University of
HUDDERSFIELD

'Investigate the efficacy of using machine learning techniques to develop a system capable of autonomously controlling a locomotive'.

Word Count: 24,218

1 CONTENTS

2 Abstract 4
3 Introduction 4
3.1 Product Description 4
3.2 Aim 4
3.3 Objectives 4
4 Literature Review 5
4.1 Driver Awareness and Processing Speed 5
4.2 Non-Neural Networked Approaches to Speed Sign Detection 6
4.3 Neural Network-Based Approaches 9
4.4 Hybrid Approaches 10
4.5 Electronic Approaches 12
4.6 Object Detection Networks 13
4.7 Conclusion 13
5 Methodology 14
5.1 Software Development Process 14
5.1.1 Software Development Methodology 14
5.2 Requirements Engineering 15
5.2.1 Requirements Elicitation 15
5.2.2 Requirements Specification 16
5.2.3 Requirements Modelling 17
5.3 Software Design 18
5.3.1 System Modelling 18
5.3.2 Wireframing 18
5.3.3 System Architecture 19
5.3.4 Design Patterns 20
5.4 Software Construction 21
5.4.1 Construction Tools 21
5.5 Software Testing 22
5.5.1 Test Process 23
5.5.2 Testing Tools 24
5.6 Professional, Legal, Social and Ethical Issues 24
5.6.1 Professional Issues 24
5.6.2 Legal Issues 25
5.6.3 Ethical Issues 25

5.6.4 Social Issues 26
5.7 Timeline 26
6 Requirements 27
6.1 System Specification 27
6.2 Functional Requirements 27
6.2.1 Main Application 27
6.2.2 Settings Menu 31
6.3 Software Qualities 32
6.3.1 Safety 32
6.3.2 Maintainability 32
6.3.3 Performance Efficiency 33
7 Software Design 34
7.1 Requirements Modelling 34
7.1.1 Requirements Diagram 34
7.1.2 Use Case Diagram 36
7.2 Application Modelling 46
7.2.1 Application Wireframing 46
7.2.2 Class Diagram 47
7.2.3 Activity Diagrams 48
7.2.4 Sequence Diagrams 51
7.3 Architecture Modelling 56
7.4 Software Testing Plan 56
7.4.1 Application Testing 56
7.4.2 Test Plans 57
7.5 Dataset Specification 61
7.6 Model Training Methodology 62
8 Implementation 63
8.1 Sprint Planning 63
8.2 Sprint 1 – Dataset Creation 64
8.3 Sprint 2 – Dataset Creation Cont. & Interface Prototypes 65
8.4 Sprint 3 – Model Training and Interface Functionality 68
8.4.1 Training Issues 70
8.5 Sprint 4 – Detection System 70
8.6 Sprint 5 - Control System & Integration 70
8.6.1 Control Prototype 70
8.6.2 Multithreading Issues 71

8.6.3 Smooth Acceleration & Braking 71
8.7 Sprint 6 – Application Testing 71
9 Software Testing 72
9.1 Application Testing 72
9.1.1 Automatic Locomotive Control System 72
9.1.2 User Interface 75
9.2 Sign Detection System 78
9.3 End-to-End Testing 79
9.4 Unit Testing 80
10 Discussion: Review and Reflections 80
10.1 Evaluation of the Product 80
10.2 Limitations 80
10.3 Evaluation of the Project & Personal Performance 81
11 Summary and Conclusions 82
11.1 Future Plans 82
12 References 83
13 Appendix 87
13.1 Project Proposal Form 87
13.1.1 Summary of the Problem 87
13.1.2 Project Objective 87
13.1.3 The Product 88
13.1.4 The Clients 88
13.1.5 Activities to Solve the Problem 88
13.1.6 Time Plan 88
13.2 Ethical Review Form 89
13.3 The Agile Manifesto 92
13.4 Maintainability Index 93
13.5 Model Training Python Script 93
13.6 Model Validation Python Script 93
13.7 Testing Evidence 94
13.8 Speed Sign Reference 104
13.9 End-to-End Test Results 115

2 ABSTRACT

Within this report, we will be investigating the creation and implementation of an application which can control a locomotive using machine learning techniques to locate and identify route speed limit signs. The main purpose of this application is to implement a system capable of navigating routes without extensive and expensive modifications, such as the installation of radio beacons, which would usually be required to operate a train automatically whilst maintaining a high level of safety.

A YOLOv8 object recognition algorithm was paired with Google's 'Tesseract OCR' technology to produce a proof-of-concept which could connect to a simulator and proves that an installation of this type would be capable of identifying speed signs and controlling a locomotives speed accordingly. Along with these findings, we have also established some limitations which an installation of this type suffers over other more common radio control beacons such as night-time operation. This application can pave the way for research into fully autonomous agents capable of operating rail locomotives.

3 INTRODUCTION

3.1 PRODUCT DESCRIPTION

One of the main causes of accidents on railways is derailments, this can commonly be caused by over-speeding. In some cases, this can have catastrophic consequences. The 2016 Croydon tram disaster was one such example, caused by a tram exceeding the allowed line speed due to a fatigued driver becoming incapacitated. This project aims to prevent this by developing an application which can integrate with a locomotive and manage speed and braking accordingly using machine learning techniques to identify speed signs. Currently 'automatic train operation' [ATO] requires costly modification of the route and additional lineside equipment to enable safe operation however a system which is capable of interpreting speed signs would be a step towards having completely autonomous trains on any route without extensive re-engineering works.

3.2 AIM

'Investigate the efficacy of using machine learning techniques to develop a system capable of autonomously controlling a locomotive.'

3.3 OBJECTIVES

- *Identify stakeholder requirements for the automatic train control system.*
- *Explore design solutions for an automatic train control system including software architecture.*
- *Develop a prototype automatic train control system & interface.*
- *Conduct testing and evaluation of the capabilities of the developed automatic train control system.*
- *Evaluate the outcome and develop comments on further research which could be performed.*

4 LITERATURE REVIEW

Kyriakidis et al. (2015) noted that 73% of the studied accidents were caused by drivers, 70% of which were caused by signal passed at danger incidents [SPAD]. The other aspect of this study also pointed out that 13.1% of accidents were caused by distraction and 15.4% were caused by unfamiliarity, behind safety culture they are the two leading causes of rail accidents studied.

Signalling has been worked on extensively through the implementation of Automatic Warning Systems [AWS] and the limited investigation into Automatic Train Protection [ATP]. There has been limited research into speed limit-based driver vigilance devices. Currently, drivers rely on route knowledge and bulletins to inform them of any changes in the speed limit, when combined with fatigue of loss of situational awareness this can become problematic as was demonstrated in 2018 when a driver of a train from Aberdeen to London exceeded the speed limit by over 100mph (Adams, 2022).

With the rise of computer vision in recent years there has been some research into the implementation of this to improve safety and operational reliability on railways, however, this has mainly been directed towards signalling systems rather than speed limitation or driver vigilance devices which could be a development to move operating companies closer to automatic train operation (ATO). This also applies to the implementation of a system which can perceive and act upon railway speed signs, however despite this, there is a larger amount of material available which performs a similar task on roads.

Due to the interest in autonomous vehicles, extensive research has been performed into the application of this technology on road signs, some of which will be referred to in this literature review as it could provide insight into additional techniques or different implementations of speed sign recognition as identification of speed limits is key to the safe operation of rail and road vehicles alike. It can also be mentioned that UK and European road signs share many visual characteristics with UK railway speed boards therefore it is highly likely that these systems could be applied to rail vehicles too.

4.1 DRIVER AWARENESS AND PROCESSING SPEED

Although not key to the chosen research area this gives good evidence as to the requirement of the proposed system. One such example is given by Li et al. (2006), who used a simulator based on a stretch of line between Manchester and Handsforth to evaluate different drivers' recognition ability for a range of different signs and signals at different approach speeds. A total of 57 drivers from multiple different rail operators participated. A range of different ages and experience was also represented. This also included a full range of different signs including speed signs. One of the most interesting outcomes from the study is that at 62mph drivers detected these speed boards over 10 seconds before the front of the train passed the location. Which equates to approximately 270m. At 124mph this drops significantly, time remaining at recognition decreases to below 0 seconds which indicates the train had passed the sign by the time the driver correctly identified it. Although drivers do not strictly rely on these speed boards for indication of line speed, they are reliant on route knowledge as is mentioned in "Train Driver's 'Lapse'" (1955), in some cases a driver may become disoriented or distracted at which point they may not be aware of their position which can have catastrophic consequences.

4.2 Non-Neural Networked Approaches to Speed Sign Detection

Non-Neural networked approaches for sign classification have been heavily researched in recent years due to their near zero training requirement and high accuracy in most conditions. “Neural network-based approaches require extensive sample collection under different conditions and extensive training time which makes this approach time-consuming and sometimes ineffective” (Liu et al, 2010).

The first type of algorithm predominately used for region-of-interest [ROI] generation for speed sign detection was the Hough Circle Transform [CHT], research by Damavandi & Mohammadi. (2004) uses a hierarchical CHT algorithm to attempt to increase the accuracy of the base algorithm, a Multi-Layer Perception [MLP] network is then used to identify the speed limit, a comparison of different algorithms is noted where MLP is selected due to its accuracy of 99.4% which could indicate that it is over-fitting due to the relatively small training set size of 66 items. However, it can also be noted that MLP is the fastest neural network listed, at 77ms compared with the next fastest being 125ms, the accuracy however, is significantly worse. The system performed best between 2-10 meters, where the accuracy was more than 90%, this decreases significantly with additional distances up to 20m where the accuracy is less than 5%. This contrasts with the findings of Barnes et al (2008) who developed a similar system using radial symmetry detection which detected signs with a reasonable level of accuracy out to 20m. This is likely due to the use of radial symmetry detection as the two studies used similar camera resolutions, 480x640 vs 320x240, with the system proposed by Damavandi & Mohammadi using a higher resolution input image.

A similar approach was taken by Biswas et al. (2014) who also used CHT. This test was focused on the identification of road signs; these have comparable properties and operating speeds to rail networks therefore they are compatible when evaluating the efficiency of a given approach. This approach achieved 98% accuracy when locating and classifying road signs on a testing set of 210 images. The test was conducted under different light and weather conditions thus also increasing the reliability of the study, it is common that adverse weather conditions can seriously affect the accuracy of computer vision applications due to the additional clutter. This increase in accuracy between the studies of Biswas et al. (2014), and Agudo et al. (2016), could be caused by a range of factors from differences in the clarity of signs in the testing set due to the on-road application or could be due to the use of a slightly different implementation of Hough transform causing the system to become slightly more accurate. This system employed a Support Vector Machine [SVM] to classify the digits similarly to the actions of Agudo et al. (2016). However, the sign was binarized using an Otsu thresholding algorithm and then resized to 50x50 pixels rather than 10x10 thus reducing the adverse effects of clutter as there are more pixels present in the processed image. Digits are then segmented rather than identified as a whole. This approach is beneficial as it requires that the SVM only be trained to recognise digits 0-9 to identify any number. This is also beneficial as it allows the system to be much more accurate as fewer classes must be trained thus simplifying the identification process, provided that the system implemented to crop individual digits is robust and accurate. This demonstrated an average accuracy of 98% across all testing patterns thus it could be inferred that the solution proposed is somewhat superior given the extended test pattern of 210 images.

