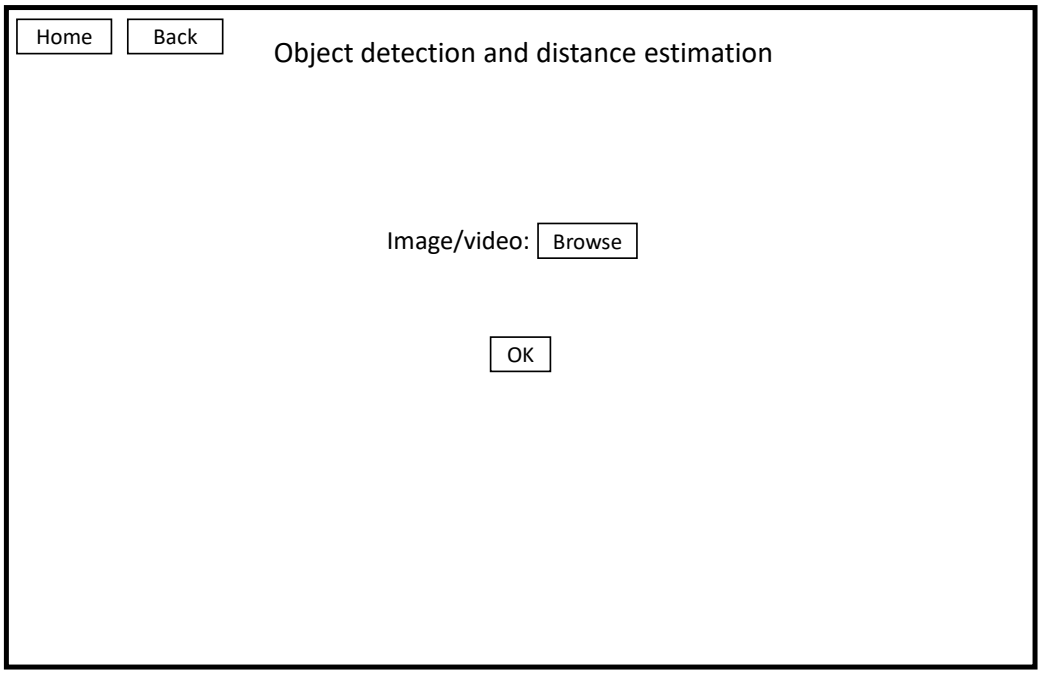


Figure 5.7: Screen 3.0 - the mono image processing screen

This screen is similar to the stereo processing screen, except it prompts for one video/image. Depending on the file type uploaded, when OK is clicked the user is taken either to the image or video results screen.

5.5.6 Mono image results screen (3.1) / Mono video results screen (3.2)

Similarly to the stereo video results, the mono video results refresh both the image and the table for each frame.

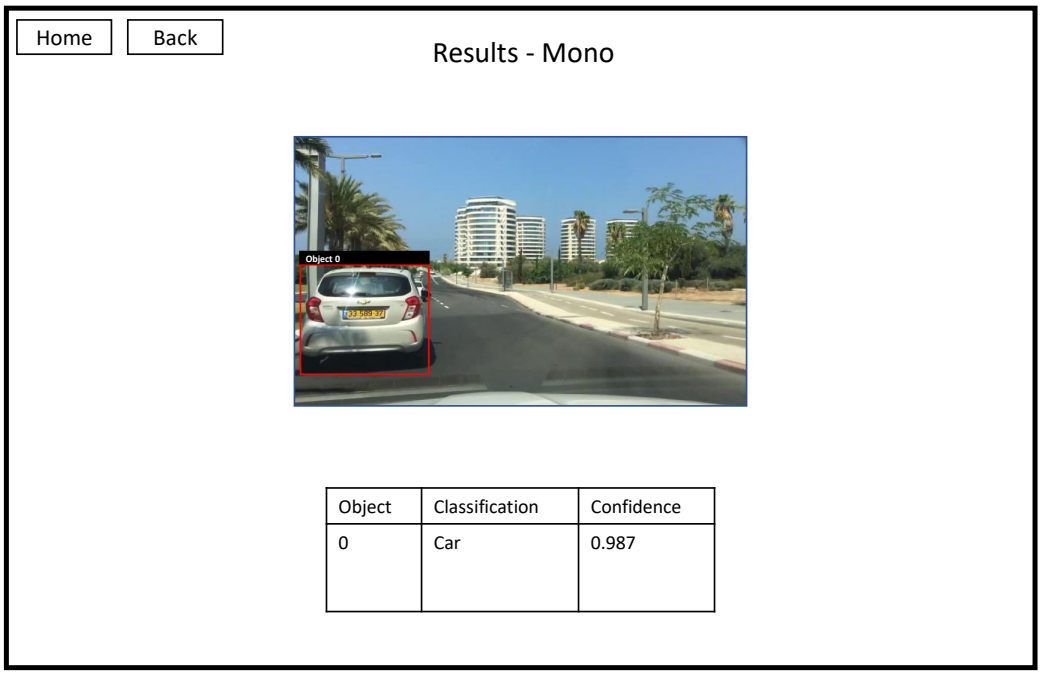


Figure 5.8: Screen 3.1 and 3.2 - the mono results screen

Source: Car image from Yu et al. (2020)

5.5.7 Camera calibration screen (4.0)

This screen is shown either if the system detects calibration has not been completed or if the user selects 'Recalibrate' on the home screen (figure 5.2). Selecting stereo or mono and then 'OK' will take the user to either screen 4.1 (figure 6.20) or 4.2 (figure 6.21).

Stereo calibration screen (4.1)

This screen displays a short explanation of the images needed for calibration, as well as a few examples. Once the images are placed in the correct folder, the user selects 'Calibrate'. Once calibration is complete, the Home screen is shown.

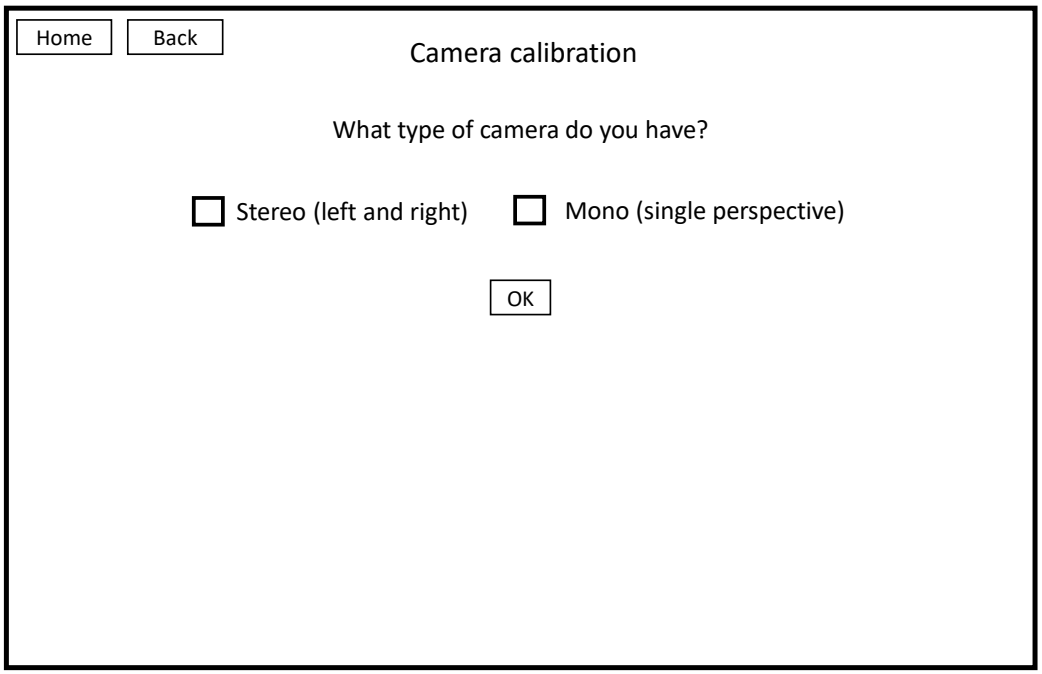


Figure 5.9: Screen 4.0 - the initial camera calibration screen

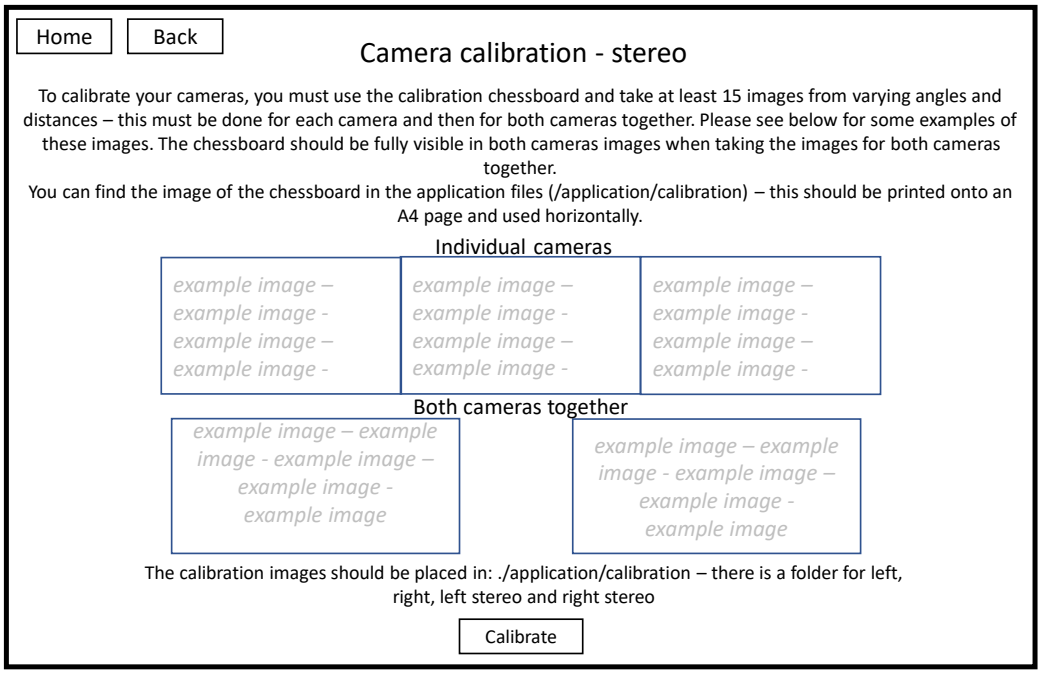


Figure 5.10: Screen 4.1 - the stereo calibration screen

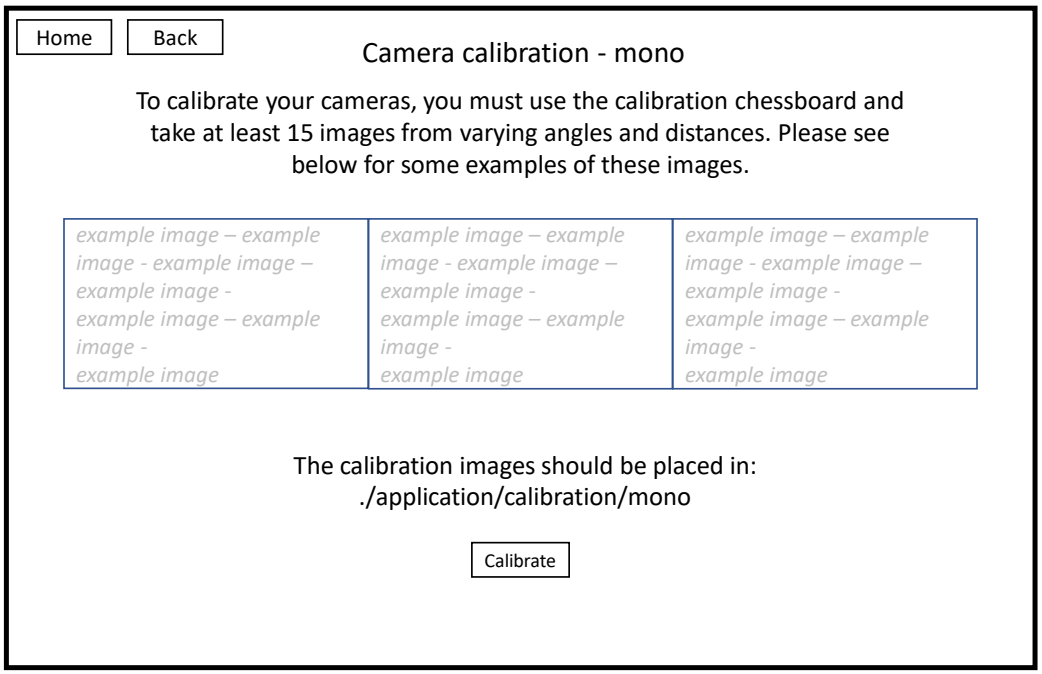


Figure 5.11: Screen 4.2 - the mono calibration screen

Error messages

Error messages should be informative without overwhelming or scaring the user. They should also give the user a clear path to the next step. Figure 5.12 shows an example of an error message that should show if no objects have been detected in the images used.

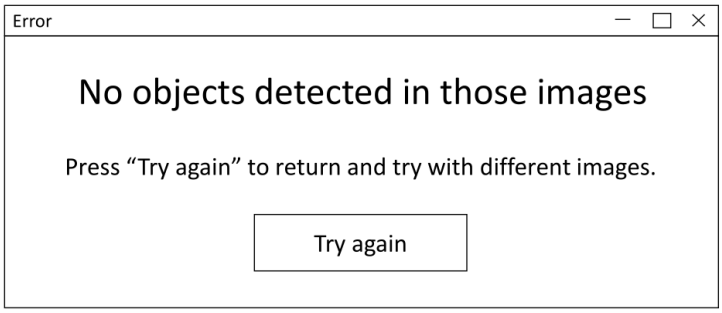


Figure 5.12: An example of an error message window

Chapter 6

Implementation

This section covers the implementation of the design - each of the main features of the software are broken up into increments following our software development methodology. For implementing the ML model, the data must be converted into the correct format and be arranged correctly to allow the ML model to read them. Appropriate variables for training the model must be found - these must compromise between accuracy and training time. Once training has been completed, analysing the results will ensure the model is working well. Test-Time Augmentation can also be trialled to see if results can be improved.

The cameras must be set up and calibration images taken.

A method to match up objects in a pair of images must be created, after which disparity and distance can be calculated.

Once these 3 increments are completed, it must be tied together using a simple GUI.

These increments can be tested briefly after implementation but tested more thoroughly in section 7.

6.1 Hardware

The following hardware has been used:

- CPU: Ryzen 5 5600x @ 3.7GHz (boost 4.6GHz)
- Memory: 16GB 3200MHz
- GPU: Nvidia GeForce GTX 1060 6GB
- Storage: 1TB SSD @ 2400MB/s read & 1950MB/s write
- 2x Yimona USB webcams

When it comes to training a ML algorithm, the GPU memory available has a large impact. Choices have been made keeping these hardware specifications in mind.

6.2 Increment 1 - Object detector

This increment is focused on requirements F4, NF1, NF2 and NF8.

6.2.1 Gathering data

Before training the model, the labels for the data must be in the YOLO format. The easiest way of doing this is to convert from the Scalabel format to the COCO format using the BDD toolkit (Yu et al., 2020) which is more widely used, and then to the YOLO format using a script provided by Mikhailyuk (2020) on github.

The data also needs to be in the correct directories. When training, the model looks at the file path given for a training image and replaces the last instance of '/images' with '/labels' in the directory. For example:

- Looking at image:

  `\FYP\Artifact\Data\bdd100k\images\100k\train\img1.jpg`

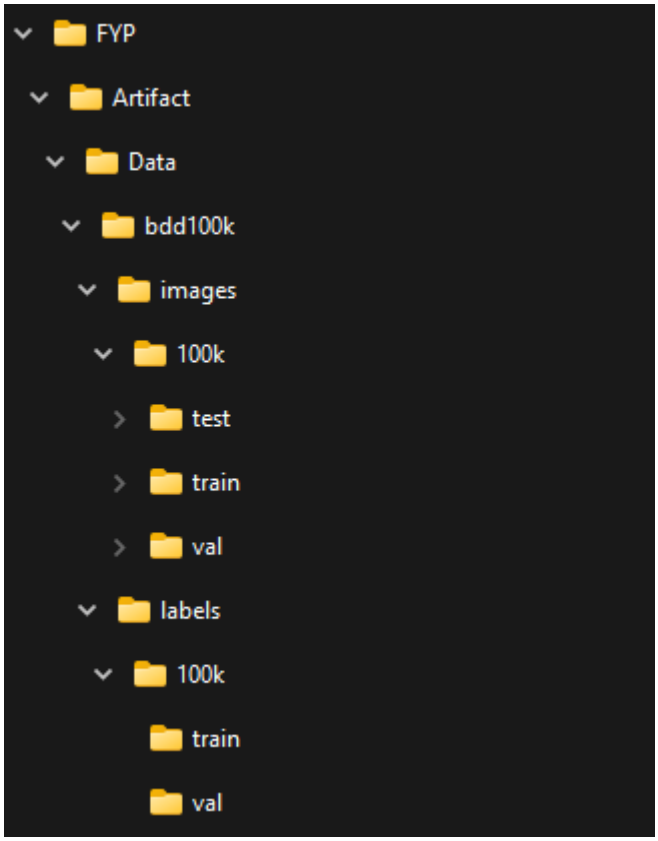


Figure 6.1: Training data directories

- Looking at labels for the image:

  \FYP\Artifact\Data\bdd100k\labels\100k\train\img1.txt

As such, the data is organised into the directories as shown in figure 6.1.