Another similar methodology was demonstrated by Agudo et al. (2016) who built a system to identify railway speed signs based on in-cab video. The system was again based on the CHT algorithm to establish ROIs. The canny edge detection algorithm was employed to generate object outlines for each frame. A series of points were then generated using Hough-based voting, this is

decided based on the circularity of the outline, candidates have an approximate centre. As the Hough algorithm can only identify regions of interest. A second SVM classifier is then employed to identify the digits within the sign unlike the approach of Biswas et al, numbers are identified as whole. Additional image processing is required to extract the digits from the bounding box generated by the previous classifier, this is one main downside of this approach. Despite the ability of the system to recognise speed signs with no training, it relies on extensive image processing which can be compute-heavy and can be seriously affected by adverse conditions of overlapping objects. Images are also resized to 10x10 before being classified, this reduction in pixel density can cause any clutter to affect the overall shape of the image requiring a more robust classifier to correctly identify the digit. It can also be noted that this approach classified the entire number, this requires more training numbers compared to a system which only classifies digits 0-9. Despite this and accuracy of 95% was noted based on a test set of 48 objects. This is a limited dataset however the testing set contained over 3 hours and 43 minutes of video.

An alternative and potentially more efficient approach to region of interest generation for sign detection was proposed by Barnes et al. (2008) who detected speed signs using radial symmetry detection rather than CHT. The proposed algorithm works by starting at the edge of the object and voting inwards in a single direction until reaching the estimated centre point, other methods vote in multiple directions, 'this single point inward voting reduces the number of voting stages thus also increasing the overall efficiency of the algorithm' (Barnes et al, 2010). This research exclusively focuses on ROI generation rather than sign classification however it can be noted that this approach is not overly accurate compared to previously discussed approaches achieving 86% accuracy. As was previously noted these approaches to ROI detection require image pre-processing this approach uses a Sorbel edge detection algorithm rather than the canny edge detection, which is commonly used by other approaches, it was noted by Joshi & Vyas. (2018) that the Sobel edge detection algorithm is much less efficient taking 50 seconds to process a 1920 x 1080 HD image, whereas the canny algorithm was able to process it in 11 seconds, also noting significantly less peak CPU usage (32% vs 60%). This is not ideal for real-time processing as this can have a knock-on effect on identification time. This study was primarily focused on identifying the location of a sign within a frame of video therefore cannot be directly compared to any of the previous studies whose results were a compound outcome of the location and classification systems. One interesting finding was that the maximum distance at which a regular road sign could be identified using this method was 20m (with larger highway signs being detected at 30m). This distance could likely have been improved with the use of a higher definition camera as it was only capturing images at 320x240 pixels. Increasing the resolution of the camera would allow more pixels to be captured thus more detail at range, at the cost of computation. It is highly possible given the real-time requirement of this system that an increase in video resolution would constitute a reduction in frame rate.

Another approach which was demonstrated by Liu et al. (2010) which follows similar methods as outlined by Barnes et al (2008), using fast radial symmetry [FRS] to identify speed signs generating ROIs based on the circularity of an outline of a given object within the frame. The Objects which exceed the expected circularity are ignored which is different to the method used by CHT. Barnes et al stated that it is known 'that due to how CHT operates when presented with large images this algorithm can be slow to process all objects'. An alternative method was proposed to try and optimize the process of generating ROIs. This is a key finding as real-time speed detection systems will have a wide field of view and aim to use high-quality cameras so that signs can be identified at a distance therefore the images which they are required to process will likely contain many pixels. Once the ROI has been established the approach demonstrated uses a fuzzy pattern matching algorithm to predict what digits are contained within the ROI. Additional image processing is applied

to each region of interest to identify the largest area inside the circle which does not contain the outline of the speed sign (red pixels vs white pixels), the newly cropped image is then resized to 20x40 pixels. This is different to the approach of Agudo et al. (2016) who removed the outline of the speed signs using a black masking layer applied to a binarized version of the sign. This method is likely more efficient as less transformation of the image is required. The use of fuzzy pattern matching in this case could be more efficient than other approaches proposed as the digit is classified based on a pattern rather than having to dedicate additional processing time to splitting each digit into its parts. Despite this it is vulnerable to incorrect recognition as the area to be classified is larger. It is noted in the paper that different conditions can make two similar templates difficult to distinguish, this was remedied using 'local feature vectors' to compare the digit more finely with that of its template, the closer the match the more likely that it is of the identified type. One upside to this approach is that "It can be easily extended by simply adding new templates to the existing list. To overcome the influence of character tilt or visibility and further improving the matching accuracy" (Liu et al, 2010, p. 264). As a result, this is a significantly less intensive method to develop once the templates have been created. Despite all the upsides to this method, the testing results showed an accuracy of 96.22% in ideal conditions, dropping to 89.08% at night. As expected, the recognition rate was significantly worse in rain or fog. This approach using FRS demonstrates a similar level of accuracy compared to the research by Biswas et al (2014) and Agudo et al. (2016), it is notable that the average recognition rate is somewhat lower compared to the results found by Biswas et al. (2014) it should also be noted that the test was performed across a larger testing set of 662 signs vs the 213 signs in the testing set of Biswas et al. (2014). This indicates that this study is much more reliable due to the significantly larger testing set likely containing a much wider range of speed signs however in a real-time processing environment accuracy is key. It can also be noted that the accuracy of this system was significantly improved compared to the findings of Barnes et al. (2008) despite a similar methodology, this is likely partially due to refinement of the FRS algorithm and the improvement in knowledge of sign classification methods due to the 2-year gap between the publishing dates of the two papers (2008 vs 2010). During the testing of this system, it ran at 15ms/frame, which equates to approximately 30fps, with the camera capturing images at 240x320 pixels, this is comparable to the resolution used by Barnes et al. (2008).

Another novel approach was proposed by Guerrieri & Parla. (2019) who used the canny algorithm to convert the image into a series of edges. However, they proposed the use of the Freeman chain code method to directly generate bounding boxes around the signs rather than establishing a series of ROIs like the previously discussed approaches. This used a series of templates for each sign type which were matched to elements within the image. This was again tested on a series of frames from an in-cab video achieving 96.92% accuracy across 400 signs. The main downside to this alongside other template matching techniques is that the algorithm is required to perform the matching exercise with all the stored templates, which with many templates can take quite a lot of processing power and time. This is not ideal for a real-time scenario.

Greenhalgh & Mirmehdi. (2012), proposed a solution based on maximally stable external regions [MSER] as a method of generating regions of interest, the implementation also used an SVM to classify these ROIs into different traffic sign classes. This research focuses on a wider range of traffic signs, the full set of UK road signs rather than exclusively speed signs, these techniques can still be employed to identify any other type of speed sign given that the base of the research is focussed around MSER detection. This system is designed to establish regions which maintain their shape when thresholded at different levels, it is more robust when detecting ROIs in different contrast and lighting conditions, this is something which other methods such as CHT are vulnerable to due to their reliance on pre-processed outlines. This type of algorithm requires a threshold range to be defined,

the larger the number of thresholding steps the higher accuracy, which was tested using a range of threshold levels and plotting them against the accuracy and processing time, there is little difference in accuracy between 24-40 thresholds however there is a steep increase in processing time, starting at 50ms increasing to 75ms at 40 thresholds. It is difficult to establish from the results table however it seems the accuracy is not significantly higher than at the selected 24 thresholds, hence its selection. This may not seem like a significant difference however at 50ms processing time this equates to 2fps vs 0.8fps at 75ms, this does not yet include the classification time. The research uses an additional stage of classification to establish the type of sign as UK road signs come in several different shapes and colours, this will not be discussed as it does not relate to the implementation of speed sign detection. This implementation could be carried over to an implementation designed to identify railway speed signs due to their similarities with UK road signs.

4.3 Neural Network-Based Approaches

The research into the use of Convolutional neural networks [CNN] as methods of identifying railway speed signs has not been extensively researched however due to the prevalence of autonomous vehicles there has been significantly more research into the on-road application of this technology.

One such example is the research performed by Li & Wang. (2019) who developed a system for real-time traffic sign recognition based on a neural network. Their approach used a single-stage classifier to identify road signs based on their attributes, for example, a 50mph sign would be a different class to a 20mph sign. This used the German Traffic Sign Recognition Benchmark [GTSRB] Dataset. The dataset contains a range of already pre-labelled traffic signs. This has the benefit that the algorithm is quicker to identify a speed sign as additional processing is not required as the speed is directly identified from its features. The main downside to this approach is that a significantly larger dataset is required for accurate identification. The other main disadvantage to this approach is that achieving a high level of accuracy for signs can be difficult as the visual appearance of all speed signs is similar, commonly a circular sign with a red outer ring and a white inner circle containing the speed limit, the only difference between all these signs is the contained digits. Therefore, in some conditions, a classifier might find it difficult to determine what class the speed sign is leading to false positives this is demonstrated by their results where they achieved 84% accuracy when testing their model against the Tsinghua-Tencent 100K dataset which contains over 100,000 traffic signs. A custom network model was used to classify the signs. As far as can be established their classification subnet was not based on any specific existing architecture. However, it was able to classify all 43 classes contained within the GTSRB dataset. They demonstrated a classification accuracy of 100% on the validation subset, this is concerning and could potentially indicate that the model is over-fitting. Validation of the model was performed using the Tsinghua-Tencent 100K dataset rather than a real-life test, thus making it difficult to evaluate its implementation or potential.

One common approach when exclusively using neural networks is to use two sub-networks. One such paper by Wang et al. (2020), proposes that there should be two separate subnets used, the first is used exclusively to identify the location of a speed sign. The second is used to classify the value of the sign detected. This separation of the classification logic creates more potential problems as two separate models are being used for sign classification, on the other hand, this also allows scope for increase classification accuracy as there are extensive datasets for both sign recognition and digit recognition. The classification subnet, which was proposed uses a DenseNet algorithm, which shares similarities with ResNet, but has a slightly different structure (Wang et al, 2020). Also noted was DenseNet's reduced processing requirement with minimal loss in accuracy compared to ResNet, whilst also having a strong anti-overfitting capability when used in combination with limited training

data (Wang et al, 2020). Classification accuracy was more than 95%, while also demonstrating the ability to correctly identify and classify signs which were partially obscured by foliage. One downside to the split classifier approach was noted by Li & Wang. (2019) is that they are much more prone to issues when a generated bounding box is not correctly positioned on a sign, this can make subsequent identification of digits on a sign more difficult without additional edge refinement. This is one of the main reasons why using a single-stage classifier is suggested. This issue can be partially mitigated by working on the basis that a should have good accuracy and be trained to recognise the entire sign rather than the centre where the digits are contained.