To point the model in the direction of this data, a YAML file is used that contains the file path for the training data and validation data, as well as the number of classes in the images and their names (car, truck, etc.).

6.2.2 Model test run

Before training the model to be used in the final solution, a test model will be trained. The aim of this is to catch any issues with the libraries or the

data. The following process was followed:

- An Anaconda environment was created
- The git repository for YOLOv5 by Ultralytics is cloned (Ultralytics, 2020)

```
git clone https://github.com/ultralytics/yolov5
```

- The YOLOv5 requirements are installed

```
Pip install -r requirements.txt
```

- The model can now be trained by providing image size, batch number, epoch number, data location and weight location.
  - As a test, the model is trained with the following values:

```
python train.py --img 416 --batch 1
--epochs 1 --data data.yaml
--weights yolov5s.pt
```

Due to an issue with the versions of some of the libraries installed (pytorch and cuda toolkit mainly), this model was trained using the machine's CPU, producing much lower training speeds than that of a GPU.

6.2.3 Training the model

A similar process was carried out again, however this time using cuda toolkit 10.2 and PyTorch-LTS to allow for GPU training.

Initially, the specifications shown in table 6.1 were used.

Image size	Batch number	Epochs	Device	Time taken
416	10	50	GPU	20 hours

Table 6.1: Initial training specifications

Image size	Batch number	Epochs	Device	Time taken
416	32	25	GPU	6 hours 47 mins

Table 6.2: Final training specifications

Due to the hardware limitations, this was predicted to take 20+ hours. Table 6.2 shows the results of increasing the batch size and reducing the epoch number.

This run completed without issue, finishing in 6 hours and 47 minutes. The model has 213 layers and 7037095 parameters, with a performance of 15.9 GFLOPs.

6.2.4 Analysing training results

During training, the model records key information that we can use to determine if it has been successful.

Table 6.3 shows that the class of objects with the highest mAP are cars (0.648). Trucks, buses, traffic lights and traffic signs are in the 0.45 - 0.5 range and most other classifications all significantly below this. We can see that, generally, the classes with the most labels also have the largest mAP - more data to learn from leads to more accurate results. A similar trend is shown with Recall.

As seen in figure 6.3, the mAP throughout the training shows a logarithmic pattern - starting to level out by the time it reaches 25 epochs. Our recall in figure 6.2 shows a similar, though less consistent pattern. Precision is not as clear, with many peaks and troughs throughout the training.

Calculating an F1 score is a popular method of measuring a model's accuracy. The F1 score is given using the calculation in figure 6.4.

As shown in figure 6.5, the F1 curve for cars is the highest - this makes sense as we know that cars had the most labels in our training data. The F1 score

Class	Labels	Precision	Recall	mAP0.5	mAP0.5:0.95
all	186008	0.623	0.353	0.374	0.191
pedestrian	13422	0.516	0.388	0.392	0.158
rider	658	0.711	0.19	0.258	0.0935
car	102910	0.593	0.636	0.648	0.389
truck	4241	0.636	0.419	0.471	0.321
bus	1659	0.616	0.394	0.451	0.330
train	15	1.00	0.00	0.00	0.00
motorcycle	460	0.557	0.215	0.258	0.106
bicycle	1039	0.475	0.282	0.290	0.120
traffic light	26881	0.541	0.524	0.484	0.158
traffic sign	34723	0.583	0.484	0.487	0.231

Table 6.3: Training results

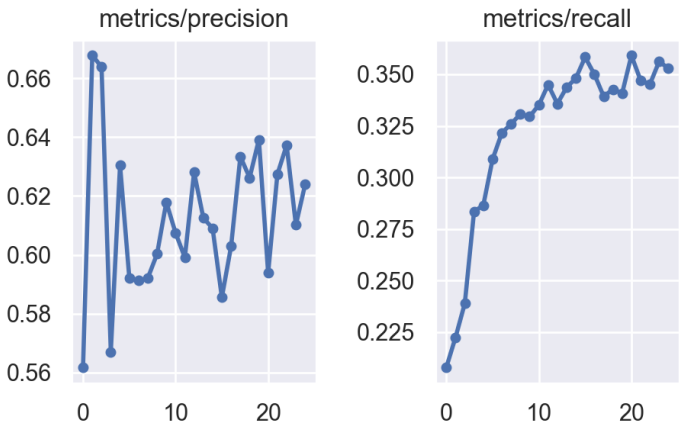


Figure 6.2: Precision and recall throughout the 25 epochs

for riders is the lowest, followed closely by motorcycles and bicycles - this is not surprising, as all of these have a low number of labels in our training

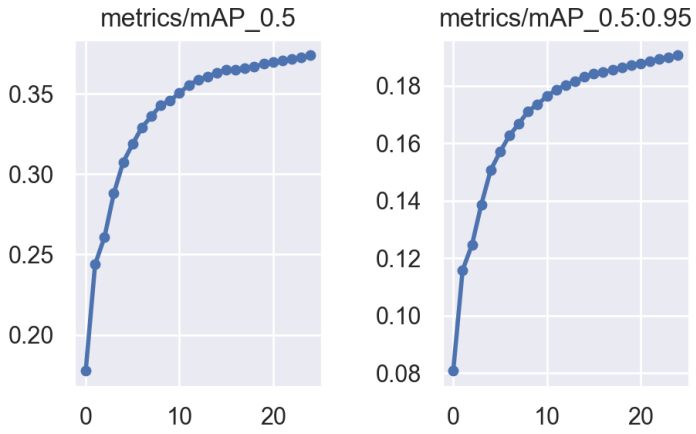


Figure 6.3: mAP throughout the 25 epochs

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Figure 6.4: F1 calculation

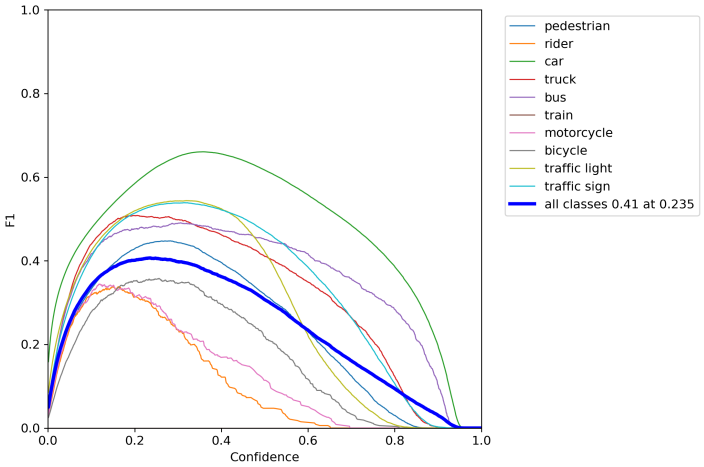


Figure 6.5: F1 curve of model

data (658, 460 and 1039 respectively).

The processing time for the model is 0.1ms pre-process, 2.7ms inference and 1.7ms non-maximum suppression per image.

Class	Labels	Precision	Recall	mAP0.5	mAP0.5:0.95
all	186008	0.585	0.349	0.358	0.181
pedestrian	13422	0.466	0.386	0.377	0.151
rider	658	0.697	0.164	0.244	0.0872
car	102910	0.514	0.638	0.630	0.374
truck	4241	0.618	0.414	0.460	0.311
bus	1659	0.571	0.399	0.432	0.316
train	15	1.00	0.00	0.00	0.00
motorcycle	460	0.55	0.207	0.243	0.1
bicycle	1039	0.488	0.265	0.276	0.112
traffic light	26881	0.447	0.533	0.452	0.145
traffic sign	34723	0.503	0.483	0.463	0.217

Table 6.4: Results using Test-Time Augmentation

6.2.5 Test-Time Augmentation (TTA)

TTA is a method of increasing the image size during detection with the aim of an improved precision and recall.

The results in table 6.4 show slightly lower results in all categories, and also increased the processing time to 0.1ms pre-process, 6.4ms inference and 2.3ms non-maximum suppression per image (an increase of 3.7ms inference and 0.6ms non-maximum suppression).

6.2.6 Retrieving object information

In order to obtain the object information from the detector, the detection script was edited to return the bounding box coordinates, the confidence and the object classification.

Each detected object will have an object created (of class 'detected') that stores bounding box, confidence, and object classification. The first function (get_objects.py) takes an input of the image file path, calls the detect function of the YOLOv5s detector and stores the results in a list. This list is then iterated through to pull out each object, remove any unwanted formatting and then create an object of class 'detected'. These objects are added to a list and returned. See Appendix B for the source code for this.

6.3 Increment 2 - camera calibration

This increment focuses on requirements F5, F6, NF6 and NF7.

The stereo cameras were set up at a baseline distance of 16cm - this was calculated using the lens and sensor properties of the cameras. This baseline should provide a minimum depth of roughly 0.95m.

6.3.1 Single camera calibration

Each camera must be calibrated individually to calculate the camera intrinsics.

In order to achieve this, we must first take images of an object of a known size - in this case, a chessboard with squares of 24mm.

The chessboard in figure 6.6 was provided by OpenCV (2018) for use with the

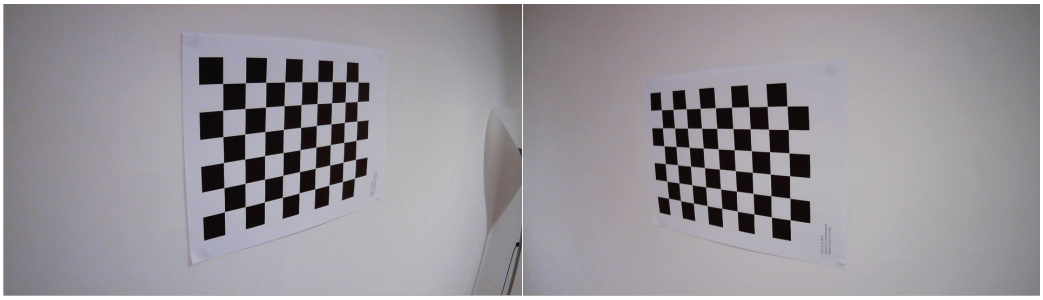


Figure 6.6: An example of two calibration images

calibration functions that their library provides. The documentation available at docs.opencv.org (OpenCV, n.d.) as well as learnopencv.com (Satya Mallick & Kaustubh Sadekar, 2020) were utilised for this.

The function has the following steps (get_cam_details.py):

- Define the world (3d) points; we know the number of squares on the chessboard as well as the size, so we can define this.
- Loop over the images, finding the chessboard corners and individual points on the board (see figure 6.7 for an example).

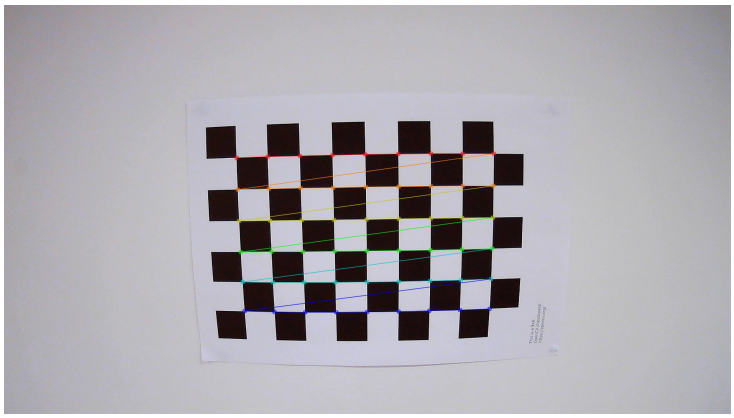


Figure 6.7: Detecting chessboard corners

- Store these as a list of image (2D) points.
- Use the OpenCV function calibrateCamera which requires both the world points and the image points as well as the image size.
- Write the camera matrix and distortion returned from this method to .csv files to be read from later on.

Then, in a separate function (undistort.py):

- Read the camera matrix and distortion from the .csv files.
- Calculate the optimal camera matrix, using the known matrix and distortion (this also returns a region of interest that can be used to

crop the image after altering it).

- Use the old matrix, the new matrix and the distortion to undistort the image (see figure 6.8).

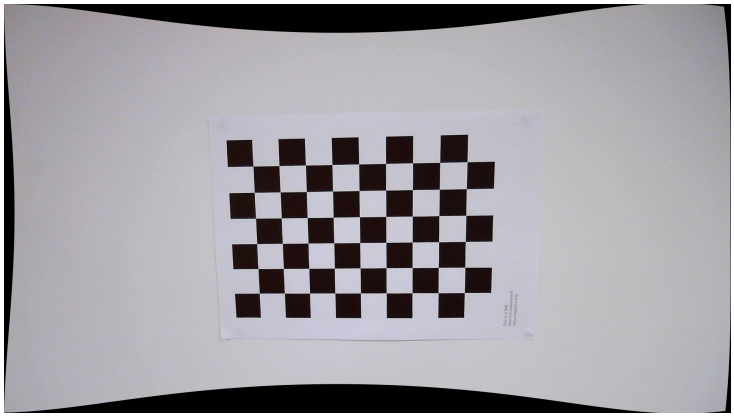


Figure 6.8: Undistorting chessboard image

- Crop the image using the ROI (see figure 6.9).

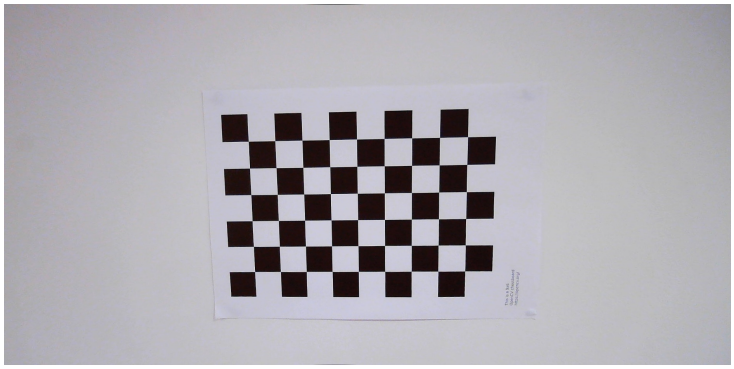


Figure 6.9: Undistorted image after being cropped

- Save the modified image, as well as the optimal matrix and ROI for later use.

6.3.2 Stereo camera calibration

The approach for this method is similar to the approach for calibrating intrinsics. The same chessboard is used, only it must be fully visible to both cameras.

This method (get_cam_details_stereo.py) has the following steps:

- Define the world (3D) points.
- Loop over the left images, searching for the chessboard.
- If found, add the coordinates (world and left) to separate lists.
- Loop over right images and, if found, add to list.
- Use the openCV function stereoCalibrate which uses world points, left and right points, camera matrices, distortion and image shape.
- Save the camera extrinsics for later use.

In a separate function (rectify_stereo.py), the camera extrinsics are used to remap the left and right images. Figure 6.10 shows an example of rectified images, with the red lines being added to demonstrate the matching y axis of features. Once the images are rectified, they are saved to the temp file for later use.

Figure 6.10: Example of stereo rectification

6.4 Increment 3 - Disparity and distance calculation

This increment focuses on requirements F7, NF10 and NF11.

Some approaches for creating disparity maps using some of the OpenCV functions were tested. StereoSGBM was used, which uses a modified version of Heiko Hirschmullers SGM method (Hirschmuller, 2008). This approach produced the varying results shown in figure 6.11, so something else was needed.

Figure 6.11: Original image and two of its disparity maps

The new method involved running the object detector on both images to produce lists of the objects in each image, and then using feature matching with both lists to pair matching objects together.

There are 2 functions that accomplish this:

6.4.1 SIFT.py

This function implements an Oriented FAST and rotated BRIEF (ORB) detector - this method was first introduced by (Rublee et al., 2011). Brute force matching is then used to find matching points between the two images. As part of the matching process, each match is given a Hamming distance value - this distance represents the similarity of the two points. The top 150 matches are chosen (a number reached through testing) and then the average disparity and distance is calculated. Any points with a disparity in the y axis of more than 10 pixels are discarded - the stereo rectification that is completed before this should keep matching points on a similar y axis, so any points with a large disparity here are likely false matches. The average distance and disparity are returned.

6.4.2 checkObjectMatches.py

This method has the following steps:

- Set the list with the fewest objects as list 1 and the other as list 2
  - This helps with discarding additional objects when the images do not have a matching number of objects detected
- Loop through list 1
  - Mask the image with the coordinates of the object of the current iteration
  - Loop through list 2
    - Mask the image with the coordinates of the object of the current iteration
    - Call the ORB function, passing it both masked images
    - If the match score is lower than the current lowest, set that object as the closest match

- If, once all objects are looped through, the lowest match score is less than 40 (a value decided through manual testing), then pair the 2 objects together as matching objects

6.4.3 Distance

Once the objects have been matched and the disparity found, the distance can be calculated. This is done by a separate function (get_distance.py), which reads the camera matrix, extracts the focal length and then uses the equation distance = (baseline * focal length)/disparity to calculate the distance.