Another approach notably acknowledges that sometimes the action of sign classification is applied in a resource-restricted environment, the research by Zhang et al. (2020), looked to alleviate this partially by transferring knowledge between two neural networks, using 'knowledge distillation'. This system uses a more heavyweight model called the teacher network and transfers it to the student network which is a more lightweight model (Zhang et al, 2020). They noted something key, which has not been mentioned in any other papers previously discussed is that "larger and deeper networks require more resources, graphics processing units (GPUs) [20] are commonly used to help speed up computation. The strong representation ability of convolutional neural networks arises because of their millions of trainable parameters; for example, AlexNet has 60 M parameters, while VGG16 has 138M parameters. However, the reduced computing power and storage space of on-board equipment or wearable devices cannot support the operational resources required by these complex networks" (Zhang et al, 2020, p. 370). This is an interesting observation as no previous research has mentioned this, despite it being a serious real-world consideration for neural networked classification problems. Their research focusses on the GTSRB dataset, this means that the image is already pre-cropped therefore in a real-life application another method would have to be used to identify the location of a sign within a frame. Despite this notion that the student model should be able to run on a lightweight machine, it is noted in the experimental results section that the experiments were conducted on a computer running an Intel Xenon processor and Nvidia Titan X, these components were considered top of the range hardware at release, the Titan X having an RRP of $999 (TechPowerUp, n.d) at launch in 2015. Not to mention that this would also require a full-size computer to accommodate these components, which does not lend itself to limited storage space of on-board equipment. Despite this downside accuracy of 98.89% was noted which makes it one of the more accurate approaches despite its complexity

4.4 Hybrid Approaches

One approach developed by Kundu & Mackens. (2015), uses a hybrid approach to speed sign recognition. This system aims to identify American speed signs which differ slightly from speed signs used in other countries or on UK railways as they tend to have a rectangular shape rather than a rounded shape, the speed limit is still defined by black letters on a white background. It is mentioned that research by Greenhalgh & Mirmehdi. (2012) also implemented a similar approach to select candidate regions based on MSER, similarly, MSER is used to select the ROIs within the frame which is likely to contain a speed sign. These regions are processed by an artificial neural network which is designed to classify each sign based on the speed limit shown. This approach demonstrated an accuracy of 98.04% with an average processing speed of 40fps which is significantly more efficient than the system proposed by Liu et al. (2010) who averaged approximately 4fps when running on a comparable device. This approach differs from the solution proposed by Greenhalgh & Mirmehdi. (2012) as it uses a neural network to classify signs rather than a trained SVM. As already stated, the machine learning-based approach achieved results of above 98% across a series of

testing videos with contained over 12,300 individual frames containing speed limit signs (Kundu & Mackens, 2015) whereas the SVM-based approach only achieved an average accuracy of 86.8% (Greenhalgh & Mirmehdi, 2012), it should be noted that these tests were performed on different datasets with different physical characteristics, which could be contributing factor, as was previously discussed when analysing the findings of Greenhalgh & Mirmehdi. (2012) out of all the errors a large portion came as a result of undetected signs (9) rather than miss-classification (5), this is likely due to the increased visual complexity and relatively smaller size of UK road signs as can be seen in Figure 1 compared to US speed signs as can be noted in Figure 2, US road signs have a much larger white surface area when compared to UK speed signs.

Figure 1 - Example of a UK road sign based on the examples given by Greenhalgh & Mirmehdi (2012, p. 1500)

Figure 2 - Example of a US road sign based on the examples given by Kundu & Mackens (2015, p. 1853)

A different hybrid approach was demonstrated in research by Gomes et al (2017) who proposed a system capable of running on a mobile phone which could identify speed signs in real-time as a means of increasing driver safety and reducing speeding. Their approach uses a 'boosted classifier' to generate regions of interest and an SVM to identify digits. The 'boosted classifier' works by scanning the frame and checking to see if there is potential that the pixels within the current search area could contain a speed sign, as is highlighted when detecting an object with an unknown size it is sometimes possible that the search must be repeated multiple timed with different size search areas. The main upside of this approach is also its main downfall, it is quite inefficient compared to

other approaches which use a single pass. Despite this, when testing the system was installed onto an android phone, with images being captured between 20 and 30 fps. As this is a more modern device it is likely that any inefficiencies are less easily detected compared with some of the older papers. The other upside to the use of a 'boosted classifier' is that for a machine learning-based approach it uses a small training set, as stated: "The positive examples included about 1,000 images with speed signs" (Gomes et al, 2017, p. 579), which gave an accuracy of 90.41% which is extremely high given the small training set. When combined with their implementation of an SVM 99.87% accuracy was noted which is extremely good given the testing set of over 12,000 images of speed signs.

As has been discussed previously Hough Circle Transform is a popular method of feature extraction, a system by Mata-Carballeira et al. (2018) aims to combine this approach with an Extreme Learning Machine [ELM], this differs from a regular neural network via its architecture. As noted, CNNs show good accuracy rates but also have many issues including excessive computational cost, ELMs aim to remedy this using an architectural change, which also coincidentally provides faster training times (Mata-Carballeira et al, 2018). The process of generating regions of interest from video frames is almost identical to all other approaches using CHT such as Biswas et al (2014) & Agudo et al. (2016). Canny edge detection is applied to the video frame then the CHT algorithm is used to identify circular objects within the frame. Once completed, these regions of interest are passed to the ELM for classification. A range of different architectures were tested, the highest accuracy seen was 91.65% which is low when compared to other machine learning or hybrid-based approaches which have been evaluated. Given that other approaches using CHT have had quite high recognition rates it is likely that the ELM is the cause of the inaccuracy within the system, ELMs are a relatively new concept having only been proposed and documented in 2006 by Huang et al (2006).

A completely different approach was proposed by Torresen et al. (2004) who used colour separation rather than an edge detection filter as is used by many of the other systems already discussed. Template matching is used to generate regions of interest. Once the ROIs have been generated, a neural network is then used to classify the digits within the speed sign. To test the system a database of 198 different images was used, a range of image manipulation techniques was also used to make the signs more difficult to distinguish. The system correctly classified 90.09% of the signs which it was shown, whilst maintaining a low false positive rate of 6%. This paper has comparable results to some of the findings published significantly more recently. The main downside to this colour-based approach was outlined by Liu et al. (2010) which is that colour-based approaches are significantly more vulnerable to different lighting conditions where the colours are less vivid, which is a core flaw based on the notion that the system should be robust and always maintain high accuracy.

4.5 Electronic Approaches

A paper by Guibert et al. (1993) outlines a purely electrical engineering-based approach to road sign recognition, although this is not strictly applicable to the research areas it raises some interesting concepts which relate to the other literature reviewed. This approach utilizes a series of laser diodes and lenses to perform a purely hardware optical function, which ultimately allows a system to identify a speed sign though a designated observation window. The system does not return any physical value only a location on a template which matches the viewed speed sign. The interesting aspect is where it is mentioned that “the zero digit of the references are suppressed because they appear in every road sign to be recognised” (Guibert et al, 1993), this is possibly partially due to a technological limitation however it is a key oversight made by other papers where digits are

recognised individually, for example, the system developed by Biswas et al. (2014) is designed to recognise road sign digits individually. It in many countries speed signs increase by an increment of 10, up to the national speed limit. This is similarly replicated on railway networks, instead in increments of 5, to save processing time a system could be developed where a second digit classifier is only required to decide if the digit is a 0 or a 5, increasing accuracy and reducing the total number of possible classes the system would be required to recognise. Despite the relatively early stages of the prototype, it is also mentioned that theoretically for each video frame, the scene could be compared with 100 different references. As this is a solid-state implementation there is not any additional processing time required, the only limitation is likely to be the speed at which the system can reset between frames which is significantly quicker than any other system proposed. The main downside is the expertise and spatial requirements of this type of system.

4.6 Object Detection Networks

There is a range of different neural network architectures which can be used for object detection purposed, one of the most recent innovations is the proposition of the You Only Look Once [YOLO] network, the paper by Redmon et al (2016) states that YOLO changes the way which object detection systems operate, rather than repurposing classifiers to perform detection they have instead developed a neural network which views detection as a regression problem, instead predicting bounding boxes and class probabilities straight from the source image. They have noted that this model gets enhanced performance, achieving 14 frames per second. 'Whilst achieving double the mAP of other real-time detectors'. When tested against other real-time processors such as Deformable Parts Model [DPM], a good inference speed enabling 100 FPS was achieved however the mAP was extremely low at 16.0. Other detectors such as Fast-R CNN were also tested achieving 0.5 FPS and 70 mAP, which is better but still does not match the performance of the YOLO model which achieves 45 FPS with a mAP of 63.4 therefore a trade-off for accuracy must be made however the test results have demonstrated that YOLO can give significant processing gains which is ideal for a real time processing scenario.

4.7 Conclusion

Throughout this review, a range of different solutions to the issue of traffic sign detection and classification have been evaluated, and several main themes have been established with both neural networked and non-neural networked based approaches.

The primary finding of this review is that algorithmic-based approaches to object detection are sometimes more accurate and more efficient than neural networked-based approaches but also have more complexity. The most popular algorithmic approach is the Circle Hough Transform, one of the earliest implementations of which was by Damavandi & Mohammadi. (2004). Notably, several different adaptations of this have been created aiming to increase efficiency and accuracy. Despite this, sometimes a neural network-based approach can be more robust thus also requiring less image pre-processing which can also be a contributing factor to inefficiency. Across many of the approaches, neural networked or not the systems aimed for a testing framerate of at minimum 30 frames per second with minimal inference time due to the real-time processing requirement of a sign detection system, it was also noted that the processing requirement should be low. Many of the proposals aimed to be capable of running on a laptop, some of which can also be run on a mobile phone. A two-stage classifier would be the easiest method for achieving high accuracy whilst facilitating modularity.

5 METHODOLOGY

5.1 SOFTWARE DEVELOPMENT PROCESS

5.1.1 Software Development Methodology

The purpose of a software development methodology is to provide scaffolding for a development team to follow which enables them to work more efficiently as a team. Due to the wide range of software products developed several frameworks and concepts have been developed which suit different applications. One of the most inflexible but robust models is the waterfall model. This model has 5 stages, Requirements and Planning, Design, Implementation, Testing and Maintenance. Each stage must be completed sequentially. Any changes to previous stages must be completed during the next cycle increasing the dependency on good requirements engineering and design. This is compounded by comments made by Boehm (1988, p.63) where he states 'it does not work well for many classes of software, particularly interactive end-user applications'. For these reasons, this type of methodology should be avoided.

Agile methodologies are designed to enable more flexible development structures whilst still encouraging good software development practices. There are many different agile development methodologies however they all aim to abide by the 12 principles as outlined in the agile manifesto as can be seen within the appendix under section 13.3

Within the agile umbrella, there are a range of different methodologies such as scrum, rapid application development [RAD] and extreme programming. Scrum focuses on subdividing the development process into smaller more manageable chunks, called sprints, these are time-specific chunks which usually last between 2 weeks and a month and contain a series of goals. Each sprint contains all the main software development processes such as planning design, implementation and testing however due to its dynamic nature changes to the specification can be quickly incorporated during the next sprint. Any goals which are not met during the sprint are placed on the 'backlog' and are planned for subsequent sprints. A similar approach to this is Kanban, instead of set sprints, requirements are split into individual tasks and then assigned, as tasks progress they are moved across the board into different swim lanes allowing for efficient progress tracking.

One of the quickest but least efficient methods of software development is RAD, once the system requirements and scope have been set the development team begins the construction phase, during which the client repeatedly interacts with the prototype system giving feedback which is incorporated mid-development with continuous testing. As stated by Chien (2020), the main requirement for this type of development is having a skilled development team as they are required to engineer the solution on the fly. Without a skilled team driving the project unforeseen issues can arise and slow development at which point a different SDLC would have better suited the project.

Due to the time-sensitive nature, high degree of complexity and unknown nature of the project being undertaken, RAD is not suitable. Fixed methodologies such as the waterfall model have some benefits however when strictly sticking to this style of development it can cause issues when adjustments in the specification and scope are required. The implementation of this project will use a scrum-based methodology as this gives the most flexibility during development whilst allowing the project to be structured in a way which allows for efficient resource and time planning. Due to the short timeframe of the project sprints will last 2 weeks.

5.2 Requirements Engineering

The requirements for this project should come from a range of sources to provide a full picture of the needs placed upon this application. As is stated in the Software Engineering Book of Knowledge v3.0 (Bourque & Fairley, 2023, p. 36-37) [SWEBOK] requirements can come from a range of sources such as the application objective, domain knowledge, potential system stakeholder requirements and the operational environment. Other sources of requirements should not be considered such as business rules as this application does not aim to solve a business process or problem. Subsequently, requirements posed by the organisational environment will also not be considered due to the experimental nature of this application. It should be made clear that the operational environment will outline many requirements due to the implementation of the system in a locomotive there will be additional real-time processing and safety requirements to adhere to. These should be actively considered as contravening these would negatively impact the suitability and usability of the product.

5.2.1 Requirements Elicitation

Requirements elicitation techniques have been developed to assist a developer with extracting requirements from a client or brief.

As noted in SWEBOK there are several different techniques including Interviews, Scenarios, Prototypes, Meetings, Observation, User Stories and Data Mining. Some of these strategies are more appropriate than others, for example, data mining or analysing competitors will not be useful for this application as this is one of the first of its kind. As can be seen below in Table 1, each of the requirements elicitation techniques have been listed and their usage in this project has been detailed.

TECHNIQUE NAME	DESCRIPTION	USAGE ON THE PROJECT
INTERVIEWS	Traditional meetings with the stakeholder to establish the requirements and scope of the project, led by the developer with a series of questions.	Not useable as the potential stakeholder cannot be interacted with.
SCENARIOS	Applies in use cases and describes how an action is done in context.	Can be used to extract how railway procedure works and should interact with the system.
PROTOTYPES	Allows the developer to map out requirements to a physical design.	This can be used to assist in ensuring that all interface requirements are met.
MEETINGS	Two-way discussion with clients to gather requirements and scope.	Not possible as the potential stakeholder cannot be interacted with.

OBSERVATION	Having the developer observe how a task is performed, helps build domain knowledge.	Observing real-life train drivers, to see how speed is handled. This is not something which would be possible.
USER STORIES	States how and what a user should be able to do with an application in plain text, this gives a high-level view of the interactions required.	State what/how a user should interact with the system and how the system should behave in accordance.

Table 1 –Requirements Elicitation Techniques and their Application within Requirements Engineering for the System

As can be seen in Table 1, there are several applicable elicitation techniques which would be of use to this project including prototyping and user stories and scenarios. These will be used within the requirements specification to state how the product should operate.

5.2.2 Requirements Specification

A formalised requirements specification is the grounds for any software engineering process as this is the basis for any models or design documentation which is generated. Once finalised it can be seen as an agreement between the developer and the stakeholders of the system listing what features the finalised product will contain. One of the most common methods is a requirements list. this enables a developer to adapt notes gathered during elicitation into a concrete requirement list, of functionality and features.

User stories will also be employed and can be used to validate the functionality of a given requirement; these describe the actions which the user wishes to perform.

A template for capturing requirements can be seen below in Table 2. As can be seen, the table also includes the facility to rate the priority of a given requirement, this is useful for capacity and resource planning and gives any personnel involved in the project the ability to quickly look at the requirements specification and understand the importance of any given feature.

	REQUIREMENT	**USER STORY**	**PRIORITY**
1			

Table 2 – Functional Requirements List Template.

Specifications can also contain a differing amount of detail; high-level specifications focus on user interaction whereas low-level specifications allocate more focus to the backend workings of an application. Given the requirement of the application for a processing-heavy implementation, a low-level specification will be employed to be sure that all aspects of this project have been mapped out appropriately.

This allows a developer to adapt notes gathered during requirements elicitation into a concrete list of features and functionality. It is also important when developing a requirements list that software qualities should also be considered, there are a range of qualities all software should have such as

maintainability and security. A limit should be placed on the number of required qualities as too many can be counter intuitive, complicating the design and development process. This application should have three key qualities: safety, maintainability and performance efficiency.

5.2.3 Requirements Modelling

Unified Modelling Language [UML] is a modelling language consisting of 14 standardised diagrams, these diagrams are further split into behavioural and structural diagrams. Behavioural diagrams state how an operation should be actioned and how the software should react to a set of inputs. These diagrams can be used as a communication tool between the client and developer to ensure that all the requirements and business restrictions are being met. Structural diagrams assist in outlining the structure of the program and how different modules are layered within the application. Which further enables the development of maintainable and well-thought-out code. UML will be used to assist in converting a series of text-based design requirements into physical designs which in turn will assist with developing a better structured product.

5.2.3.1 Use Case Diagrams & Use Case Specifications

The requirements of this project will be converted into a series of use cases, this will allow documentation of functionality. This will also be extended into use case specifications. This provides the developer with a more granular level of detail on the process flow behind a given use case. An example of the use case specification which we will be using to model the system can be seen below in Table 3.

Section	
Designation	
Name	
Criticality	
Description	
Actors	
Scalability	
Path	
Related Use Cases to the path	
Alternatives	
Related use cases to the alternatives	
Exceptions	
Use Cases related to the exceptions	

Table 3 – Use case specification template. Based on the layout provided by Allen (n.d) on Brightspace.

5.2.3.2 Requirements Diagram (SysML)

Systems Modelling Language [SysML] is an extension of UML, the intention behind this was to develop a framework of diagrams which complimented UML but also added the ability to illustrate a wider range of systems including integrating hardware, software, information, processes, personnel, and facilities (Lucidchart, n.d) into the diagrams which isn't something which is natively supported by UML. SysML was designed to be a language specifically for modelling software and digital systems. SysML contains 9 different diagrams, some of which are beneficial to the modelling of this system such as the requirements diagram. This will be used to map out the system requirements in a way which can be easily visualised this will also enable linking of the requirements to additional non-functional requirements and software qualities as have been previously discussed.

5.3 SOFTWARE DESIGN

5.3.1 System Modelling

5.3.1.1 Activity Diagrams & Sequence Diagrams

These two models will be used extensively during the design of the software, this allows a developer to map out how a given process should flow and any decision logic which is required. Activity diagrams give a more simplistic flow-chart like view to process flow which will be used to plan the high-level logic required for the application. Sequence diagrams are similar however use exact method calls and commonly reference planned classes. These diagrams have been selected as they allow process flows to be mapped and documented before development commences. This also allows any potential issues with the application structure to be fixed. This especially helps with maintainability. Ensuring that modules have low cyclomatic complexity through mapping decision logic.

5.3.1.2 Class Diagrams

As previously discussed, sequence diagrams model specific method calls through different objects, the class diagram will be used to enable this by modelling each component within the application. This also facilitates hierarchical structuring and abstraction to be employed during the design phase.

5.3.1.3 Package Diagrams

This type of diagram will be employed to map out the application structure, this is key to ensure that it is maintainable and assists with visualising external dependencies.

5.3.2 Wireframing

Wireframing can be a key asset to a developer, this encourages construction of non-functional prototype interfaces as a communication tool. This facilitates communication with clients about the operation and functionality which the interface will have. This also enables the designer and developer to ensure that the functionality listed within the specification has been mapped to the interface. There are primarily three different types of wireframes, low, medium, and high fidelity. A breakdown of each type can be seen below in Table 4.

WIREFRAME FIDELITY	DESCRIPTION
LOW	Simple layout designs with minimal styling e.g., basic whiteboard sketches. As stated by MasterClass (2021), these types of wireframes use placeholder text and exclude interactive options.
MEDIUM	This provides a middle-ground design, having replaced the placeholder text with design text however styling and typography elements are still excluded.

HIGH	This is close to the full interface design; this will map out exactly how the interface will look without implementing any of the functionality.

Table 4 – Types of wireframes.

For this project a medium-fidelity wireframe will be most appropriate due to the nature of the application. Each element within the wireframe will be linked back to the requirement through annotations.

5.3.3 System Architecture

As outlined by Bass et al (2021, p. 22) “The software architecture of a system is the set of structures needed to reason about the system. These structures comprise software elements, relations among them and properties of both.” This defines how the underlying system will be structured and how the classes withing the application should pass data and interact with the data sources. There are three main types of system architecture modelling: component-and-connector structures, module structures and allocation structures. Based on the descriptions given by Bass et al (2021, p. 27-30) Table 5 has been created to summarise the several types of structure diagrams.

STRUCTURE TYPE	DESCRIPTION
COMPONENT & CONNECTOR	This structure primarily focuses on how different components within the system will interact and interconnect with each other to perform the system's function. These components can be abstracted to represent various elements such as services, clients, or servers.
MODULE	This splits the application into single implementation units, this specifically focuses on how the code of the application will be implemented. This ensures that all objects within the system conform to single responsibility principles [SRP].
ALLOCATION	These structures more closely map software structures to other non-software-based structures such as the execution environment.

Table 5 – The three different types of software structure diagrams.

Based on the information summarised in Table 5, the application should follow a module structure as this is the most applicable to the type of software which we are developing. There are multiple ways of expressing a module structure within a diagram the most applicable being a decomposition

diagram to demonstrate how each component will interact. As has already been discussed this will be expressed using a UML package diagram.

The GUI should also conform to the same 'module' structure. A lightweight view controller architecture is the most suitable and ensures that the views and their respective controllers are separated thus ensuring maintainability and SRP.

To optimize the processing speed, this application will make use of multithreading, by default Python applications run on a single core. The use of threading will enable the system to utilize different CPU cores for different purposes during runtime, the interface, control system and detection system will all operate on different threads to ensure that maximum performance is achieved.

5.3.4 Design Patterns

Design patterns are a predefined set of programming constructs, the concept was first proposed by Gamma et al (1994) in a book titled 'Design Patterns - Elements of Reusable Object-Oriented Software', it proposed a series of 23 unique design patterns, split into three distinct categories. Creational patterns, structural patterns and behavioural patterns based on their function. These patterns have been designed to help developers better structure their applications. Within this project we will make use of the adapter pattern, this will enable us to develop a system designed to enable interchangeability. This is most useful for allowing the application to connect with a wide range of different locomotive interfaces, although during testing we will only be using a simulator enables us to design a system which could theoretically connect to a locomotive with minimal re-engineering if required.