6.5 Increment 4 - Graphical User Interface

This last increment is focused on piecing the functions of the program together into a simple, easy to use piece of software. This increment focuses on requirement F8.

To allow for creation of a user interface, the package PySimpleGUI was used - it wraps tkinter, Qt, Remi and WxPython and allows for fast, simple creation of a GUI (PySimpleGUI, n.d.).

The resolution of most windows was chosen to be 1280, 720 - this is small enough to fit comfortably on most standard monitors whilst being large enough to display images and text simultaneously at a readable size. statcounter.com (n.d.) found 1920x1080 to be the most popular monitor resolution as of February 2022, with 22.25% of all desktops using this resolution).

A slightly gray background was chosen as a white background was found to be too bright - the shade of gray used strikes a nice balance between contrast and brightness.

Confidence is not included on the bounding box for stereo images/videos as it was decided that it could be confused for distance at a glance, so distance

is the only number (other than object name) shown.

As distance is not produced for mono images, the confidence can be included on the bounding box without worrying about confusion between the two.

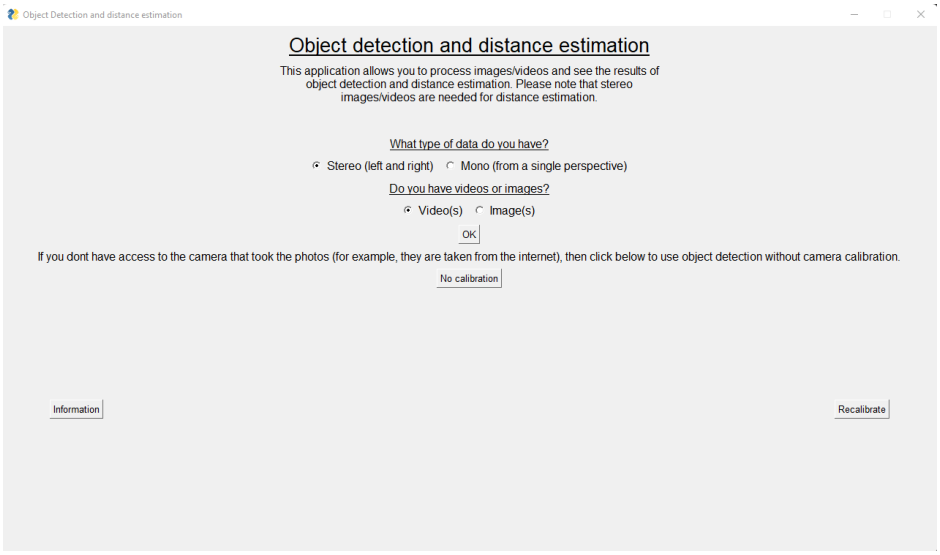


Figure 6.12: The home screen (1.0)

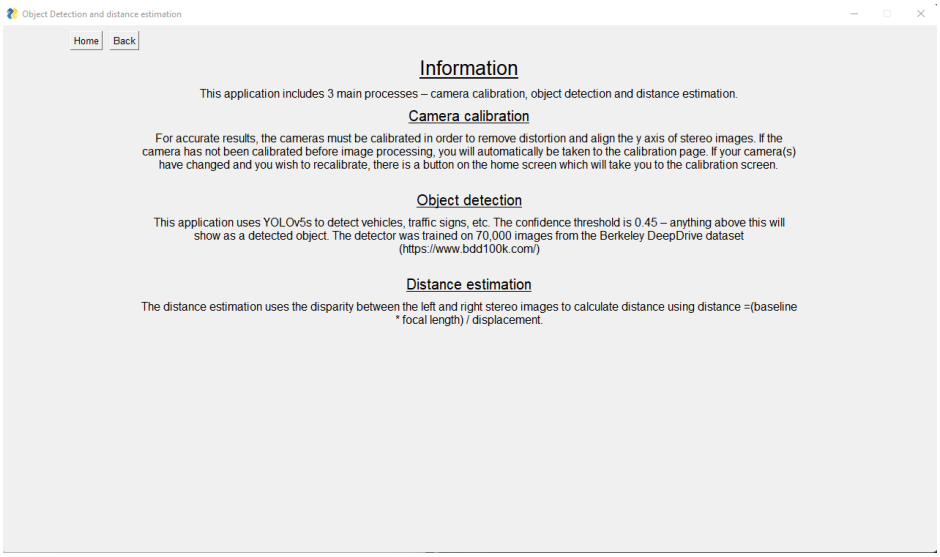


Figure 6.13: The information screen (1.1)

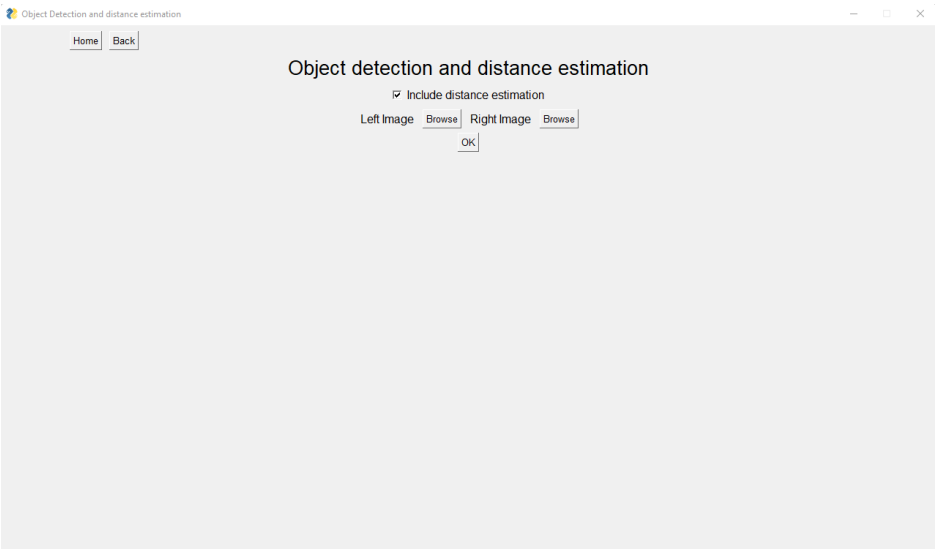


Figure 6.14: Stereo image processing screen (2.0)

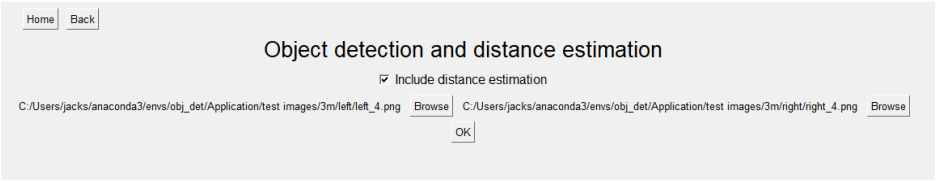


Figure 6.15: Stereo image processing screen with images uploaded

Figure 6.16: Stereo results screen (2.1/2.2)

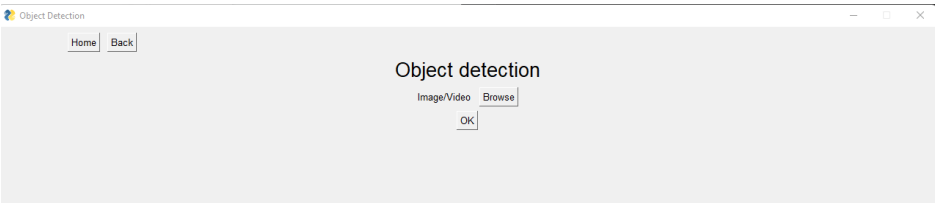


Figure 6.17: Mono processing screen (3.0)

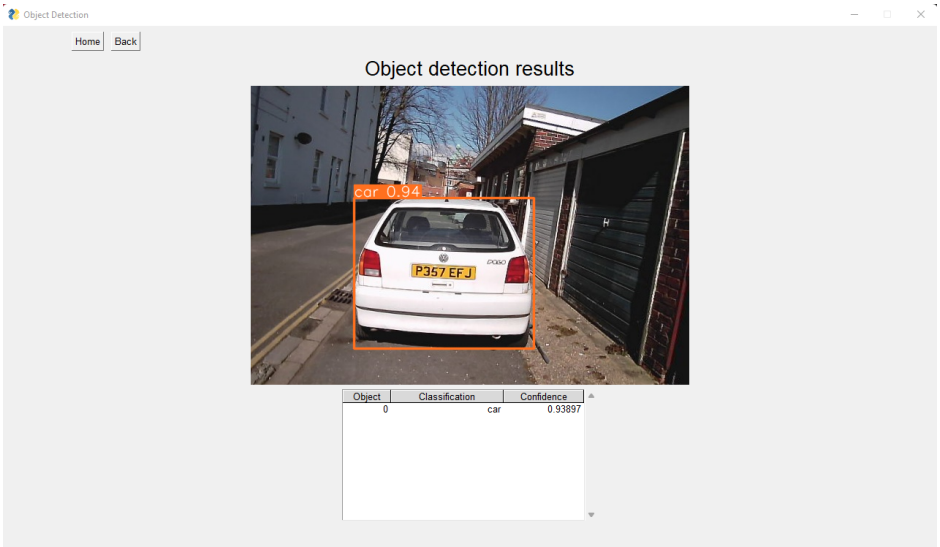


Figure 6.18: Mono results screen (3.1/3.2)

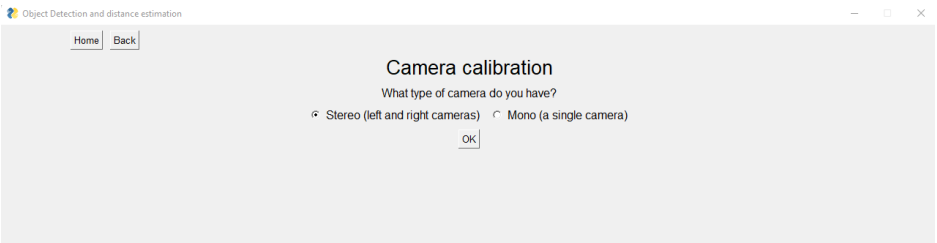


Figure 6.19: First calibration screen (4.0)

Figure 6.20: Calibration for stereo cameras (4.1)

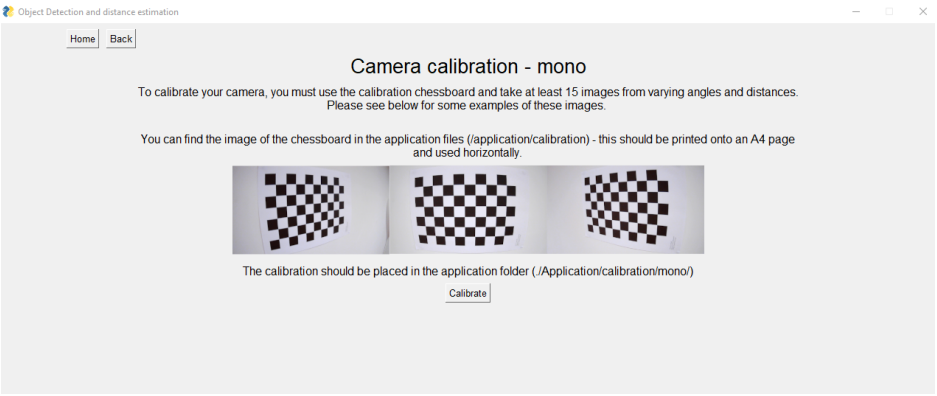


Figure 6.21: Calibration for single camera (4.2)

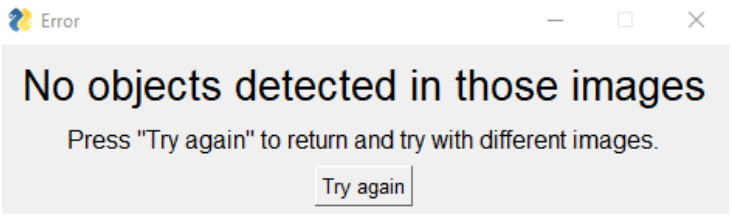


Figure 6.22: An error message that shows if no vehicles are detected in the images

Chapter 7

Testing

As part of the development methodology, unit testing has taken place in each increment to ensure that each function of the program carries out the intended task. In this section, functions will be tested in more detail to fine tune parameters, understand the characteristics of the detector, look at specific driving scenarios and ensure the accuracy of calibration and distance measurement. Additionally, all of these tests will also act as integration testing, ensuring each function of the program works as expected when tied together.

7.1 Object Detector

The data given after the implementation of the object detector gives us precision and recall results for each class, but we also have to complete some testing to set the confidence threshold.

7.1.1 Initial testing

To establish a baseline reading several images from the Berkeley DeepDrive (BDD) (Yu et al., 2020) dataset are used (these images were not used to train the model and have been reserved for testing).

The confidence threshold of the model is the minimum confidence at which it will detect an object - initially set at 0.1.

Image 1

The detector picks up a total of 15 cars and 4 traffic signs, at confidences varying from 0.9 to 0.1. Visually, only 9 cars and 2 traffic signs can be seen.

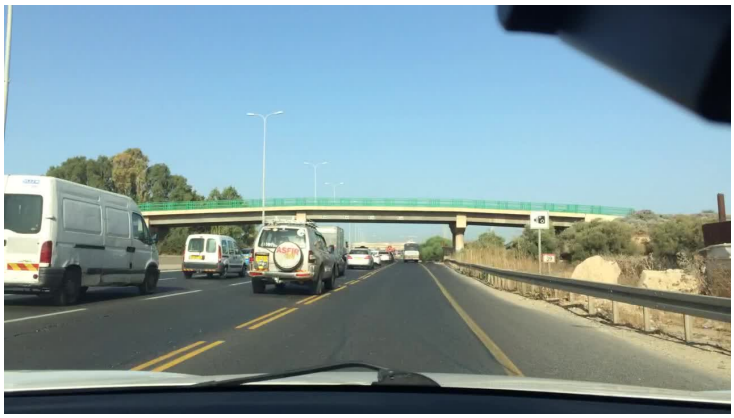


Figure 7.1: Image 1 before processing

Source: Yu et al. (2020)

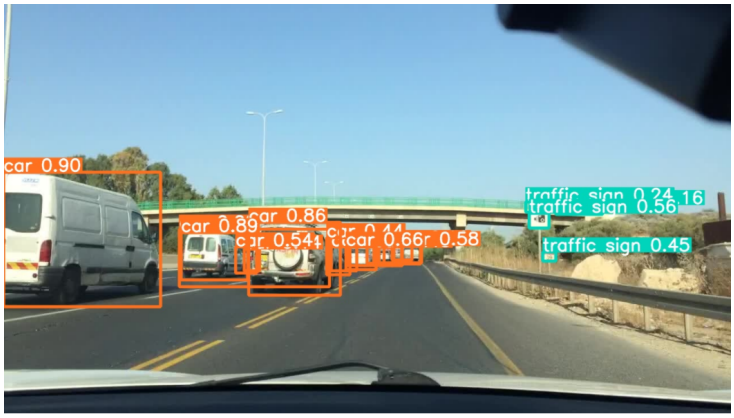


Figure 7.2: Image 1 after processing

Source: Yu et al. (2020)

Image 2

This test picks up 4 cars, 1 traffic light and 9 traffic signs, at confidences ranging from 0.8 to 0.1 Visually, only 2 cars are visible compared to the detected 4.

Figure 7.3: Image 2 before processing

Source: Yu et al. (2020)

Figure 7.4: Image 2 after processing

Source: Yu et al. (2020)

7.1.2 Altering the confidence threshold

As the results above show, a confidence threshold of 0.1 is too sensitive. To tune this parameter, a variety of test images will be used.

The confidence threshold must be at a value that does not miss any close objects, but also does not have many false positives.

Figure 7.5 will be used as our first test.

Figure 7.5: Test image of busy road

Source: Robert Hanashiro and Nathan Bomey (n.d.)

Leaving the confidence threshold at 0.1 gets us the results in figure 7.6.

From this result, we can derive a graph of the number of objects per confidence level (figure 7.7) - it shows that many of the detected objects are at a confidence of less than 0.2, with the peak being 0.1. Moving back to the Berkerley DeepDrive (BDD) dataset, we can take this same approach to see the different levels of confidence. Figure 7.8 and 7.9 show a test image of a street.

Figure 7.10 shows the objects per confidence level for this street test image

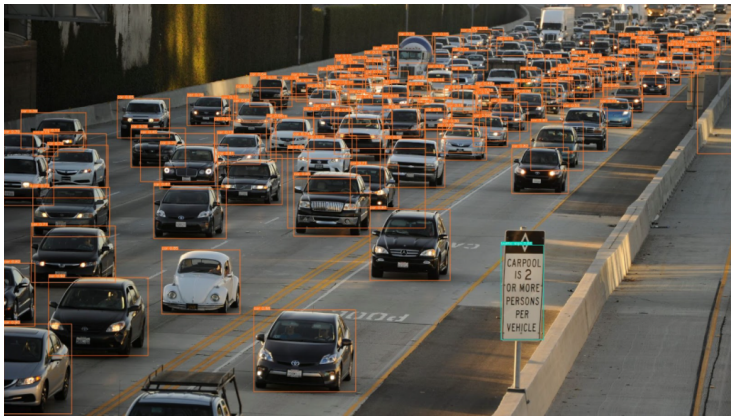


Figure 7.6: Test image at 0.1 confidence

Source: Robert Hanashiro and Nathan Bomey (n.d.)