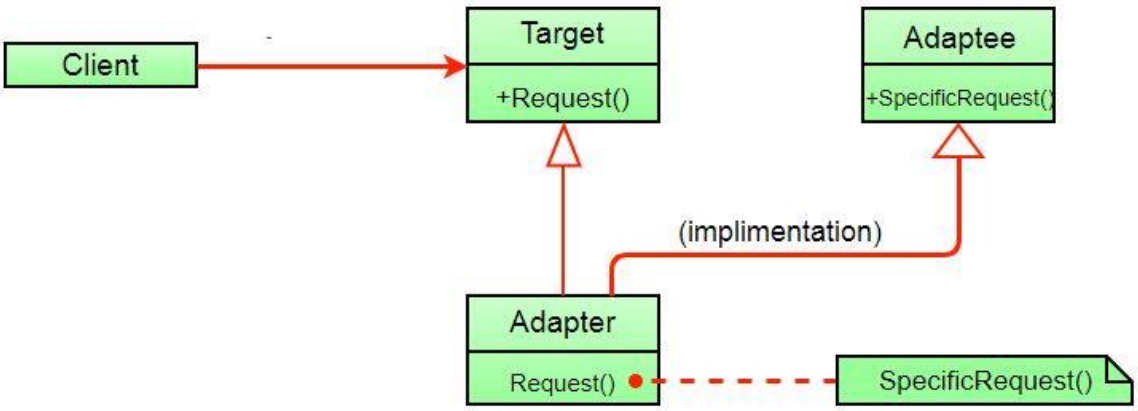

Figure 3 – Example of the adapter pattern (GeeksforGeeks, 2023).

The observer pattern will also be used during the detection cycle to inform different components within the application such as the GUI to update certain information, this could extend to changes in speed or control inputs. This allows us to notify specific parts of the application efficiently.

5.4 SOFTWARE CONSTRUCTION

5.4.1 Construction Tools

5.4.1.1 Language and Libraries

For data science and machine learning applications, Python is one of the most popular languages, this is due to its sequential nature and its simplicity. The alternative is C++, this is a more difficult language to learn. Due to the short timeframe of this project and our existing knowledge of python, this will be what we will develop the model and the application using.

This project will make use of several off-the-shelf libraries such as the machine learning framework as will be discussed below, the primary reason for this is that the code which is provided by these libraries already implements the functionality which we are seeking therefore it would not make sense to re-implement this ourselves.

5.4.1.1.1 Object Detection System

A library must be selected which can train a YOLO type model, the two most popular and customisable, both of which are built in Python:

- Ultralytics (YOLOv8)
- ImageAI (YOLOv3)

This project will be based on the infrastructure provided by Ultralytics as this offers a wide range of YOLOv8 models varying in size and complexity. It is also a widely used package having over 2,000 stars on GitHub. The YOLOv8 architecture is the most recent version of the architecture which builds upon previous versions which further reduces inference time and increases accuracy compared to previous versions. As stated in the documentation (Ultralytics, n.d) latency is reduced by over 1 ms when compared to a YOLOv5 model. The alternative, ImageAI is only capable of training a YOLOv3 model, which loses the additional performance enhancements built into later versions of the architecture.

5.4.1.1.2 Character Identification System

Once the location of signs has been identified within a video frame, the values must then be extracted. This is the sole responsibility of the recognition subnet which will make use of optical character recognition [OCR] to identify the signs' value. Similarly, to the detection subnet, any application used must have a low inference time and processing overhead to ensure that signs are being read quickly and correctly. The most popular open-source library being PyTesseract (madmaze, 2022), based on ‘Tesseract OCR’.

5.4.1.1.3 Graphical User Interface

Another key aspect of this project will be the graphical user interface [GUI], this will allow the driver of the locomotive to interact with the system. After research, there are three popular and well-supported GUI libraries for Python, TKinter (Python Software Foundation, n.d), PyQT5 (Riverbank Computing, n.d) and Kivy (Kivy, n.d). Tkinter is packaged with Python. The main limitation of this library is that it does not have much flexibility concerning styling and element positioning. PyQT is a wrapper around the existing QT library, which has a set of published design tools which enables drag-and-drop interface design. The alternative is Kivy, which makes use of a structured element file, this allows the developer to quickly define what elements should be within the interface and the

layout constraints imposed on them. Primarily this allows for rapid interface development an even quicker layout changes during development. This separation of controller and layout code affords an additional degree of separation between controller and view code enhancing the maintainability of the system. It should also be noted that Kivy is designed to be used for mobile development thus meaning that the background library is extremely lightweight, both in size and complexity. Based on some experimentation and the above reasons this project will make use of Kivy for its increased flexibility over PyQT or Tkinter.

5.4.1.2 Version Control

For any code-based project, version control is key. This allows a developer to better organise code, view past and present changes and manage what code is incorporated into the project by different developers. There are a range of different VCS solutions on the market such as Git, SVN or CVS. According to research by Slintel (2023), Git has the largest market share at 89.08% with over 98,000 current customers. SVN having only a 0.20% market share with CVS only having a 0.07% market share. Git also provides the added benefit of being able to branch the source code, allowing multiple changes to be made concurrently without affecting the main source code. Due to this project's use of scrum, this will be key as during a sprint it will be required that each goal has its own branch with the changes being merged at the end. There are two distinct options for hosting Git repositories, the first being GitHub and the second being GitLab. As mentioned by Vats (2022) there is a philosophical difference between the two hosts, with GitLab offering a more feature-rich service whereas GitHub offers increased efficiency and uptime. As we do not need these additional features and GitLab offers facilities for CI integration this project will make use of GitHub.

5.5 SOFTWARE TESTING

Testing is a key aspect to any project; this ensures that the application delivered has minimal bugs and conforms to the clients' requirements.

Different test techniques can be employed when developing software and applications, these can range from in-person acceptance tests where the client physically interacts with the application to automated tests which are used to ensure that the base-level application code is operating as expected and conforms to the requirements. Outlined below are a range of different commonly employed test techniques, a brief description, and their applicability to the development of this software.

TECHNIQUE NAME	DESCRIPTION	APPLICABILITY AND JUSTIFICATION
UNIT TESTING	This comprises of a range of different test cases and is designed to test individual methods within a class to ensure that their output meets the expected value. This is commonly used to test edge cases and error handling within an application.	This is applicable for use to test the system as it enables the developer to create tests against the pre-defined test plans and run these as required when changes are made to ensure that process logic has not been affected.
INTEGRATION TESTING		This is also applicable to the application as test cases can

	This is like unit testing and can be automated. This tests a wider range of functionality within the application and tests multiple modules at the same time rather than focusing on single methods.	be written to test the application end to end.
ACCEPTANCE TESTING	This type of testing focuses on the client testing the application for the developer using real-life scenarios.	This is not possible as there is not a physical client involved in the development of the application however this can be substituted for system testing, as this will test against the initial requirements.

Table 6 – Different test techniques.

5.5.1 Test Process

As has already been discussed the application will make use of unit/integrations tests and end-to-end tests to ensure that the software developed is robust and meets the initial requirements. Each requirement within the specification will be linked back to a test plan as will be discussed later in this section.

Unit tests will be used for functionality where possible; it is unavoidable that some of the functionality of the application can only be tested manually due to the nature of its implementation using computer vision. Ideally, all classes which are being tested should have as close to full test coverage as possible.

Manual system testing will be used to cross-validate the requirements with the functionality of the application. All requirements captured within the specification and referenced on the requirements diagram should be linked back to test cases and recorded accordingly. The format of the testing table has been outlined below in Table 7. The associated test plan will also be recorded using Table 8.

REQUIREMENT	DESCRIPTION	TEST PLAN	PASS/FAIL	EVIDENCE	NOTES

Table 7 – Testing Table.

TEST PLAN NUMBER	
ASSOCIATED REQUIREMENTS	
TESTING STEPS	

Table 8 – Test Plan Template.

5.5.2 Testing Tools

As previously discussed, this application will employ unit testing, Python is packaged with an automated testing framework, 'unittest'. As stated on the documentation it was inspired by JUnit (Python Software Foundation, 2023). Due to is similarity with other testing frameworks, we will be employing this to test the application, this similarity will ensure that any new developer picking up the project can quickly familiarise themselves with the tests and write new tests.

5.5.2.1 Simulator Selection

As testing on a real railway would be significantly cost-prohibitive this project elects to test the application within a simulator. This allows us to emulate a range of scenarios for a fraction of the cost. There are three main simulators (Train Simulator Classic, Open Rails and Trainz Railroad Simulator) which we have evaluated for testing this has been outlined below.

Train Simulator Classic, developed and published by Dovetail Games (2009), has a feature-rich and easy-to-understand route builder. This will allow the development of the custom test track used to test this application. The route builder also enables us to modify existing routes within the game thus allowing any packaged route to be customised and used as a test routes. This may be beneficial for testing as it will allow us to make use of pre-existing and modelled routes with extensive clutter. Another notable point is that the developers of this simulator have extensively modelled UK railways therefore there is a range of pre-built UK speed sign assets to choose from. Finally, a dynamic linking library [DLL] has been bundled with the application which enables external hardware or software to interact with the application.

Open Rails (n.d) was another candidate, it is an open-source application based on an older simulator published in 2001, Microsoft Train Simulator [MSTS], as a result, there is extensive user-generated content available allowing it to be extremely flexible when testing. The main downside of this application was the lack of a dedicated route builder. A 3rd party tool has been published however the complexity and instability of the program makes its use prohibitive.

Trainz Railroad Simulator 22 is a rail simulator published by NV3 Games (2022). It has many of the features of its contemporaries including a route builder and a range of environments and content, including accurate modelling of UK railway signs. However, it has one significant downside, its lack of an external control interface which is key for our implementation as it allows the application to interact with the simulator in a comparable way it would to a locomotive.

Due to its utility and flexibility as a testing environment, we will be using Train Simulator Classic to test our application, it encapsulates the 3 key requirements for a testing environment: accessible and functional route builder, control interface and modelled UK rail signs thus allowing us to repeat end-to-end tests accurately and quickly.

5.6 Professional, Legal, Social and Ethical Issues

5.6.1 Professional Issues

The key professional issue which any developer faces is ensuring that any code published is of excellent quality, maintainable and robust. This is essential when working with a system which relies on accuracy for operation, without these core principles any system can be rendered unstable. Without robust code, a system is likely to be more inaccurate due to its ability to make more

mistakes. Some of this can be remedied by ensuring that any code is thoroughly tested, this reduces the number of bugs which make it into the final code. All edge cases must be considered and tested. The implementation of testing frameworks makes it significantly quicker for a developer to test their code and ensures that any changes did not affect the operation of the system (regression testing). This is key when developing software which is capable of automatically controlling a vehicle as any issues or bugs could cause significantly costly incidents which in many cases could also have significantly legal implications. Another professional and partially ethical consideration to make is the use of open-source software, the intention is that the code is made available by its authors for use by other parties. As part of these software agreements, it is key that the authors of the original code are acknowledged. Depending on the type of open-source agreement it is also sometimes noted that the code used cannot be incorporated into a system which is used to generate profit or used to develop a closed-source system, one such example as highlighted by Losio. (2022) in an article on InfoQ speaks of a class action lawsuit against GitHub's Copilot program which is an AI-powered assistant used to write code. It was trained using open-source code written by developers but has been absorbed into a closed-source system. The lawsuit argues that this is in breach of the open-source licenses which the code was written under.