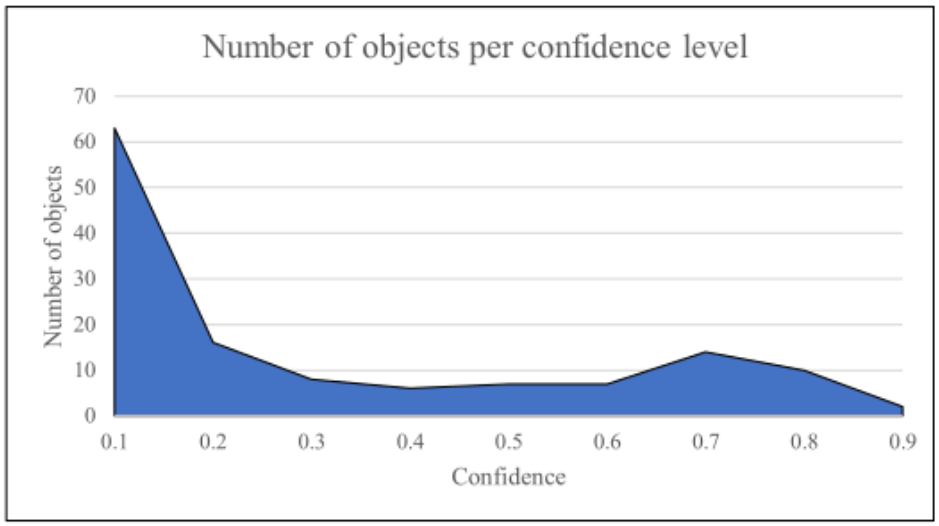


Figure 7.7: Number of objects per confidence level for the busy road test image (figure 7.5)

(figure 7.8) and demonstrates a similar trend to figure 7.7, with the majority of objects being detected at 0.2 or less.

Figure 7.8: Test image of street

Source: Yu et al. (2020)

Figure 7.9: Street test image (figure 7.8) results

Source: Yu et al. (2020)

Running this same test on a subset of 1000 BDD test images we get the results shown in figure 7.11. This test shows a peak at 0.1 confidence (just over 8000 objects) which decreases steadily to level out between 0.6 and 0.7 (1470 and 1474 objects respectively). During this test, the average post processing time was 0.24ms, the average inference time was 8.61ms and the average non-maximum suppression time was 1.51ms. From these tests, and erring on the side of caution (due to the use case), we can choose our confidence

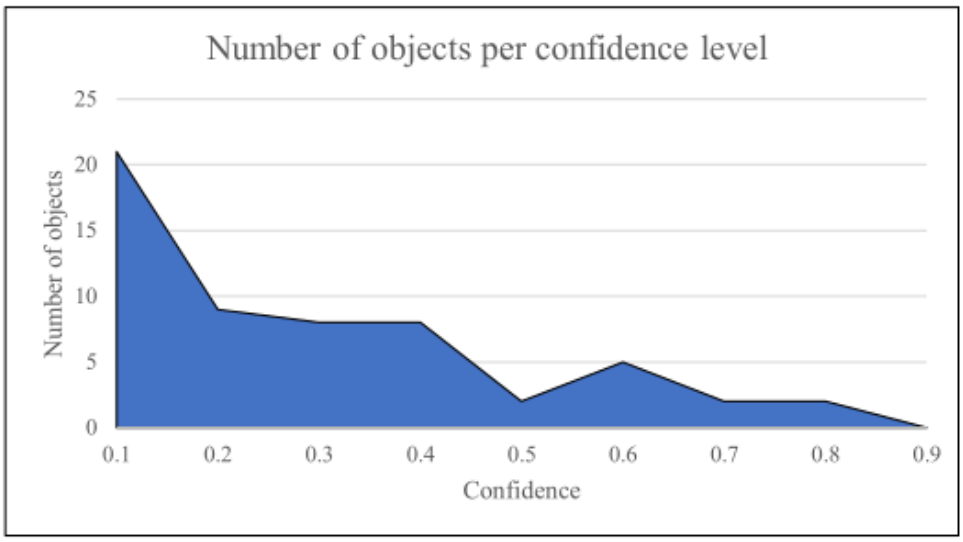


Figure 7.10: Number of objects per confidence level for street test image (figure 7.8)

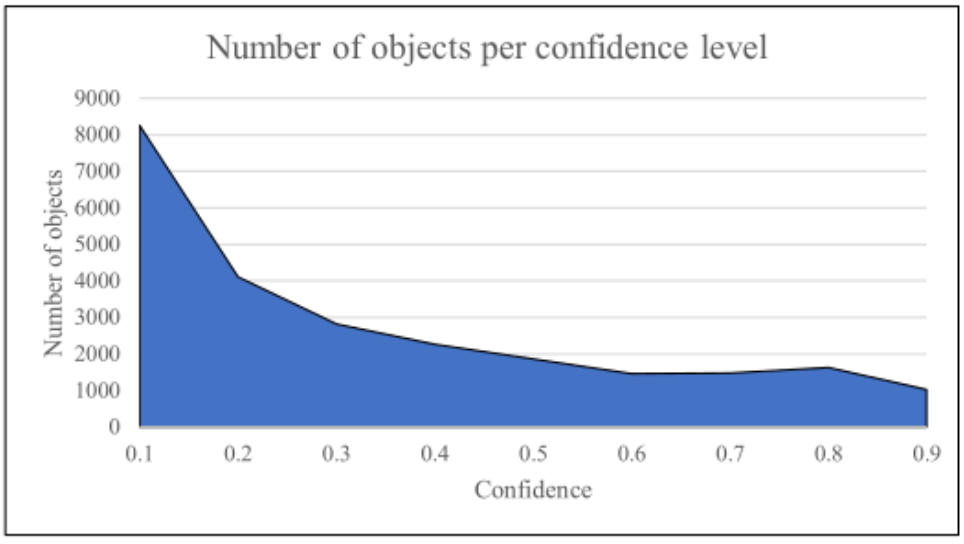


Figure 7.11: Objects per confidence level for 1000 images

threshold of 0.4 - this should minimise false positives while also ensuring any objects closer to the camera are detected.

7.1.3 Retesting

Using the same images as our initial testing, we can visually check if our new threshold value provides good results.

Image 1

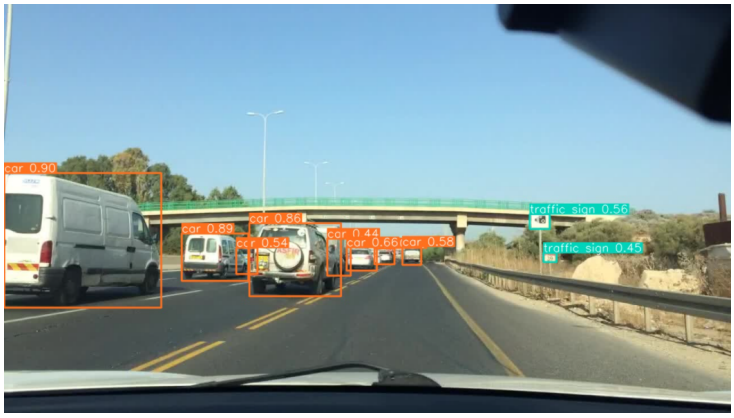


Figure 7.12: Test image 1 with a threshold of 0.4

Source: Yu et al. (2020)

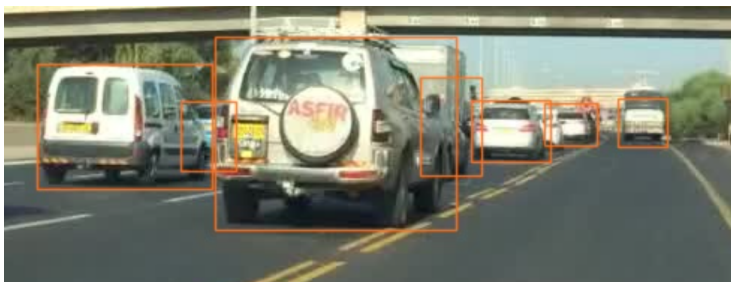


Figure 7.13: A closer look at test image 1

Source: Yu et al. (2020)

Figure 7.12 detects a total of 8 cars and 2 traffic signs, with a lowest confidence of 0.44 on a car that is only partially visible.

Image 2

Figure 7.14 shows no false positives and detects everything except the car which is only partially visible.

Figure 7.14: Test image 2 with a threshold of 0.4

Source: Yu et al. (2020)

7.1.4 Specific scenarios

The following test images are gathered from the DrivingStereo dataset (Yang et al., 2019).

Motorway/Highway driving

Figures 7.15 and 7.16 are a good demonstration of the distance that the system can recognise vehicles and road signs. Figure 7.16 also shows how the reflections on the right side of the image are not picked up by the system.

Roundabout/Turning

Figure 7.17 shows that all vehicles and signs are identified correctly. However, behind the vehicle closest to the camera you can see a pedestrian waiting to cross - this is not detected as they're mostly obstructed.

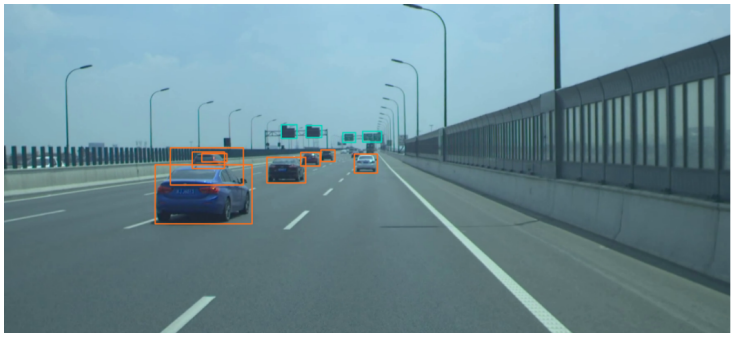


Figure 7.15: Motorway driving test image

Source: Yang et al. (2019)

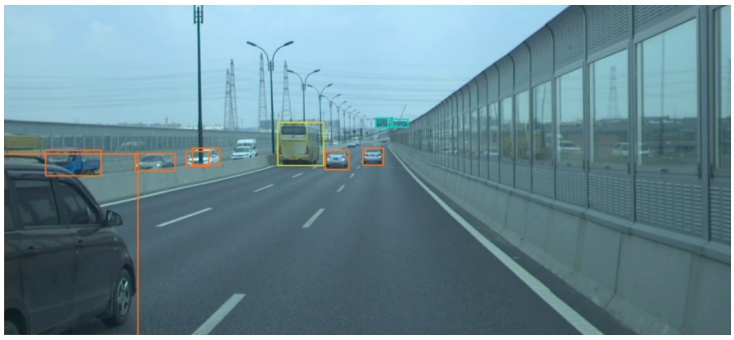


Figure 7.16: Motorway driving test image 2

Source: Yang et al. (2019)

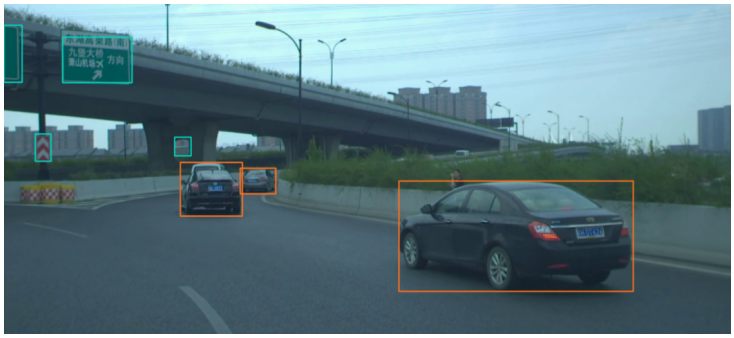


Figure 7.17: Roundabout test image

Source: Yang et al. (2019)

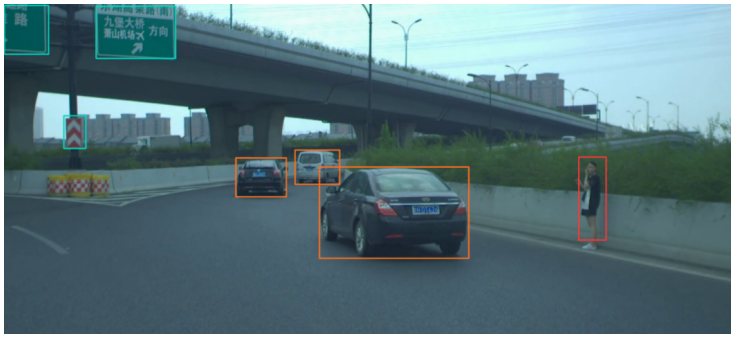


Figure 7.18: Roundabout test image 2

Source: Yang et al. (2019)

Figure 7.18 is taken a few frames after figure 7.17 and shows that the pedestrian is recognised when they are no longer obstructed.

Vehicle cutting in front

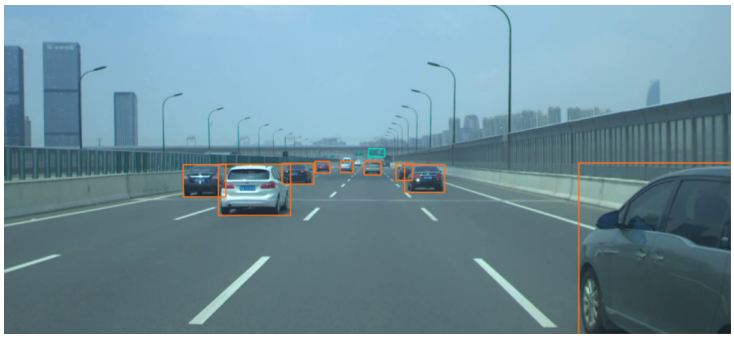


Figure 7.19: Test image of a vehicle cutting in front

Source: Yang et al. (2019)

Figure 7.19 shows that a vehicle cutting in front of the camera is still recognised - this would be important for autonomous vehicle features. In this same image, we can see that one of the signs in the far distance is recognised. A closer look at the information given to the system for these objects in the distance is shown in figure 7.20.

Figure 7.20: A closer look at the objects in the distance of figure 7.19

Source: Yang et al. (2019)

Vehicles at a distance

Figure 7.21: Test image for vehicles at distance

Source: Yang et al. (2019)

Figure 7.21 shows a long road with a vehicle at the far end. Going forward frame by frame we can see the furthest distance that the vehicle is recognised (figure 7.22).

Night

As shown in figure 7.23,7.24 and 7.25,the detector still works well at night. However, due to the decreased amount of information, there are more errors.

Figure 7.22: A few frames after figure 7.21

Source: Yang et al. (2019)

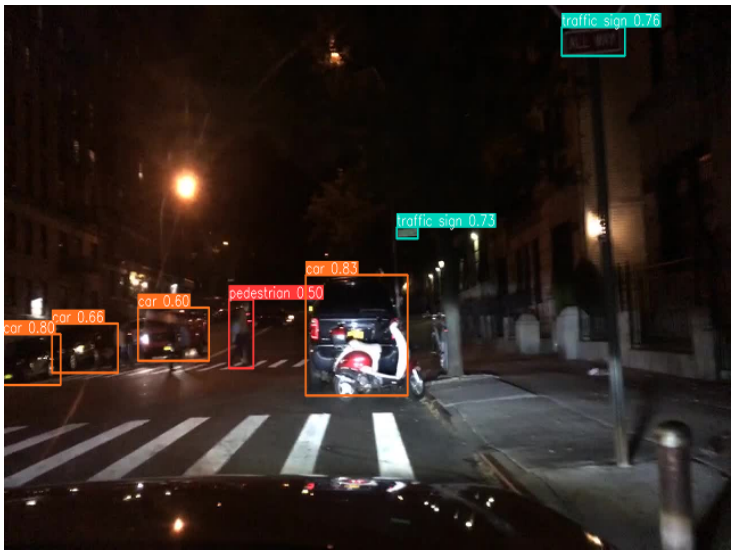


Figure 7.23: Night test image 1

Source: Yang et al. (2019)

For example, in figure 7.24 cars that are further down the street are missed and in figure 7.23 the motorbike in front of the car is missed.

Rain

As shown in figure 7.26, rain on the windscreen of the car causes confusion. Figure 7.27 shows that the car in front of the camera is missed.

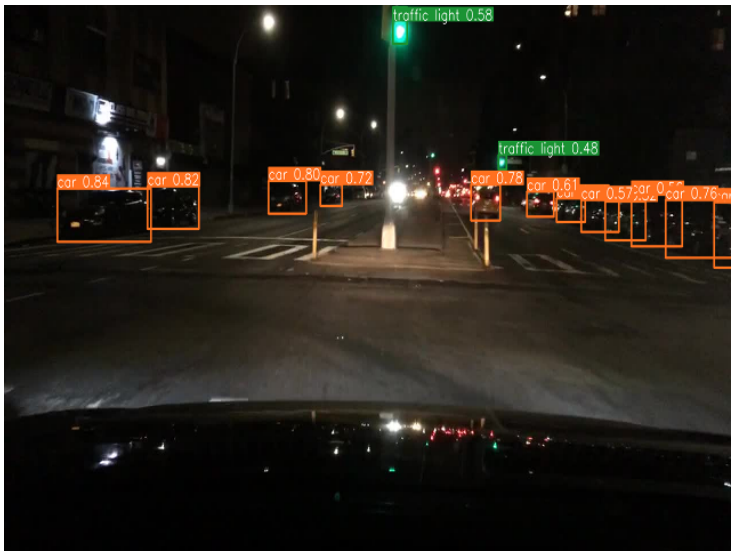


Figure 7.24: Night test image 2

Source: Yang et al. (2019)

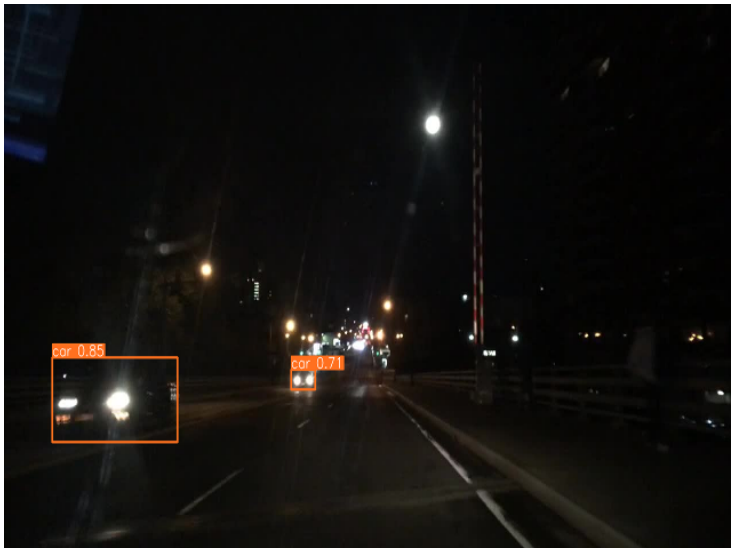


Figure 7.25: Night test image 3

Source: Yang et al. (2019)

Figure 7.26: Rain test image 1

Source: Yang et al. (2019)

Figure 7.27: Rain test image 2

Source: Yang et al. (2019)

7.2 Camera calibration

To test this, we look at the rectified images to see if there is any lens distortion and if the y axis lines up for a given feature.

7.2.1 Distortion

Figure 7.28 shows one of the calibration images for the left camera before and after the lens distortion is removed. Looking to the left of both images, it is visible that the rounded wall in the left image is straightened out in the right image.

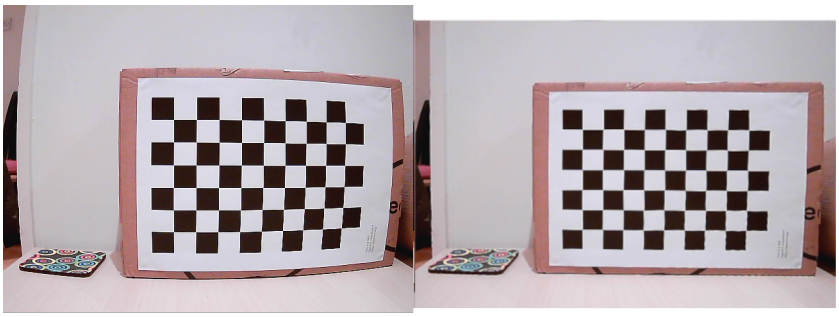


Figure 7.28: A left camera calibration image before (left) and after (right) removing distortion

Figure 7.29 shows a calibration image taken with the right camera before and after lens distortion is removed.

The focal length of the two cameras used varies slightly, despite being the same model - this is likely due to the manufacturing process. This makes it important to calibrate each camera individually and adjust the images accordingly.

7.2.2 Rectification

To check the rectification process, the left and right images are compared to see if the features are on a similar y axis. To do this, a feature is identified

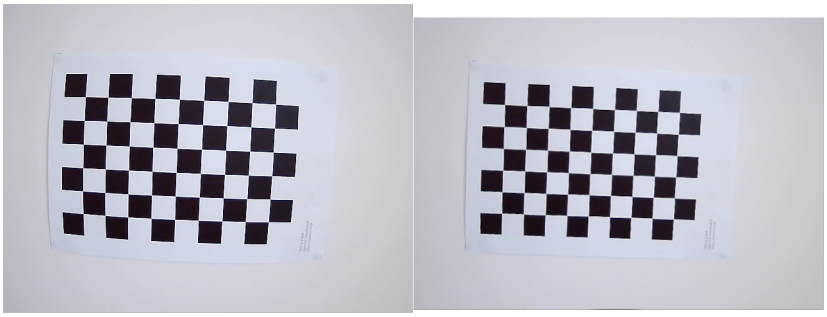


Figure 7.29: A right camera calibration image before (left) and after (right) removing distortion

in the left image and a horizontal line is drawn to the same feature in the right image.

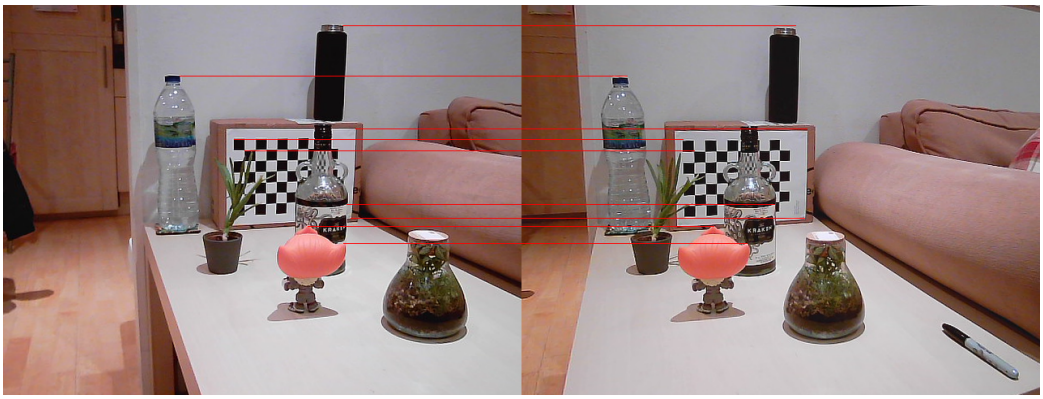


Figure 7.30: Left and right rectified images with lines joining matching features

Figure 7.30 shows the left and right rectified images of a scene. Most of the features are on the same y axis, however some are off by several pixels.

To take these several pixels into account, when calculating disparity a threshold of 10 pixels is used - if two points match but their y disparity is larger than this, they will not be used.

7.3 Disparity and distance estimation

The original approach for estimating the distance was to use the bounding boxes predicted by the YOLOv5s object detector. This worked well for some images, however images that were not taken directly behind the car produced unpredictable results.

As such, a new method was designed that uses feature matching to calculate disparity. After testing with different numbers of matches, using the top 150 was found to produce the best results. Table 7.2 shows the accuracy when using 150 top matches as opposed to table 7.1 which shows the top 15.

Absolute Distance	Disparity (pixels)	Distance (cm)	% error
200cm	36	225	12
400cm	22	370	-8
500cm	9	887	77

Table 7.1: Accuracy using the top 15 matches

Absolute Distance	Disparity (pixels)	Distance (cm)	% error
200cm	43	191.87	4.07
400cm	20.47	402.48	0.62
500cm	15.32	537.79	7.56

Table 7.2: Accuracy using the top 150 matches

The distance test data was taken every 100cm from 1m to 10m, and then every 5m up until 30m. Figure 7.31 shows some examples of these images from different distances (3m, 7m, 15m). An average of 3 images were taken from each distance, all from different angles.

Figure 7.32 shows the percentage error for each distance for both the best results and the average results.

Figure 7.31: An example of measurement images - 3m, 7m, 15m

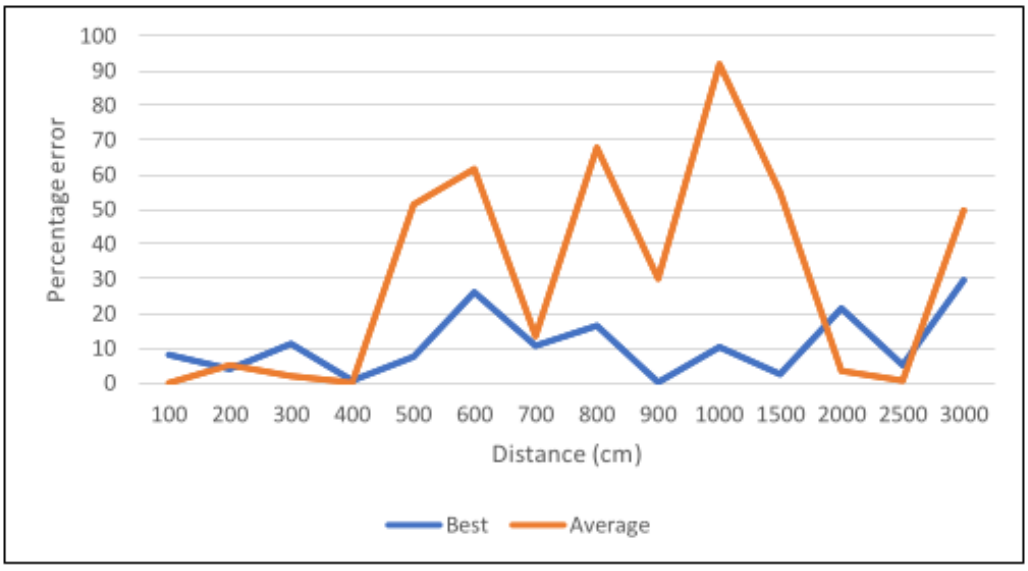


Figure 7.32: Percentage error against distance for the best and average results

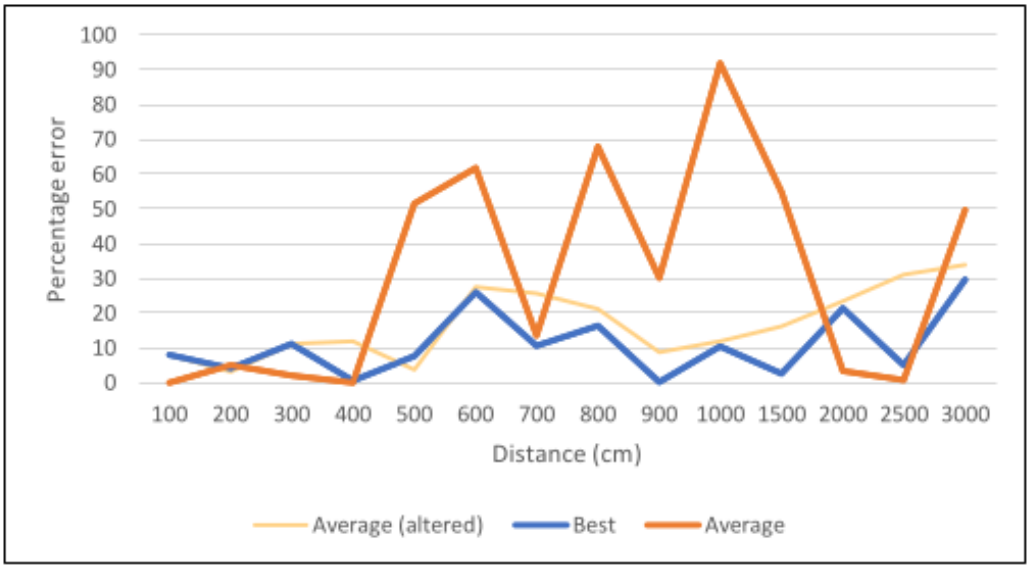


Figure 7.33: Percentage error against distance for new average values

The average error peaks much higher than the best results - this is due to inconsistencies with the results. For example, the results for 500cm were 537cm, 425cm and 1309cm - the erroneous results of 1309cm increases the average substantially. This is the case for many of the peaks of the average results. Removing the outlying value and calculating the new, altered average gives figure 7.33. It also reduces the standard deviation drastically - see figure 7.34.

Figure 7.33 shows that by removing these outlying values, more consistent results are obtained. As this system aims to process multiple frames per second, a feature could be added during further development that discards the outlying value from the last few frames.

Further testing was completed at shorter distances of 30cm, 45cm and 60cm - this is because the results so far suggest that the system is more accurate at shorter distances (100cm - 400cm). The results of these tests are shown in table 7.3. The results for each distance are more consistent than with the larger distances and the percentage error is consistent.

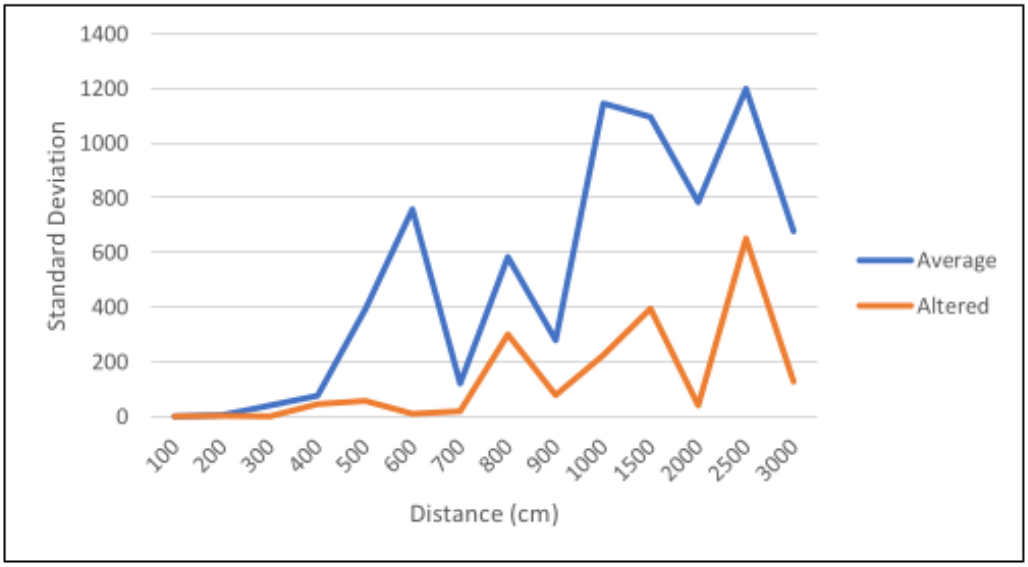


Figure 7.34: Standard deviation against distance for the average and the average with outlying values removed

Absolute distance (cm)	Distance results (cm)	Average	% error	sd
30	21.91, 23.14, 21.53, 23.28, 23.51	22.67	24.45	0.80
45	33.34, 31.74, 32.09, 34.36, 35.22	33.35	25.89	1.32
60	45.44, 42.68, 41.69, 47.8, 45.65	44.65	25.60	2.20

Table 7.3: Results from shorter distance testing

7.4 User interface

The user interface is tested by:

- Ensuring there are data type errors whenever the user uploads an unsupported file type.
- Checking all text fits within the window.
- Verifying all navigational buttons work correctly.
- Checking all action buttons work correctly.

Throughout the user interface testing, the only issue that was found is with

video processing. Once the user has selected the 'Go' button and the video starts to process they are unable to use the navigation buttons. This is due to the program looping through the frames of the video.

As a temporary workaround, closing out of this window while the video is processing will return the user to the home screen.

Chapter 8

Evaluation and further testing

This chapter shows which requirements have been met and provides further clarification on some of these. Requirements that are not met are evaluated and discussed.

The project as a whole is evaluated and possible future work is discussed.

8.1 Requirements analysis

Requirement	Completed?	Requirement	Completed?
F1	Yes	NF1	No
F2	Yes	NF2	No
F3	Yes	NF3	Yes
F4	Yes	NF4	Yes
F5	Yes	NF5	Yes
F6	Yes	NF6	Yes
F7	Yes	NF7	Yes
F8	Yes	NF8	Yes
F9	Yes	NF9	Yes
		NF10	Yes
		NF11	No

Table 8.1: Requirement evaluation

8.1.1 Requirements discussion

F6 - Calibrate the stereo cameras to keep matching points on a similar y axis

While it does work for the purpose, the stereo image rectification is not perfect and can sometimes be off by up to 10 pixels - this is likely due to either the calibration images used or the camera alignment. Any features with more than a 10 pixel disparity in the y axis are not used for feature matching, so the impact of this is minimised.

F9 - Process images/videos in real-time

Videos were fed into the system to be processed in "real-time" - without any pre-processing of the video. Processing videos produced a low frame rate - this is something that I feel could be optimised and improved with further development. Processing time is discussed further below (requirement NF3).