5.6.2 Legal Issues

Autonomous vehicles face a range of legal issues in their implementation, the primary being who is legally responsible for the vehicle in the case of an incident. As is noted in an article by Enable Law (2022), "The Law Commission agrees with the current position that passengers in a vehicle being operated by a fully Automated Driving System [ADS], should not be legally responsible for its actions". This also currently encompasses a legal grey area as to what defines an ADS, one of the leaders in the field of autonomous vehicles, Tesla where their support page heavily emphasises that its autopilot is not an example of an ADS (Tesla, n.d). This issue is also carried over to a system which is designed to identify speed signs and control a vehicle's speed, if the system does not provide accurate identification and causes an over-speed event without driver intervention, this could potentially result in legal action being taken against the system due to the consequences of this action.

5.6.3 Ethical Issues

One of the most important aspects of any project or piece of research is ensuring that it is carried out ethically. This is twofold, firstly public opinion is likely to judge strongly against any research if it is deemed to have been conducted in an unethical manner, this can potentially devalue the findings. One such example was highlighted in an article by Regan et al. (2018) on CNN Health which outlines the findings of a Chinese doctor, He Jiankui who developed a medical procedure capable of modifying the DNA of embryos before birth to make them resistant to HIV. As a result of this unethical application of medical science, he was forced to stop research, which later cost him his job, facing 3 years imprisonment and a $500,000 fine (Regalado, 2019). There are some significant ethical issues faced by companies looking to research and develop commercial autonomous vehicle technology. One of the most prominent moral and ethical issues as proposed by Thomson (1967, p.206), the trolley problem, this involves a trolley whose brakes have just failed, there are five people on the track ahead, there is also an alternative path which would take the trolley to the right however there is a further person. Either the trolley can hit the five workers or one. This moral dilemma has been debated extensively with many having a range of different viewpoints on the issue, this can also be translated into an autonomous system, should the vehicle be able to make this decision? The same can be noted for a system which is designed to identify speed signs, although it

may never directly affect the end user, sometimes a sign may be miss identified. It is key that the system is robust enough to acknowledge this, and display that the speed sign was not understood rather than guessing and hoping that it was correct as this could cause significantly more damage. One other ethical issue to remain conscious of is the security of any potential system developed. An insecure autonomous vehicle is inherently unsafe. This insecurity could lead to a malicious actor gaining access to the system, given that the system will be developed to increase safety by reading speed signs this would give the intruder the ability to control a vehicles speed. It is important that any system which is developed should have the base understanding that it should be developed securely, this includes ensuring that any libraries used should be tested and widely used this reduces the likelihood of a bug resulting in a security breach. This also means that the system should have the ability to physically disable it if it begins to malfunction or an intruder gains access. This application will be entirely offline to mitigate this risk.

5.6.4 Social Issues

One of the key issues and barriers to autonomous vehicles is public perception. This is partially due to a lack of trust in these systems. A Eurobarometer survey by data.europa.eu. (2020) questioned over 27,000 participants to gather data on their trust in autonomous vehicles. One of the most notable responses was to the question “How comfortable would you feel travelling in a fully autonomous vehicle without a human operator”, 50% of the given responses answered, “not comfortable at all”. This gives good evidence to the basis of public mistrust given that if a human operator were in the vehicle only 14% of people would feel uncomfortable. This is 12 something which will only change over time as additional research is performed into improving the accuracy and decision-making process of autonomous systems.

5.7 TIMELINE

As can be seen below in Figure 4 – Development Time Plan, a basic estimate of the duration to complete each aspect of the project has been established. This started from the 3rd week of September spanning through to the final hand-in date of the project during the first week of May. Top-level tasks such as the development of the system have been given additional consolidation time at the end of the planned phase, this allows us to have an additional week’s leeway if anything unexpected occurs during the phase which delays the completion of the task.

Figure 4 – Development Time Plan.

6 REQUIREMENTS

6.1 SYSTEM SPECIFICATION

The Autonomous Locomotive Control System [ALCS] should be a system capable of detecting and identifying railway speed signs in real-time. This should be capable of taking input from frames captured directly from a computer monitor or captured from a webcam. Once the speed stated on the sign has been identified it should then be capable of increasing or decreasing the speed of the locomotive following a pre-defined braking or acceleration curve. When a new speed limit is detected, the system should play an alert noise to notify the driver of this change.

The system should present the user with an option to start or stop detection as required along with a comprehensive user interface which displays the current speed of the locomotive, current line speed, detected speed limit when a new detection is made, an image of the detected speed sign.

6.2 FUNCTIONAL REQUIREMENTS

6.2.1 Main Application

	REQUIREMENT	USER STORY	PRIORITY
1	Adjust System Settings (5.2.2 Settings Menu)	"I want to be able to have an area of the system I can use to	Required

		adjust different settings of the application".	
2	Detect Railway Speed Signs	"I want the system to be able to detect railway speed signs from video frames".	Required
3	Start Detection System	"I want the ability to start the detection system when required".	Required
4	Stop Detection System	"I want the ability to stop the detection system when required".	Required
5	Acknowledge Change in Detected Speed Limit	"Due to safety requirements, I wish to approve all detected changes in speed before the system acts upon them".	Required
6	View the Current Line Limit	"I wish for a location to view the currently	Required

		detected line speed being fed into the control system".	
7	View my Current Speed	"I want to be able to view the current speed of the locomotive to ensure that the control system is correctly following the line speed".	Required
8	Live Preview the Detection System's View	"I wish to be able to see what the detection system is seeing, when a sign is detected, I wish to see the bounding box and confidence level of the system".	Required
9	Control Locomotive Acceleration	"I wish for the system to be able to accelerate when the line speed increases, the acceleration of the locomotive must be smooth and controlled".	Required

10	Control Locomotive Breaking	“The system should be able to slow the locomotive when the line speed decreases or if the locomotive increases beyond the currently approved line speed. The deceleration should be gradual and controlled”.	Required
11	View the Control Inputs of the Control System	“As part of the safety of the system I want a facility to be able to view the control inputs of the system made to the locomotive”.	Required
12	Emergency Stop	“As part of the safety requirements of the system, the driver should have the facility to enact an emergency stop from the systems interface”.	Required

6.2.2 Settings Menu

	REQUIREMENT	USER STORY	PRIORITY
1	Close Settings Menu	"If I do not need to save any settings, I wish to be able to close the settings menu".	Required
2	Adjust capture Width and Height	"The system should be able to be adapted to any locomotive or camera position therefore I want to be able to adjust what regions within the cameras field of view will be processed by the system".	Required
3	Select the Input Source	"The system should be able to be adapted to any locomotive or device therefore I want to be able to switch between capturing video from a computer screen for evaluation and testing to a physical camera for future real-life employment"	Required

4	Preview the Capture	"I wish to be able to preview what affects the changes I have made to the settings before saving".	Required
5	Set the Path to the Control DLL	"The system should be able to connect to a simulator for testing".	Required
6	Save Settings	"As a user, I want to be able to click the save button and have the settings be stored in a form of persistent storage".	Required

6.3 SOFTWARE QUALITIES

6.3.1 Safety

To ensure that the system is safe there should also be a provision made that in unexpected circumstances the driver/operator can enact an emergency stop, when the button is pressed, the system will stop detecting speed signs and a full brake application shall be made. Once the locomotive has stopped, detection can be resumed if required.

A passcode should also have to be entered before the user is given access to the settings menu, there does not need to be a facility to change this passcode as this will be handled by the maintenance team in the depot if required and will be set when the system is installed on a locomotive.

6.3.2 Maintainability

- The system should conform to PEP8 coding standards.
- System methods should be appropriately commented.
- Should have a cyclomatic complexity below 20.

- Should have a maintainability index above 70 – for more information please see, 13.4. This will be tested using Radon (rubik, n.d)
- Should have a combined feature and unit test coverage of at least 80%

6.3.3 Performance Efficiency

- The system should be able to capture, detect and identify speed signs once every 200ms, which equates to approximately 5 FPS.

7 SOFTWARE DESIGN

7.1 REQUIREMENTS MODELLING

7.1.1 Requirements Diagram

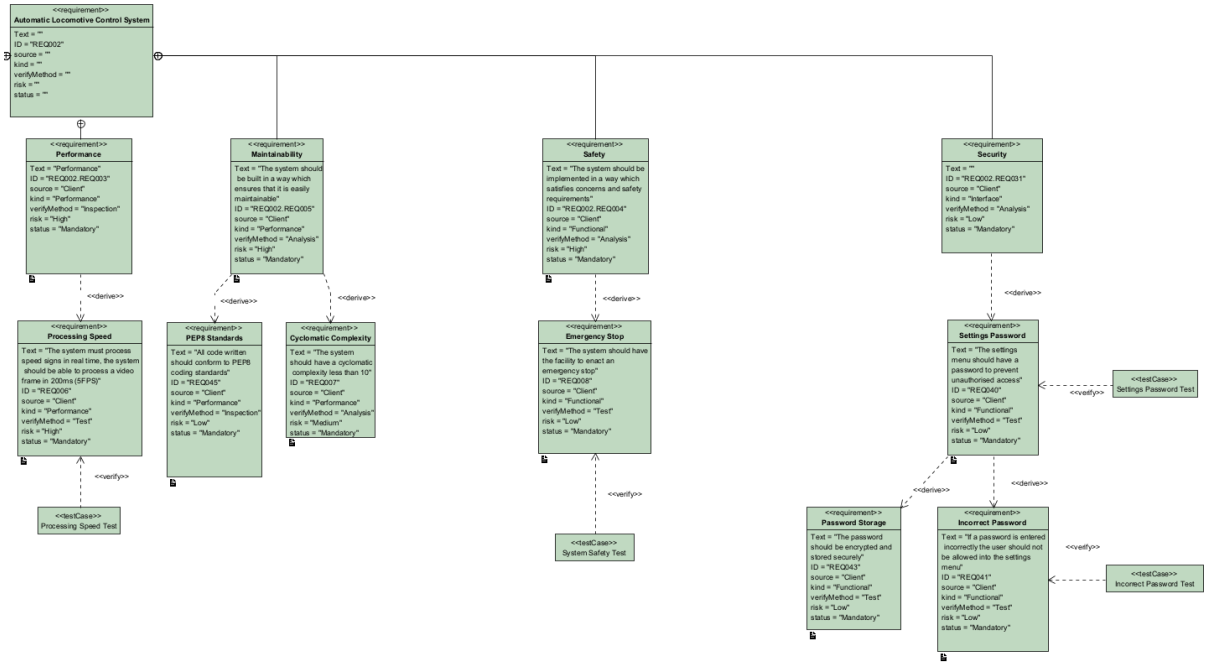

Figure 6 – Automatic Locomotive Control System Requirements 1.

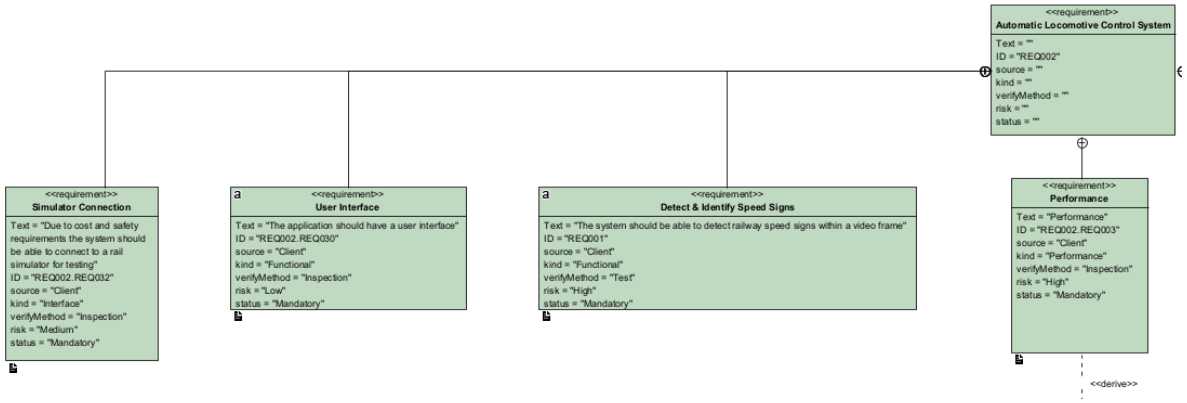

Figure 5 – Automatic Locomotive Control System Requirements 2.

Please note, for readability this diagram has been split into two separate diagrams.

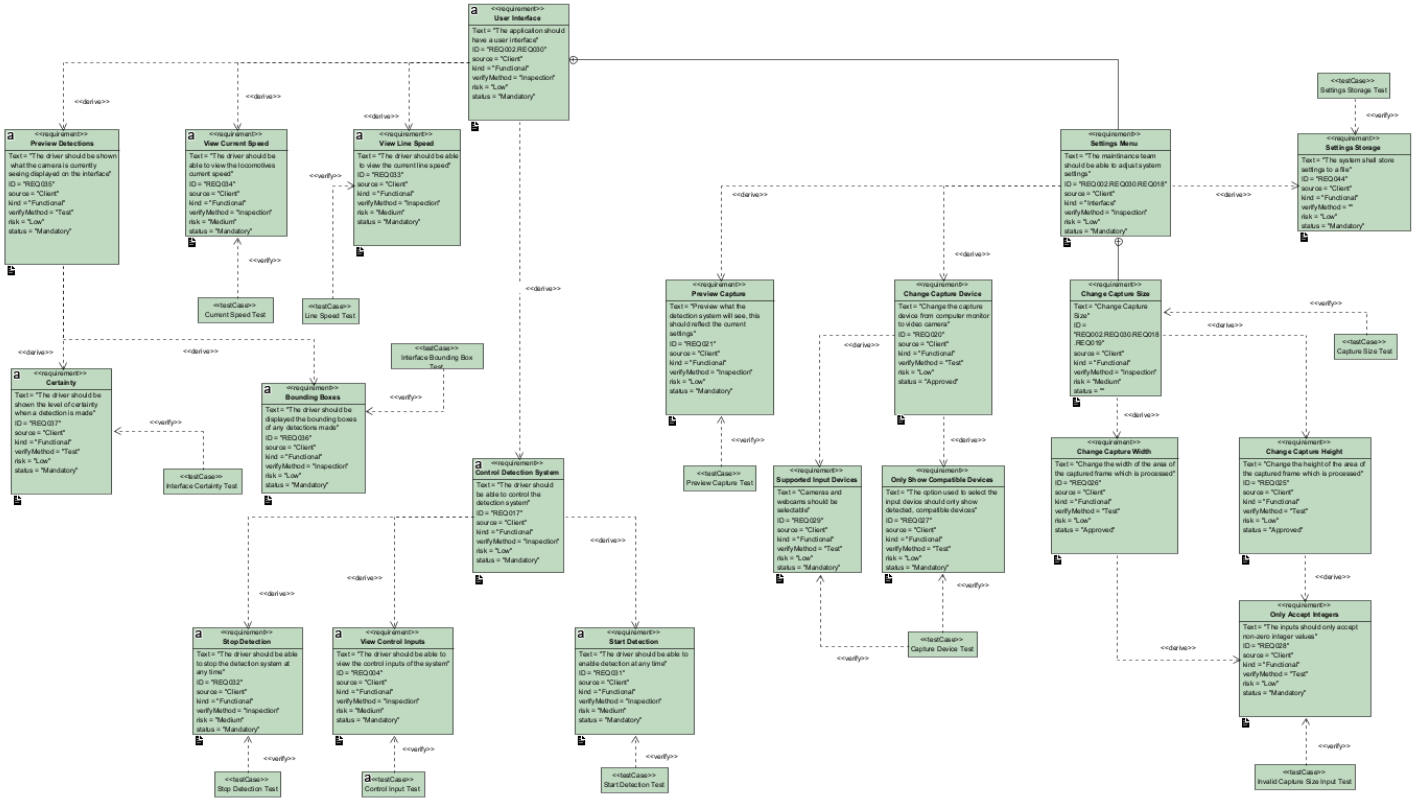

Figure 7 – User Interface Requirements Diagram.

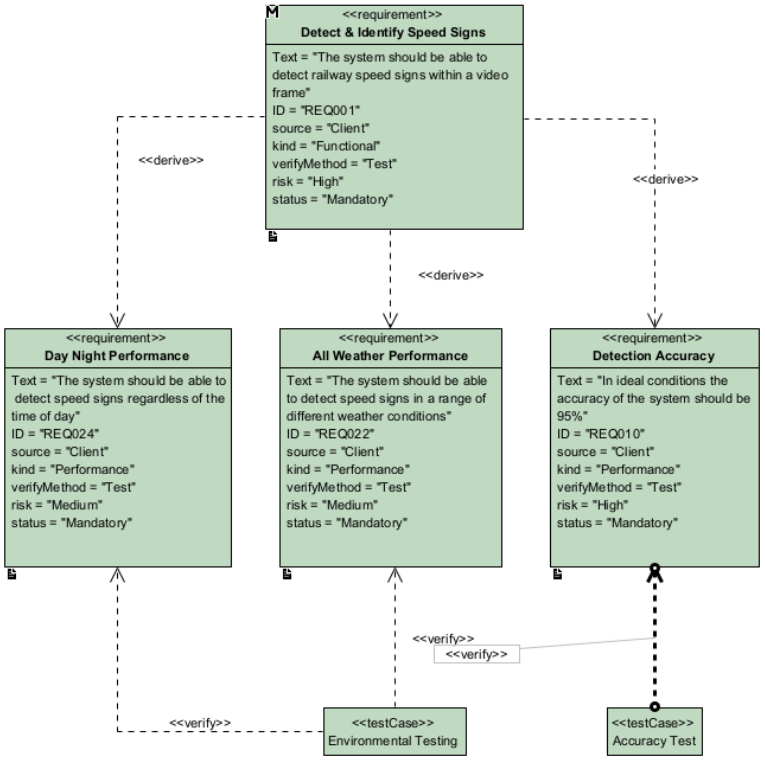

Figure 8 – Sign Detection & Identification Requirements Diagram.

7.1.2 Use Case Diagram

Based on the requirements listed within 6.1, a use case diagram has been generated to cover as many use cases of the application as possible, for simplicity maintainers have been generalised as a type of driver. In the full implementation, drivers would not have access to the settings menu as maintainers would be the only users with the passcode however there would be no end-point separation therefore a driver would be able to access the same use cases a maintainer would.

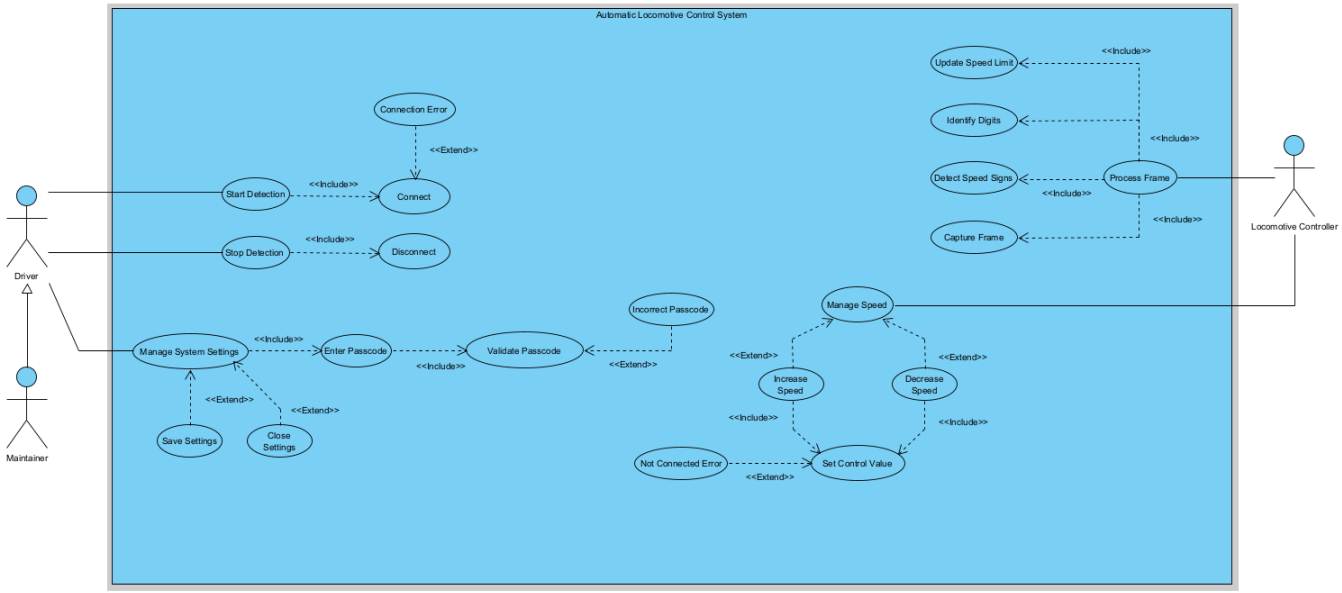

Figure 9 – Use Case Diagram.