NF1 - Detect objects at a distance of up to at least 200m

The YOLOv5s detector recognises key objects and obtains their bounding box coordinates. In clear conditions, we get excellent results. When conditions are not clear, we get satisfactory results, with few objects being missed. The test data taken with the cameras ranges from distances of 1m to 35m. In the 35m images, vehicles can be seen in the background (roughly 70m), however the resolution of the cameras is not high enough for the detector to locate these further vehicles. When the detector is provided with more information (e.g., higher resolution), objects can be found at a further distance (figure 7.19 is a good example of this).

NF2 - Detect objects with a high rate of precision (at least 40%)

The mAP0.5 of the entire system is 0.374 - this falls short of the 40% target.

As discussed in section 6.2.4, the mAP0.5 for cars is 0.648, which does meet our 40% goal - this is likely due to the number of labels given in the training data. Of our 10 categories, 5 meet our 40% map0.5 requirement, with pedestrians coming close at 0.392.

Given a new, larger set of training data, I predict that this requirement could be met.

NF3 - Process images/frames at a speed faster than the average human reaction time

On our hardware, the system takes on average 600-700ms to calculate distance for a stereo pair, taking slightly longer (up to 1 second) on complex images with multiple objects. See figure 8.1 for a more complex frame that took 0.98s with distance estimation - 0.07s of this is the distance calculation, 0.09s is image rectification and 0.58s is the object detector. The rest of the time is taken up by feature matching, bounding box drawing, etc. Without the bounding box drawing (just generating all of the data), the time taken is 0.91s.

Figure 8.1: A more complex scenario

Monocular images process at a much faster rate - around 0.29s for this same image. This is as only one image needs to be processed, stereo rectification is not carried out and neither is feature matching or bounding box drawing (this is drawn by the object detector).

However, when running the GPU of the system at +100Mhz core and memory clock, we see an improvement from 0.91 seconds to 0.85 seconds, showing how hardware dependent this program is.

With the average stereo image processing time being 600-700ms, our system meets the overall requirement of a processing time of less than 1.4 seconds. However, in order to reduce this to closer to 200ms, the speed of the object detector must be increased - this could be done by using hardware specifically

designed for these types of calculations (for example, the NVIDIA (n.d.) A100 GPU).

NF4 - Produce a data stream of equal or less than a similar LiDAR system

During testing, the largest data stream per second was found to be around 35Mbit/s - this falls within our goal of 20-100Mbit/s. This was when processing stereo video - the data stream is less when processing stereo images and less again when processing mono images.

This data is in the form of 7 PNG files of resolution 640x480 which are stored in the applications temp file and updated each frame.

As the data from each frame is overwritten, the amount of memory the application needs (without calibration and test images) is roughly 5GB, and does not increase over time.

NF5 - Have a total sensor cost of less than LiDAR or RADAR sensors

The two cameras used cost a total of £51.98 - this is comparable with RADAR (£38-£152) and much cheaper than LiDAR (£380-£761).

NF11 - calculate distance to within an accuracy of 1cm

The distance calculation is not as accurate as I had hoped, with the altered average of the results being accurate to within about 10% up to 500cm, and ranging between 10-25% up to 3000cm. While some of the results give good precision (accurate to 1 or 2 cm), it is the inconsistency in these results that lets the system down and means that we have not met requirement NF11.

Possible reasons for this are:

- Camera resolution: as distance is calculated using pixel disparity, the

number of pixels available in an image is proportional to the precision of the calculated distance - the higher the resolution of the camera, the more accurate the disparity can be. Our system uses images of 640x480 pixels - this was to keep processing time and storage size down. At below 5m, this resolution works well. When testing at 10m and 20m the disparity is calculated at 2.3 and 1.9 respectively. This small change in disparity at double the distance suggests that the resolution is not high enough to accurately calculate longer distances.

- Camera alignment: our cameras may not have been aligned accurately enough - the more accurately these are aligned the less the system has to do to compensate for differences. This could be fixed by using a specially designed stereo camera where both lenses are fixed.
- Camera baseline: the baseline of our cameras (the distance between them) is proportional to the distance at which we can detect objects. Increasing the baseline could lead to more accurate results at further distances, but would increase the minimum distance at which an object is in both frames.

8.2 Evaluation of project management

The PID document (Appendix A) outlines the initial problem, aims, deliverables, constraints, approach, methodology and time management. Over the course of the project, these have changed and evolved.

The overall aim, to test if cameras can be used as a replacement for LiDAR or RADAR, has not changed.

When comparing our system to LiDAR, we mostly compared to solid state LiDAR (as opposed to 360 degree LiDAR as specified in the PID) - this is as many companies are developing solid state LiDAR specifically for automotive purposes, making it a more suitable comparison.

Most of the secondary objectives specified in the PID were met:

- Detect the vehicle in front
  - We expanded this to detecting all vehicles in the scene, as well as other objects such as traffic signs, pedestrians, etc. This was done to increase the number of possible features of the program (reading signs/traffic lights, stopping for pedestrians, etc.).
- Estimate the distance to the vehicle ahead with a good degree of accuracy
  - We estimate the distance to all objects, however the accuracy varies.
- Warn when the distance goes under a threshold
  - The simplest of the objectives, which takes the detected distance and checks if it is under a certain threshold. If an object is closer than 5m, a message is printed to the console (when integrated with other software, this is where another function would be called, such as emergency braking).

The original plan was to use traditional computer vision approaches for vehicle detection. However, after conducting research for my literature review, using a ML algorithm looked to be more suitable for my use case. As such, the switch to using ML was made - this required a large amount of research on algorithms and implementation as it was not something I was familiar with.

The PyCharm IDE was used instead of Visual Basic, this was due to the fact it is "considered the best IDE for python developers" (Anu Upadhyay, 2021). As it is focused on Python development specifically, it was chosen for this project.

Additionally, instead of mounting the cameras to a vehicle they were mounted to a movable base and this was held by hand to capture test data. It was

deemed unsafe to set up the cameras and laptop in a vehicle, and holding this by hand achieved very similar results.

8.2.1 Methodology

This project loaned itself well to an incremental methodology, allowing us to split our main features (object detector, camera calibration, distance calculation, GUI) into different increments that could be worked on separately.

Having 4 distinct increments also allowed for easier time management.

8.2.2 Time management

Our original gantt chart planned for 3 increments and a finish date of 25th February.

As it was decided that further research into machine learning methods was needed during the literature review stage, this increased how long the review took, completing it by mid January.

An extra increment was also added - this is as the original plan was to not use a GUI and instead run everything through command line. It was decided that using a simple GUI would improve usability greatly, allowing for a better showcase of the program.

All of this pushed the timeline back to a completion date in late March. The final gantt chart can be found in appendix C.

8.3 Future work

In our testing and evaluation, a few topics were covered that would be included in any future work on this project. These include:

- Using higher resolution , more accurately aligned stereo cameras at a higher baseline

- Training the detector on more data
- Take an average distance from the last few frames of a video, removing any outlying data
- Using more suitable hardware

Other topics that have caught my attention while researching this project include:

- Lane detection
  - This is something that is common in many cars today. It would provide useful information to the system that could be used in combination with other data to drive important decisions.
- Generating a 3D point cloud
  - Provided the distance calculation becomes accurate enough, generating a 3D point cloud of depth information would make this system much more comparable to LiDAR and RADAR (as this is what they usually generate). This could also make integration with software that has been designed for LiDAR and RADAR much simpler.
- Object path estimation
  - Using the relative motion of objects to predict their path. For example, tracking the motion of a vehicle and taking an action before it crosses in front of the camera.

Chapter 9

Conclusion

As discussed, this project meets most of its objectives and requirements, with a few needing further work in order to be met. Chapter 8 covers which objectives were met, what changes should be made, and future work that could be carried out.

It has been shown that, to some extent, cameras can replace some functionality of LiDAR and RADAR systems in vehicle autonomy. The distance calculation is what lets the system down - if this can be developed to give a more accurate, consistent value, then it will be more comparable to the other sensors.

Applying this technology to the area of vehicle autonomy is high risk as it deals with human life, so it must be ensured that the system is robust and accurate. Stereo camera systems are currently used in automation, for example, random bin picking (RBP) where a specific object must be picked up from a pile of parts scattered on an assembly line. Using ML and cameras for automation is something I can only see becoming more common, as it is cheaper and simpler than the comparable active sensors.

This project has made me a more well rounded programmer and software engineer, becoming much more comfortable with the python programming

language and coming up with more creative solutions to problems. In addition, my project management skills have benefited greatly from planning and structuring this project.

If I were to start this project again, I would use multiple camera setups in order to test different lenses, resolutions and baselines, allowing me to find the optimal setup for this application. For future projects, I will be more inclined to incorporate ML when appropriate and more comfortable doing so.

References

Abdulateef, S. K., & Salman, M. D. (2021). A Comprehensive Review of Image Segmentation Techniques [Publisher: Republic of Iraq Ministry of Higher Education & Scientific Research (MOHESR)]. *Iraqi Journal for Electrical & Electronic Engineering*, *17*(2), 166–175. https://doi.org/10.37917/ijeee.17.2.18

Anu Upadhyay. (2021). Top 10 Python IDE and Code Editors in 2020 [Section: GBlog]. Retrieved April 2, 2022, from https://www.geeksforgeeks.org/top-10-python-ide-and-code-editors-in-2020/

Arenado, M. I., Oria, J. M. P., Torre-Ferrero, C., & Rentería, L. A. (2014). Monovision-based vehicle detection, distance and relative speed measurement in urban traffic. *IET Intelligent Transport Systems*, *8*(8), 655–664. https://doi.org/10.1049/iet-its.2013.0098

Blickfeld. (n.d.). Vision Plus - Blickfeld. Retrieved April 1, 2022, from https://www.blickfeld.com/lidar-sensor-products/vision-plus/

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection [arXiv: 2004.10934]. *arXiv:2004.10934 [cs, eess]*. Retrieved February 9, 2022, from http://arxiv.org/abs/2004.10934

Brownlee, J. (2018). Difference Between a Batch and an Epoch in a Neural Network. Retrieved February 27, 2022, from https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

Chris Mellor. (2020). Autonomous vehicle data storage: We grill self-driving car experts about sensors, clouds ... and robo taxis. Retrieved March

31, 2022, from https://blocksandfiles.com/2020/02/03/autonomous-vehicle-data-storage-is-a-game-of-guesses/

Christoph Domke, & Quentin Potts. (2020). LiDARs for self-driving vehicles: A technological arms race. Retrieved April 3, 2022, from https://www.automotiveworld.com/articles/lidars-for-self-driving-vehicles-a-technological-arms-race/

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection [ISSN: 1063-6919]. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, *1*, 886–893 vol. 1. https://doi.org/10.1109/CVPR.2005.177

Delua, J. (2021). Supervised vs. Unsupervised Learning: What's the Difference? Retrieved February 10, 2022, from https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

Despa, M. L. (2014). Comparative study on software development methodologies. *Database Systems Journal*, *5*(3), 21.

Dulari Bhatt. (2021). A comprehensive guide for Camera calibration in computer vision. Retrieved February 24, 2022, from https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-for-camera-calibration-in-computer-vision/

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object Detection with Discriminatively Trained Part-Based Models [Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(9), 1627–1645. https://doi.org/10.1109/TPAMI.2009.167

Ge, Z., Liu, S., Wang, F., Li, Z., & Sun, J. (2021). YOLOX: Exceeding YOLO Series in 2021 [arXiv: 2107.08430]. *arXiv:2107.08430 [cs]*. Retrieved February 14, 2022, from http://arxiv.org/abs/2107.08430

Hirschmuller, H. (2008). Stereo Processing by Semiglobal Matching and Mutual Information [Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence]. *IEEE Transactions on Pattern*

Analysis and Machine Intelligence, *30*(2), 328–341. https://doi.org/10.1109/TPAMI.2007.1166

Hoffer, E., Hubara, I., & Soudry, D. (2018). Train longer, generalize better: Closing the generalization gap in large batch training of neural networks [arXiv: 1705.08741]. *arXiv:1705.08741 [cs, stat]*. Retrieved February 27, 2022, from http://arxiv.org/abs/1705.08741

Intellias Automotive. (2018). The Ultimate Sensor Battle: Lidar vs Radar. Retrieved March 31, 2022, from https://medium.com/@intellias/the-ultimate-sensor-battle-lidar-vs-radar-2ee0fb9de5da

Jain, A., Bansal, R., Kumar, A., & Singh, K. (2015). A comparative study of visual and auditory reaction times on the basis of gender and physical activity levels of medical first year students. *International Journal of Applied and Basic Medical Research*, *5*(2), 124–127. https://doi.org/10.4103/2229-516X.157168

Karczewski, D. (2020). What Is The Best Language For Machine Learning In 2022? Retrieved February 26, 2022, from https://www.ideamotive.co/blog/what-is-the-best-language-for-machine-learning

Le, J. (2021). How to do Semantic Segmentation using Deep learning. Retrieved February 3, 2022, from https://nanonets.com/blog/how-to-do-semantic-segmentation-using-deep-learning/

Luminar. (n.d.). Retrieved February 7, 2022, from https://www.luminartech.com/products/

Mikhailyuk, A. (2020). COCO2YOLO. https://github.com/alexmihalyk23/COCO2YOLO

Mittal, K. (2020). A Gentle Introduction Into The Histogram Of Oriented Gradients. Retrieved February 8, 2022, from https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa

Mohammed, A. S., Amamou, A., Ayevide, F. K., Kelouwani, S., Agbossou, K., & Zioui, N. (2020). The Perception System of Intelligent Ground

Vehicles in All Weather Conditions: A Systematic Literature Review. *SENSORS*, *20*(22), 6532. https://doi.org/10.3390/s20226532

Naik, D., & Shah, P. (2014). A Review on Image Segmentation Clustering Algorithms.

NASA. (n.d.). Rover Cameras. Retrieved February 26, 2022, from https://mars.nasa.gov/mars2020/spacecraft/rover/cameras/

National Highway Traffic Safety Administration. (n.d.). The Road to Full Automation. Retrieved April 3, 2022, from https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

Nepal, U., & Eslamiat, H. (2022). Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs. *SENSORS*, *22*(2), 464. https://doi.org/10.3390/s22020464

NOAA. (2021). What is LIDAR. Retrieved February 7, 2022, from https://oceanservice.noaa.gov/facts/lidar.html

NVIDIA. (n.d.). NVIDIA A100 GPUs Power the Modern Data Center. Retrieved April 1, 2022, from https://www.nvidia.com/en-gb/data-center/a100/

O' Mahony, N., Murphy, T., Panduru, K., Riordan, D., & Walsh, J. (2016). Adaptive process control and sensor fusion for process analytical technology. *2016 27th Irish Signals and Systems Conference (ISSC)*, 1–6. https://doi.org/10.1109/ISSC.2016.7528449

Ohnsman, A. (2021). Luminar Surges On Plan To Supply Laser Sensors For Nvidia's Self-Driving Car Platform [Section: Transportation]. Retrieved February 7, 2022, from https://www.forbes.com/sites/alanohnsman/2021/11/09/luminar-to-supply-laser-sensors-for-nvidias-self-driving-car-platform/

OpenCV. (n.d.). OpenCV: Camera Calibration. Retrieved March 7, 2022, from https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html

OpenCV. (2018). Opencv/opencv. Retrieved April 25, 2022, from https://github.com/opencv/opencv/blob/767857c5164da39ce8d3aa719b5747cbda02fc59/doc/pattern.png

Park, S., Baek, F., Sohn, J., & Kim, H. (2021). Computer Vision–Based Estimation of Flood Depth in Flooded-Vehicle Images. *Journal of Computing in Civil Engineering*, *35*(2), 1–13. Retrieved November 1, 2021, from https://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=148750072&site=eds-live

ProductPlan. (n.d.). MoSCoW Prioritization. Retrieved March 3, 2022, from https://www.productplan.com/glossary/moscow-prioritization/

PySimpleGUI. (n.d.). PySimpleGUI. Retrieved March 29, 2022, from https://pysimplegui.readthedocs.io/en/latest/

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection [arXiv: 1506.02640 version: 5]. *arXiv:1506.02640 [cs]*. Retrieved February 9, 2022, from http://arxiv.org/abs/1506.02640

Robert Hanashiro, & Nathan Bomey. (n.d.). South bound traffic on the San Diego Freeway south of the Santa Monica Freeway in Los Angeles. Retrieved March 30, 2022, from https://www.usatoday.com/story/money/2015/07/29/new-car-sales-soaring-but-cars-getting-older-too/30821191/

Rohit Sharma. (2021). Python Anaconda Tutorial: Everything You Need to Know. Retrieved May 4, 2022, from https://www.upgrad.com/blog/python-anaconda-tutorial/

Royce, D. W. W. (1970). MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS, 11.

Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF [ISSN: 2380-7504]. *2011 International Conference on Computer Vision*, 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544

Satya Mallick, & Kaustubh Sadekar. (2020). Camera Calibration using OpenCV. Retrieved March 7, 2022, from https://learnopencv.com/camera-calibration-using-opencv/

Saxena, A., Schulte, J., & Ng, A. Y. (2007). Depth Estimation using Monocular and Stereo Cues, 7.

Schwaber, K. (1997). SCRUM Development Process. In J. Sutherland, C. Casanave, J. Miller, P. Patel, & G. Hollowell (Eds.), *Business Object Design and Implementation* (pp. 117–134). Springer London. https://doi.org/10.1007/978-1-4471-0947-1_11

Shchetko, N. (2014). Laser Eyes Pose Price Hurdle for Driverless Cars - WSJ. Retrieved April 3, 2022, from https://www.wsj.com/articles/laser-eyes-pose-price-hurdle-for-driverless-cars-1405969441

Stack Overflow. (2021). Stack Overflow Developer Survey 2021. Retrieved February 26, 2022, from https://insights.stackoverflow.com/survey/2021/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2021

Stanisław Jurecki, R., Lech Stańczyk, T., & Jacek Jaśkiewicz, M. (2017). Driver's reaction time in a simulated, complex road incident [Publisher: Vilnius Gediminas Technical University]. *Transport (16484142)*, *32*(1), 44–54. https://doi.org/10.3846/16484142.2014.913535

statcounter.com. (n.d.). Desktop Screen Resolution Stats Worldwide. Retrieved March 31, 2022, from https://gs.statcounter.com/screen-resolution-stats/desktop/worldwide

Takeuchi, H., & Nonaka, I. (1986). The New New Product Development Game [Section: Product development]. *Harvard Business Review*. Retrieved February 18, 2022, from https://hbr.org/1986/01/the-new-new-product-development-game

Tyagi, D. (2019). Introduction to ORB (Oriented FAST and Rotated BRIEF). Retrieved April 25, 2022, from https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

Ultralytics. (2020). YOLOv5. Retrieved March 14, 2022, from https://github.com/ultralytics/yolov5

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, *1*, I–511–I–518. https://doi.org/10.1109/CVPR.2001.990517

Wang, B., Qi, Z., Chen, S., Liu, Z., & Ma, G. (2017). Multi-vehicle detection with identity awareness using cascade Adaboost and Adaptive Kalman filter for driver assistant system (X. Hu, Ed.). *PLOS ONE*, *12*(3), e0173424. https://doi.org/10.1371/journal.pone.0173424

Wang, C.-Y., Yeh, I.-H., & Liao, H.-Y. M. (2021). You Only Learn One Representation: Unified Network for Multiple Tasks [arXiv: 2105.04206]. *arXiv:2105.04206 [cs]*. Retrieved February 9, 2022, from http://arxiv.org/abs/2105.04206

Wang, J., & Hu, X. (2021). Convolutional Neural Networks with Gated Recurrent Connections [Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–1. https://doi.org/10.1109/TPAMI.2021.3054614

Woods, A., Docherty, T., & Koch, R. (2002). Image Distortions in Stereoscopic Video Systems. *Proc SPIE*, *1915*. https://doi.org/10.1117/12.157041

Yang, G., Song, X., Huang, C., Deng, Z., Shi, J., & Zhou, B. (2019). DrivingStereo: A Large-Scale Dataset for Stereo Matching in Autonomous Driving Scenarios [ISSN: 2575-7075]. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 899–908. https://doi.org/10.1109/CVPR.2019.00099

Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., Madhavan, V., & Darrell, T. (2020). Bdd100k: A diverse driving dataset for heterogeneous multitask learning. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zaarane, A., Slimani, I., Al Okaishi, W., Atouf, I., & Hamdoun, A. (2020). Distance measurement system for autonomous vehicles using stereo camera. *Array*, *5*, 100016. https://doi.org/10.1016/j.array.2020.100016

Zaidi, S. S. A., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M., & Lee, B. (2021). A Survey of Modern Deep Learning based Object Detection Models [arXiv: 2104.11892]. *arXiv:2104.11892 [cs, eess]*. Retrieved February 14, 2022, from http://arxiv.org/abs/2104.11892

Zhe, T., Huang, L., Wu, Q., Zhang, J., Pei, C., & Li, L. (2020). Inter-Vehicle Distance Estimation Method Based on Monocular Vision Using 3D Detection. *IEEE Transactions on Vehicular Technology*, *69*(5), 4907–4919. Retrieved November 3, 2021, from https://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=143316893&site=eds-live

Appendix A

Project Initiation Document and Ethics Review Certificate

School of Computing
Project Initiation Document

Jack Sines

Using cameras for depth perception as an alternative to LiDAR and RADAR

Engineering Project

1. Basic details

Student name:	Jack Sines
Draft project title:	Using cameras for depth perception as an alternative to LiDAR and RADAR
Course:	Computing
Project supervisor:	Jiacheng Tan
Client organisation:	N/A
Client contact name:	N/A

2. Degree suitability

Some of the modules that this course covers include Software Engineering, Computer Vision and programming.
This project will make use of what I have learned so far in these modules and be used as a way of furthering my knowledge in these topics.

3. Outline of the project environment and problem to be solved

In 2019, there were 78,855 reported road traffic accidents in the UK. Of those, 66% were due to “driver/rider error or reaction”. A further breakdown shows that 37% of accidents were caused when the “driver/rider failed to look properly”, 20% when the “Driver/Rider failed to judge other person`s path or speed” and 12% were due to a “poor turn or manoeuvre”. In addition to this, an additional 23% of all reported accidents were due to “behaviour or inexperience” and 15% due to “impairment or distraction” (Department for Transport, 2019).

These statistics show what, while some accidents are unavoidable (for example vehicle defects, which the Department for Transport also reported accounted for 2% of all accidents in 2019), many are caused by driver error.

Autonomous vehicles aim to reduce the number of road accidents by limiting human input and instead having software control the vehicle (or specific functions of the vehicle).
The National Highway Traffic Safety Administration (NHTSA) has suggested a five-part model to classify different levels of automation – this is widely used in the area:

- Level 0: Zero autonomy; the driver performs all driving tasks.
- Level 1: Vehicle is controlled by the driver, but some driving assist features may be included in the vehicle design.

- Level 2: Vehicle has combined automated functions, like acceleration and steering, but the driver must remain engaged with the driving task and monitor the environment at all times.
- Level 3: Driver is a necessity, but is not required to monitor the environment. The driver must be ready to take control of the vehicle at all times with notice.
- Level 4: The vehicle is capable of performing all driving functions under certain conditions. The driver may have the option to control the vehicle.
- Level 5: The vehicle is capable of performing all driving functions under all conditions. The driver may have the option to control the vehicle.

(National Highway Traffic Safety Administration, 2021)

There are 3 sensors that are commonly used in autonomous vehicles: LiDAR (Light Detection and Ranging), RADAR (Radio Detection and Ranging) and camera – each of these has their strengths and weaknesses.

	Advantages	**Disadvantages**
Camera	- Similar to our own vision - Good at recognizing 2D information - Low cost	- Vulnerable to poor weather conditions
LiDAR	- High definition, meaning it is able to recognize objects more easily than RADAR - Laser not affected by weather	- High computing power needed - High cost
RADAR	- Radio waves not affected by weather	- Harder to identify objects due to a weak resolution

(Autocrypt, 2021)

One of the issues from a consumer standpoint is the cost that LiDAR and RADAR add to the price of the vehicle. In 2014, Shchetko noted that “Light Detection and Ranging (LIDAR) systems on top of many of today’s AVs cost $30,000 to $85,000 each” (Shchetko, 2014). While this price has decreased drastically, with “Luminar [offering] autonomous driving-grade LiDARs for under US$1000” (Domke & Potts, 2020), it is still more expensive than a camera would be. RADAR is not as expensive, but “varies from $50 to $200” (Intellias Automotive, 2018). Despite this, many manufacturers are still using LiDAR and RADAR in their vehicles.

The overall aim of this project is to find out if cameras can be used as a suitable replacement for RADAR or LiDAR to reduce vehicle cost.

4. Project aim and objectives

As part of meeting my aim and reaching a conclusion, I will be creating a piece of software that takes footage from multiple cameras fitted on the front of a vehicle and estimates the distance to the vehicle ahead using computer vision.

The more expensive rotating LiDAR sensors can provide a 360-degree view of what is around the vehicle, while the cheaper solid-state ones have a more limited field of view. Cosworth (in partnership with XenomatiX) claims the following functionality about their solid-state LiDAR on their website:

- “detects obstacles and pedestrians”
- “automatic emergency braking”
- “ability to differentiate road markings & hazardous objects”
- “data accurate to within a millimetre”

(Cosworth, 2021)

RADAR is used for similar purposes but works by using radio waves instead of laser.

To test our overall aim, we are going to focus on detecting the distance to a vehicle to a high degree of accuracy. By doing this, we are covering one of the main purposes of LiDAR and RADAR in autonomous vehicles. If we were to continue development after this, we could then apply this same method to detecting pedestrians and potentially hazardous objects to cover more of the functionality. This gives us the following aims for the software:

- Detect the vehicle in front
- Estimate the distance to the vehicle ahead with a good degree of accuracy
- Warn when the distance goes under a threshold

These objectives may be added to or altered during my literature review and research.

5. Project deliverables

- Research data (searches for papers, narrowing them down, etc.)
- Iterations of the Gantt chart for the project plan
- Proposed piece of software
- Main report and literature review

6. Project constraints

- Time: This project must be completed between October 2021 and May 2022. As this is an individual project, this time frame limits the scope of the project. To combat this, time will have to be carefully planned out between now and May.

- Cannot test in real time: If cameras were to be used to estimate distance on an AV, then this would be happening in real time and have a direct impact on the controls of the vehicle. It would be impractical to do this in real time, so instead pre-recorded footage will be used.

- Cannot test against LiDAR sensor: As it would be impractical to test my solution against a LiDAR sensor, instead we are looking at the main uses of LiDAR and then trying to meet those.

7. Project approach

I will create this project with Visual Basic using Python and the OpenCV library. I am familiar with Python and OpenCV, but I need to research the common methods used for depth perception – I will then decide if these methods will work for my use case or if I need to take a different approach.
While I already have some basic requirements, a literature review will allow me to collect any additional requirements that could benefit the project.

For methodology, I have decided to choose an incremental method. I chose this as it means the main requirements can be broken down into different increments, allowing me to focus first on segmenting the video to isolate the vehicle, and then to estimate depth.
This comes with the added benefit of being able to create the software quickly and gives me the chance to be flexible with requirements (meaning I can add any features that will be beneficial).
While an incremental method does require good planning, this shouldn't be an issue as we have a clear goal.

I also considered an iterative approach, as this gives me flexibility with adding new features (though not as easy as incremental) and is best suited for smaller teams. However, I was concerned that any late issues could cause delays.

8. Literature review plan

For my literature review, I will start by searching the library resources (EBSCO). To document this and allow me to narrow down my search, I will keep a record of the searches I make and the number of results returned. My aim will be to narrow this search down until I have an appropriate number of relevant papers to read.
In addition, I will repeat this process with Google Scholar to ensure I don't miss any helpful resources (though I expect this to provide less results than searching the library resources).
My goal for the literature review will be to look at any existing systems for this type of software and some of the common approaches.

9. Facilities and resources

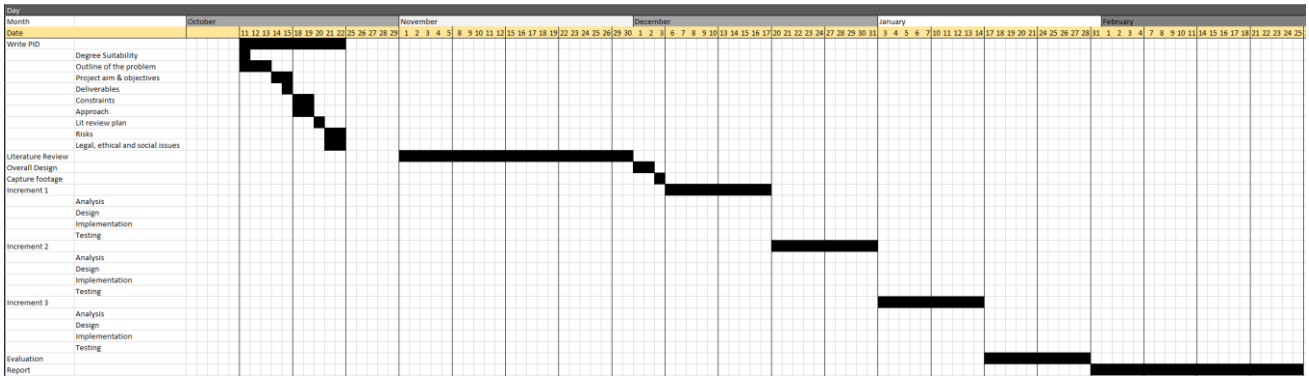
For capturing the footage, I will need 2 identical cameras which I will mount on the dashboard of my car at a fixed distance apart – knowing this distance will be crucial for estimating distance to other objects.
Once the footage is captured, I will be using Visual Basic to create the program – this allows me to work on it either in the university labs or at home.

10. Log of risks

Description	Impact	Likelihood	Mitigation	First indicator
Lockdown/measures put in place due to Covid-19	Will be unable to speak to supervisor in person or use the university labs	Likely	- Pose any questions to supervisor via email if necessary - Will keep all project files cloud-accessible so that I am able to continue from home	UK Government announcement
Running out of time to complete project	Will have missing parts of the project (likely the artifact/report as they will be the last things completed)	Unlikely	- Planning each phase of the project and, if for whatever reason I am unable to follow this plan, altering it to take this into account	- Missing project deadlines that I have given myself
Illness	May cause delays	Unlikely	- Allowing extra time in my planning	N/A
Computer crashing	Could lose work	Unlikely	- Save work regularly	N/A

11. Project plan

Initial version of the project plan below. This has been made to give me the flexibility to add requirements if my literature review/research reveals any.
This plan will likely change throughout the project but will serve as good evidence of the evolution of the project.



12. Legal, ethical, professional, social issues (mandatory)

During this project I do not plan on gathering any user data – if this changes, however, then I will ensure that I comply with the General Data Protection Regulation (GDPR) and the Data Protection Act when collecting information.

I will be taking dashcam footage to use for testing the software. In the UK, there are no laws against filming with a dashcam provided it is installed safely in the car. The footage will be used for testing and then deleted once the project has been completed.

UNIVERSITY OF PORTSMOUTH

Certificate of Ethics Review

Project title: Vision based real-time lane detection and object recognition software for use in autonomous vehicles

Name:	Jack Sines	**User ID**:	901354	**Application date**:	25/10/2021 11:53:27	**ER Number**:	TETHIC-2021-101407

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are **Philip Scott, Matthew Dennis**

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

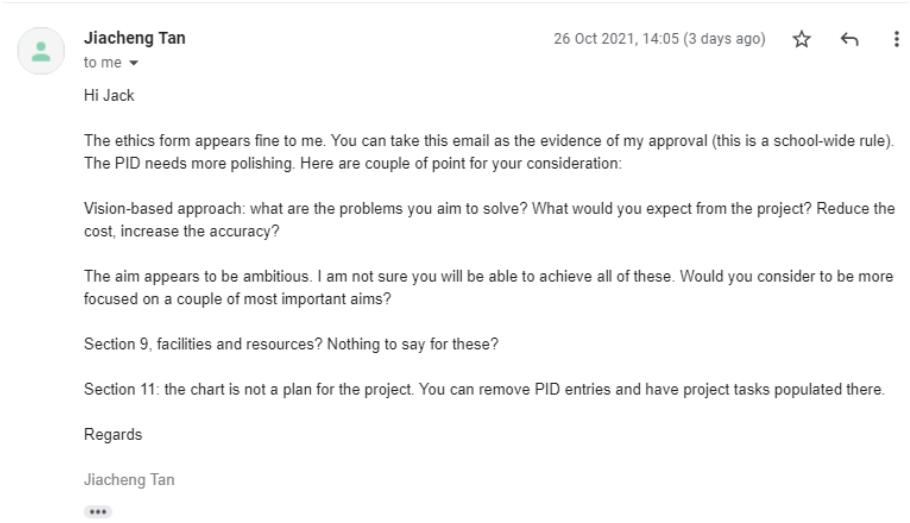
Which school/department do you belong to?: **School of Computing**
What is your primary role at the University?: **Undergraduate Student**
What is the name of the member of staff who is responsible for supervising your project?: **Jiacheng Tan**
Is the study likely to involve human subjects (observation) or participants?: No
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No
Are there risks of significant damage to physical and/or ecological environmental features?: No
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No
Does the project involve animals in any way?: No
Could the research outputs potentially be harmful to third parties?: No
Could your research/artefact be adapted and be misused?: No
Does your project or project deliverable have any security implications?: No

I confirm that I have considered the implications for data collection and use, taking into consideration legal requirements (UK GDPR, Data Protection Act 2018 etc)
I confirm that I have considered the impact of this work and and taken any reasonable action to mitigate potential misuse of the project outputs
I confirm that I will act ethically and honestly throughout this project

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor's signature: Date:



Jiacheng Tan 26 Oct 2021, 14:05 (3 days ago)

to me

Hi Jack

The ethics form appears fine to me. You can take this email as the evidence of my approval (this is a school-wide rule). The PID needs more polishing. Here are couple of point for your consideration:

Vision-based approach: what are the problems you aim to solve? What would you expect from the project? Reduce the cost, increase the accuracy?

The aim appears to be ambitious. I am not sure you will be able to achieve all of these. Would you consider to be more focused on a couple of most important aims?

Section 9, facilities and resources? Nothing to say for these?

Section 11: the chart is not a plan for the project. You can remove PID entries and have project tasks populated there.

Regards

Jiacheng Tan

References

Department for Transport. (2019). RAS50001: Contributory factors in reported accidents by severity, Great Britain, 2019. https://www.gov.uk/government/statistical-data-sets/reported-road-accidents-vehicles-and-casualties-tables-for-great-britain

National Highway Traffic Safety Administration. (2021, August 28). The Road to Full Automation. https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

Autocrypt. (2021, August 10). Camera, Radar and LiDAR: A Comparison of the Three

Types of Sensors and Their Limitations.

https://autocrypt.io/camera-radar-lidar-comparison-three-types-of-sensors/

Shchetko, Nick. (2014, July 21). Laser Eyes Pose Price Hurdle for Driverless Cars, Wall Street Journal. https://www.wsj.com/articles/laser-eyes-pose-price-hurdle-for-driverless-cars-1405969441

Christoph Domke & Quentin Potts. (2020, August 3). LiDARs for self-driving vehicles: a technological arms race. https://www.automotiveworld.com/articles/lidars-for-self-driving-vehicles-a-technological-arms-race/

Intellias Automotive. (2018, August 9). The Ultimate Sensor Battle: Lidar vs Radar. https://medium.com/@intellias/the-ultimate-sensor-battle-lidar-vs-radar-2ee0fb9de5da

Cosworth. (2021, October 10). Solid State Lidar.

https://www.cosworth.com/capabilities/digitalisation/solid-state-lidar/

Appendix B

'detected' class and 'get objects' function

(Please note that formatting has been slightly changed to fit the page)

```
import yolov5.detect

# Each object detected will have an object of this class created
class detected:
    def __init__(self, x1, y1, x2, y2, confidence, classification):
        self.x1 = float(x1)
        self.y1 = float(y1)
        self.x2 = float(x2)
        self.y2 = float(y2)
        self.confidence = confidence
        self.classification = classification

        # coordinates in format x1,y1, x2,y2, confidence, class
        # from top left corner

        # Classifications:
```

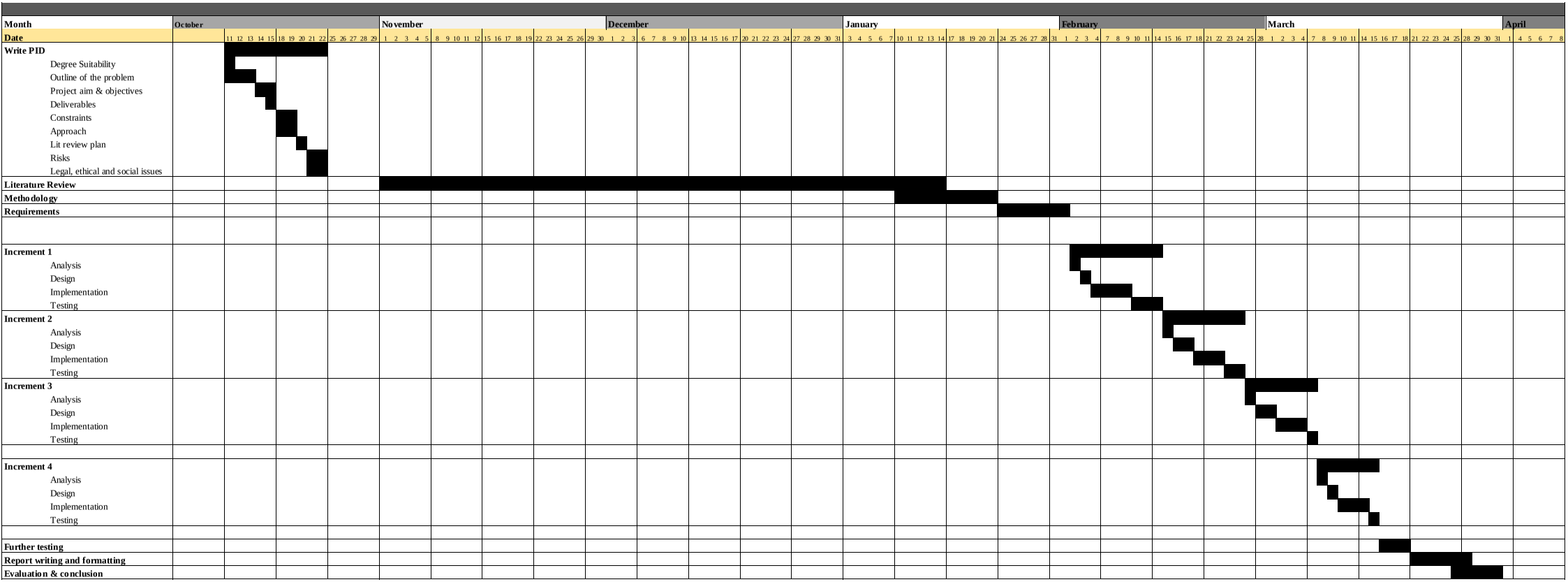
```
        # 0: pedestrian
        # 1: rider
        # 2: car
        # 3: truck
        # 4: bus
        # 5: train
        # 6: motorcycle
        # 7: bicycle
        # 8: traffic light
        # 9: traffic sign


# This method calls the YOLOv5s object detector and
formats the data before creating a class for each object

def get_objects(filepath, cpu):
    print('\nStarting object detection...')

    # The option to use CPU for detector is present in case of issues
    with unsupported or out of date GPU's

    if cpu:
        bb, t, img_bb =
        yolov5.detect.run('../yolov5/runs/train/exp3/weights/best.pt',
        filepath, device='cpu', imgsz=(256, 416))

    else:

        # the detect.run function defaults to GPU,
        so if it is compatible it will use the GPU
```

```
    bb, t, img_bb =
    yolov5.detect.run('../yolov5/runs/train/exp3/weights/best.pt',
    filepath, imgsz=(256, 416))

# initialise list of objects in image and
create a dictionary of object types

object_list = []
classes = dict(
    [(0, 'pedestrian'), (1, 'rider'), (2, 'car'), (3, 'truck'),
    (4, 'bus'), (5, 'train'), (6, 'motorcycle'),
     (7, 'bicycle'), (8, 'traffic light'), (9, 'traffic sign')])

# the following loop reformats the data from the yolo object
detector into x1,y1,x2,y1,confidence, classification

# each object detected then has a 'detected' object
created with this data

for i in bb:
    car = (str(i)).replace('tensor([', '')
    if "cuda" in car:
        car = car.replace('device=\'cuda:0\')', '')
    else:
        car = car.replace(')', '')
    car = car.replace(']', '')
    car = car.split(', ')
    x1 = car[0]
    y1 = car[1]
    x2 = car[2]
    y2 = car[3]
```

```
        confidence = car[4]
        classification = classes[int(float(car[5]))]
        obj = detected(x1, y1, x2, y2, confidence, classification)
        object_list.append(obj)
    print('Object detection complete.')
    # write image with bounding boxes to temp file
    cv.imwrite('./temp/detected.png', img_bb)
    return object_list, t, img_bb
```

Appendix C

Gantt chart plan


Month
October
November
December
January
February
March
April
Date
11 12 13 14 15 18 19 20 21 22 25 26 27 28 29
1 2 3 4 5 8 9 10 11 12 15 16 17 18 19 22 23 24 25 26 29 30
1 2 3 6 7 8 9 10 13 14 15 16 17 20 21 22 23 24 27 28 29 30 31
3 4 5 6 7 10 11 12 13 14 17 18 19 20 21 24 25 26 27 28
31 1 2 3 4 7 8 9 10 11 14 15 16 17 18 21 22 23 24 25
28 1 2 3 4 7 8 9 10 11 14 15 16 17 18 21 22 23 24 25 28 29 30 31 1
4 5 6 7 8
Write PID
Degree Suitability
Outline of the problem
Project aim & objectives
Deliverables
Constraints
Approach
Lit review plan
Risks
Legal, ethical and social issues
Literature Review
Methodology
Requirements
Increment 1
Analysis
Design
Implementation
Testing
Increment 2
Analysis
Design
Implementation
Testing
Increment 3
Analysis
Design
Implementation
Testing
Increment 4
Analysis
Design
Implementation
Testing
Further testing
Report writing and formatting
Evaluation & conclusion

Appendix D

Literature review search results

Date	Terms	Engine	Results	What is this search?	Result
1.11.21	"camera distance"	ebsco	1585842	Initial search filtering by source types: journals, books, magazines, reports, theses, ebooks and electronic resources	Too many results, many of which are not relevant
	"camera depth perception"	ebsco	453524	Trying to narrow down on just depth perception. Still filtering by source type (as above)	Still too many sources but some are useful. Many are quite specialised
	depth perception AND ("stereo" OR "multiple" OR "cameras")	ebsco	60448	Using boolean search to find papers relating to both depth perception and multiple cameras	Cuts number of papers down dramatically, but still far too many to review. There are some useful resources to be found in the first few pages of results
	"depth perception" AND "computer vision"	ebsco	5386	Trying to filter out any of the irrelevant sources	Again, cuts down number of results by a lot. Many of the sources here are more helpful
	"depth perception" AND "computer vision" AND "autonomous vehicles"	ebsco	102	Narrowing down to depth perception used in AV's Filtering to only journals	While this does cut the number of results down to 64, it also exludes anything not explicity related to autonomous vehicles.
2.11.21	"computer vision" AND "autonomous vehicles" AND ("depth perception" OR "depth sensing" OR "depth estimation")	ebsco	497	Another attempt at narrowing down to depth perception in Avs Filtering to journals and reports	Too many results, many of which rely on different sensors
	"stereo" AND ("vision-based" OR "vision based") AND "depth estimation" AND "vehicle" AND "distance"	ebsco	429	As above. Also filtering to just journals	
3.11.21	"depth sensing" AND "vehicles" AND "distance" AND "cameras"	ebsco	241	Boolean search filtering to journals	Intersting, relevant results but too many to go through
	"depth sensing" AND "autonomous vehicles" AND "distance" AND "cameras"	ebsco	63	As above	Less results but many not relevant
	"stereo cameras" AND "depth" AND "distance" AND "traffic" and "vehicles"	ebsco	407		
	stereo camera depth	ebsco	1995	As we haven't been able to find too much specifically relating to AV's, I'm broadening the search to all uses of camera depth perception (journals only)	Too many results, but useful papers
	comparison of autonomous vehicle sensors	ebsco	52494	As the above searches haven't been too successful, I'm trying to find a comparison of sensors in the hope that we can find more resources from this	Many results but from these I have found some useful key words to search for
	vision based autonomous vehicle	ebsco	12789	More of a broad search - filtering by journals only	Some useful papers but need to narrow them down
	vision based autonomous vehicle AND depth	ebsco	10330	Trying to find anything related to estimating depth - filtering by journals	Too many to search through but have found some useful resources from the first few pages
	"vision based" AND "autonomous vehicle" AND ("depth" OR "distance")	ebsco	2040		

Date	Terms	Engine	Results	What is this search?	Result
4.11.21	vision based autonomous vehicle	Google Scholar	773000	Initial search with google scholar	Too many results
	vision based autonomous vehicle depth sensing	Google Scholar	77900	Refining initial search slightly	
	stereo vision-based autonomous vehicle distance	Google Scholar	20500	Looking for instances of using multiple cameras for depth sensing	
	stereo vision autonomous vehicles	Google Scholar	65800	Looking for the phrase "stereo vision" in relation to autonomous vehicles	Too many results, but useful resources found on first few pages
	stereo vision distance measurement autonomous vehicles	Google Scholar	44700		
12.11.21	autonomous vehicles stereo vision distance measurement system	Google Scholar	45500		Found a useful paper on first page of results

Paper	Author(s)	Relevant?	Notes	Location	Read?
A stereo vision system for an autonomous vehicle	Donald B. Gennery	Yes	Goes over a few techniques for stereo vision with examples	Zotero - relevant papers	Yes
computer vision-based estimation of flood depth in flooded-vehicle images	Park, Somin Baek, Francis Sohn, Jiu Kim, Hyoungkwan	No	Uses NN to detect vehicles and the rest of the paper is on estimating flood depth	Zotero - relevant papers	Read
Distance measurement for autonomous vehicles using stereo camera	Abdelmoghit Zaarane, Ibtissam Slimani, Wahban Al Okaishi, Issam Atouf, Abdellatif Hamdoun	Yes	Uses 2 cameras as a single stereo camera to calculate distance to the vehicle in front	Zotero - relevant papers	Yes
Embedded Stereo Vision System for Intelligent Autonomous Vehicles	J. Koglerl, H. Hemetsberger, B. Alefs, W. Kubingerl and W. Travis	No	Goes over an algorithm for lane recognition and object detection - likely not useful as we will only be locating the vehicle in front for now	Zotero - relevant papers	Read

Error analysis in a stereo vision-based pedestrian detection sensor for collision avoidance applications	David F. Llorca, Miguel A. Sotelo, Ignacio Parra, Manuel Ocaña and Luis M. Bergasa	Yes	Uses stereo cameras for to estimate depth and goes over the some of the issues, limitations, etc.	Zotero - relevant papers	Mostly
Inter-vehicle distance estimation method based on monocular vision using 3d Detection	Zhe, Ting Huang, Liqin Wu, Qiang Zhang, Jianjia Pei, Chenhao Li, Liangyu	No *	Uses monocular vision *If I wanted to compare stereo and mono this could be good	Zotero - relevant papers	Read
Object distance measurement by stereo vision	Manaf A. Mahammed, Amera I. Melhum, Faris A. Kochery	Yes	Uses triangulation with stereo cameras to estimate distance	Zotero - relevant papers	Yes
Onboard Real-Time Dense Reconstruction in Large Terrain Scene Using Embedded UAV Platform	Zhengchao Lai, Fei Liu, Shangwei Guo, Xiantong Meng, Shaokun Han and Wenhao Li	No	Uses monocular vision as being used in UAV's so need to estimate large distances (stereo cameras are only able to do small distances)	Zotero - relevant papers	Read
Power pylon detection and monocular depth estimation from inspection UAVs	Araar, Oualid Aouf, Nabil Dietz, Jose Luis Vallejo	No	Monocular camera	Zotero - relevant papers	Read
Real-time Depth Estimation Using Recurrent CNN with Sparse Depth Cues for SLAM System	Lee, Sang Jun Choi, Heeyoul Hwang, Sung Soo	No	CNN architecture for SLAM system to estimate depth	Zotero - relevant papers	Read
Stereo Vision enhancements for Low-Cost outdoor autonomous vehicles	Alonzo Kelly and Anthony Stentz	Yes	Discusses techniques that could be used to Improve efficience/reliability of dense stereo	Zotero - relevant papers	Skim read
Stereo Vision-based vehicle detection	M. Bertozzi, A. Broggi, A. Fascioli, S. Nichele	Yes	Methods for localisation and tracking with a stereo vision system	Zotero - relevant papers	Read
Vision-based distance measurement in advanced driving assistance systems	Ding, Meng Zhang, Zhenzhen Jiang, Xinyan Cao, Yunfeng	No*	*Unless I want to mention NN, which I probably should Depth map estimation based on smi-supervised learning to train a deep learning network	Zotero - relevant papers	Read
Collision avoidance method of autonomous vehicle based on improved artificial potential field algorithm	Feng, Song Qian, Yubin Wang, Yan	No	cant access rn		Read
Efficient Vehicle Detection and Distance Estimation Based on Aggregated Channel Features and Inverse Perspective Mapping from a Single Camera	Jong Bae Kim	No	Single camera, but could be useful to have a quick read through	Zotero - FYP - Journals/Data	Read
Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations	Daniel J.Fagnant, Kara Kockelman	Not for lit review*	*Has good background info which can be used for intro - also relatively recent (2015)	Zotero - FYP - Journals/Data	Read
Sensors and Sensor's Fusion in Autonomous Vehicles	Andrzej Stateczny, Marta Wlodarczyk-Sielicka Pawel Burdziakowski	Maybe?*	*Worth a quick read - could be useful for some info on various sensors (req gathering maybe?)	Zotero - FYP - Journals/Data	Read
How Neural Nets, Computer Vision and Autonomous Vehicles are Related	Ashwin Singh	Not for lit review*	*background info on sensors, manafacturers, etc.	Zotero - FYP - Journals/Data	Read

Vision-based Autonomous Vehicle Recognition: A New Challenge for Deep Learning-based Systems	AZZEDINE BOUKERCHE and XIREN MA,	Not really*	*Review of deep learning models for vision based av recognition	Zotero - FYP - Journals/Data	Read
Vision-based long-distance lane perception and front vehicle location for full autonomous vehicles on highway roads	Xin, L. Xin, X. Bin, D.		cant access rn		Read
DISTANCE ESTIMATION FROM STEREO VISION: REVIEW AND RESULTS		Yes	very similar project	zotero-fyp-relevant papers	Part read
The perception system of intelligent ground vehicles in all weather conditions			Thorough comparison of all sensors with advantages, disadvantages and uses	Papers folder	Read
Vision-based vehicle detection and counting system using deep learning in highway scenes			Example of using neural networks to detect vehicles		Read