7.1.2.1 Use Case Specifications

Section	
Designation	UC001
Name	Manage System Settings
Criticality	High – Without this a maintainer or driver will be unable to view system settings
Description	This allows the driver or maintainer to open a menu which will allow them to view system parameters
Actors	Driver / Maintainers
Scalability	Single Instance
Path	1) The user clicks the settings menu button. 2) The application loads the passcode window.
Related Use Cases to the path	2 – Enter the passcode
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC002
Name	Save Settings
Criticality	High – Without this a maintainer or driver will be unable to adjust or save system settings
Description	This allows the user to save any changes which they have made to system settings to a file.
Actors	Driver / Maintainers
Scalability	Single Instance
Path	1) The user clicks the save settings button. 2) Application iterates through all fields within the settings menu and converts them into a key, value dictionary. 3) The dictionary is then serialised into a JSON format and saved to the internal settings file. 4) Upon completion, the settings menu is hidden from the user.
Related Use Cases to the path	None
Alternatives	None
Related use cases to the alternatives	None
Exceptions	3.1 - Settings file not found / cannot be written to
Use Cases related to the exceptions	Settings File Not Found / Settings Writer Exception

Section	
Designation	UC003
Name	Close Settings
Criticality	Low – If this fails there will be no outcome other than the application requiring a restart to exit the settings menu
Description	This enables the user to close the setting menu.
Actors	Driver / Maintainers
Scalability	Single Instance
Path	1) The user clicks the close settings button. 2) The settings menu is hidden from the user's view.
Related Use Cases to the path	None
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC004
Name	Enter Passcode
Criticality	Medium
Description	Enter the passcode to access the settings menu
Actors	Driver
Scalability	Single Instance
Path	1) Enter the passcode. 2) Submit the passcode. 3) Validate the passcode. 4) Show the settings menu.
Related Use Cases to the path	3 – Validate Passcode
Alternatives	4.1 – Invalid Passcode, Show an error message.
Related use cases to the alternatives	4.1 – Incorrect Passcode
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC005
Name	Validate Passcode
Criticality	High
Description	Validate the entered passcode to see if it is the same as the stored passcode.
Actors	Driver
Scalability	Single Instance
Path	1) Load the passcode from the settings file. 2) Compare the two values.

	3) Return True.
Related Use Cases to the path	None
Alternatives	2.1 – Passcode does not match, Invalid Passcode
Related use cases to the alternatives	2.1 – Invalid Passcode
Exceptions	None
Use Cases related to the exceptions	None

Section

Designation	UC006
Name	Incorrect Passcode
Criticality	Low
Description	The incorrect passcode was entered
Actors	Driver
Scalability	Single Instance
Path	1) Show the error message to the user. 2) Return to the enter password screen.
Related Use Cases to the path	None
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section

Designation	UC007
Name	Start Detection
Criticality	High – If this fails the application will not be able to function as specified within the requirements
Description	This enables the user to initiate the detection system.
Actors	Driver / Maintainers
Scalability	Single Instance
Path	1) The user starts the detection system. 2) The detection state is set to true. 3) The start detection button is disabled, and the stop detection button is enabled. 4) Initialise the loop which calls the detection and control systems.
Related Use Cases to the path	4 - Process Frame / Manage Speed
Alternatives	5.1 – No detections made, the system returns and captures another frame of video. 7.1 – No digits identified; the next speed sign is processed.
Related use cases to the alternatives	None
Exceptions	None

Use Cases related to the exceptions	None

Section	
Designation	UC008
Name	Connect
Criticality	High, without this the application will be unable to communicate with the locomotive
Description	Connect the application to the locomotive to enable control inputs to be passed and status values to be returned
Actors	Driver
Scalability	Single Instance
Path	1) Get the DLL location from the settings file. 2) Open the DLL 3) Connect to the locomotive/simulator using the adapter. 4) Set the connection status as true. 5) Log the Boolean response from the simulator.
Related Use Cases to the path	None
Alternatives	None
Related use cases to the alternatives	None
Exceptions	3.1 – Connection Error, set the connection status as false.
Use Cases related to the exceptions	Connection Error

Section	
Designation	UC009
Name	Connection Error
Criticality	Low
Description	An exception logged when unable to connect to the locomotive
Actors	Driver
Scalability	Single Instance
Path	1) Catch the exception. 2) Show an error message to the user that a connection to the locomotive could not be made.
Related Use Cases to the path	None
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC010
Name	Disconnect
Criticality	Low
Description	Disconnect the application and the locomotive
Actors	Driver
Scalability	Single Instance
Path	1) Set the connection status on the locomotive to false. 2) Set the application connection status to false. 3) Test the connection to ensure that a disconnect has occurred.
Related Use Cases to the path	None
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC011
Name	Stop Detection
Criticality	High – If this fails the application continue to detect speed signs and control locomotive speed after it has been requested to end
Description	This enables the user to halt the detection system.
Actors	Driver / Maintainers
Scalability	Single Instance
Path	1) The detection loop is broken. 2) The control loop is broken. 3) The application sets its detection state to false. 4) The start detection button is re-enabled
Related Use Cases to the path	None
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC012
Name	Manage Speed
Criticality	High – If this fails the application will not be able to control locomotive speed
Description	This allows the control of throttle and braking.
Actors	Locomotive Controller
Scalability	Single Instance
Path	1) Get the current speed of the locomotive. 2) Get the speed limit. 3) Calculate the control value based on the difference between the limit and the current locomotive speed. 4) Apply the brake accordingly based on the calculated control value.
Related Use Cases to the path	4 - Decrease Speed
Alternatives	4.1 – If the speed limit is greater than the current locomotive speed apply throttle following the calculated control value
Related use cases to the alternatives	4.1 - Increase Speed
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC013
Name	Increase Speed
Criticality	High – If this fails the application will not be able to control locomotive speed
Description	This allows the control of the throttle.
Actors	Locomotive Controller
Scalability	Single Instance
Path	1) Take the control value. 2) Store this control value within the application. 3) Set the brake to 0. 4) Update the throttle position on the GUI. 5) Update the brake position on the GUI. 6) Send the control value to the locomotive throttle
Related Use Cases to the path	6 – Set Control Value
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC014
Name	Decrease Speed
Criticality	High – If this fails the application will not be able to control locomotive speed
Description	This allows the control of the brake.
Actors	Locomotive Controller
Scalability	Single Instance
Path	1) Take the control value. 2) Store this control value within the application. 3) Set the throttle to 0. 4) Update the throttle position on the GUI. 5) Update the brake position on the GUI. 6) Send the control value to the locomotive brake
Related Use Cases to the path	6 – Set Control Value
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC015
Name	Set Control Value
Criticality	High – If this fails the application will not be able to control locomotive speed
Description	This allows the control values to be sent to the locomotive.
Actors	Locomotive Controller
Scalability	Single Instance
Path	1) Take the control value. 2) Test the connection to the locomotive to ensure a current connection is made. 3) Send the control value to the locomotive.
Related Use Cases to the path	6 – Set Control Value
Alternatives	None
Related use cases to the alternatives	Increase Speed
Exceptions	Not Connected Error
Use Cases related to the exceptions	Not Connected Error

Section	
Designation	UC016
Name	Process Frame
Criticality	High, this is the parent process controlling the detection and identification logic

Description	Run the detection and identification logic on a captured frame
Actors	Locomotive Controller
Scalability	Single Instance
Path	1) Take the capture frame. 2) Run detection. 3) Run Identification. 4) Update the speed limit.
Related Use Cases to the path	1 – Capture Frame, 2 – Detect Speed Signs, 3 – Identify Digits, 4 – Update Speed Limit
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section

Designation	UC017
Name	Identify Digits
Criticality	High
Description	Identify digits on an image
Actors	Locomotive Controller
Scalability	Single Instance
Path	1) Run identification on the frame. 2) Return the detected value.
Related Use Cases to the path	None
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section

Designation	UC018
Name	Update Speed Limit
Criticality	High
Description	Set the speed limit across the application
Actors	Locomotive Controller
Scalability	Single Instance
Path	1) Set the limit on the GUI. 2) Notify the locomotive controller of the updated speed limit.
Related Use Cases to the path	None
Alternatives	1.1 & 2.1 – Speed Limit Passed is 'null' or equal to the current limit, ignore the value
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC019
Name	Detect Speed Signs
Criticality	High
Description	Identify speed signs within the frame
Actors	Locomotive Controller
Scalability	Single Instance
Path	1) Run detection on the frame. 2) Construct bounding boxes around the detections. 3) Crop the image to the detected speed sign. 4) Return the detections
Related Use Cases to the path	None
Alternatives	2.1 – No detections made, return early with 'false'.
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

Section	
Designation	UC020
Name	Capture Frame
Criticality	High
Description	Capture the frame to be processed
Actors	Locomotive Controller
Scalability	Single Instance
Path	1) Load the capture settings from the settings file. 2) Capture a single frame using the settings. 3) Return the frame.
Related Use Cases to the path	None
Alternatives	None
Related use cases to the alternatives	None
Exceptions	None
Use Cases related to the exceptions	None

7.2 APPLICATION MODELLING

7.2.1 Application Wireframing

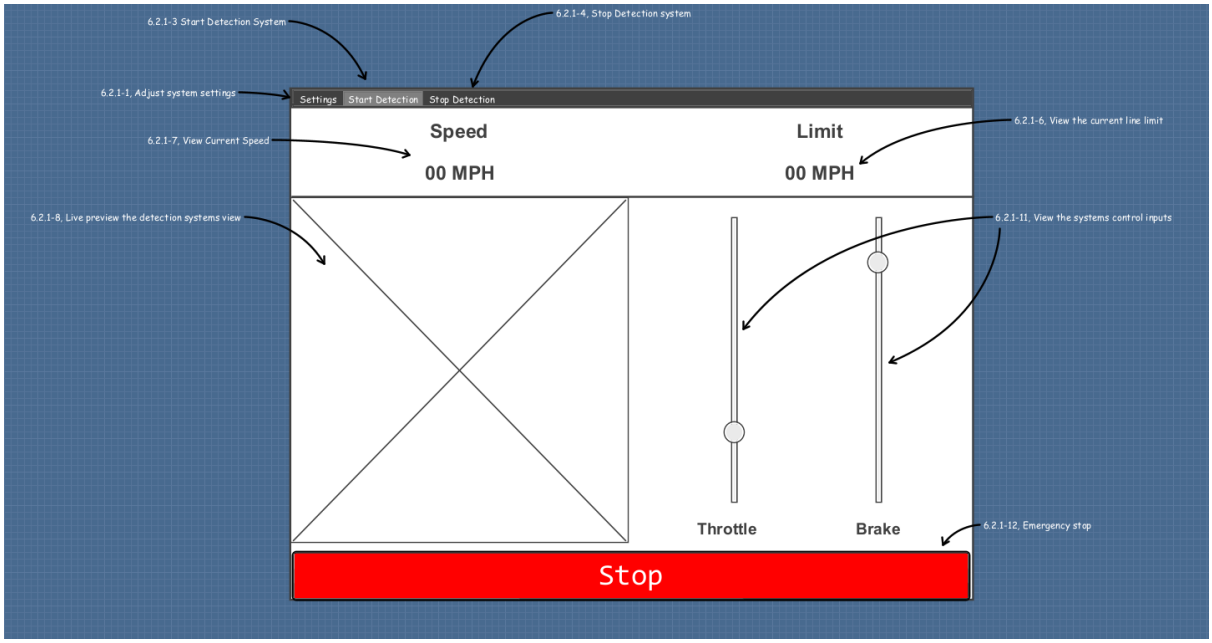

Figure 10 – Main Application Wireframe. All interface elements have been linked back to their specific requirements as listed within the specification.

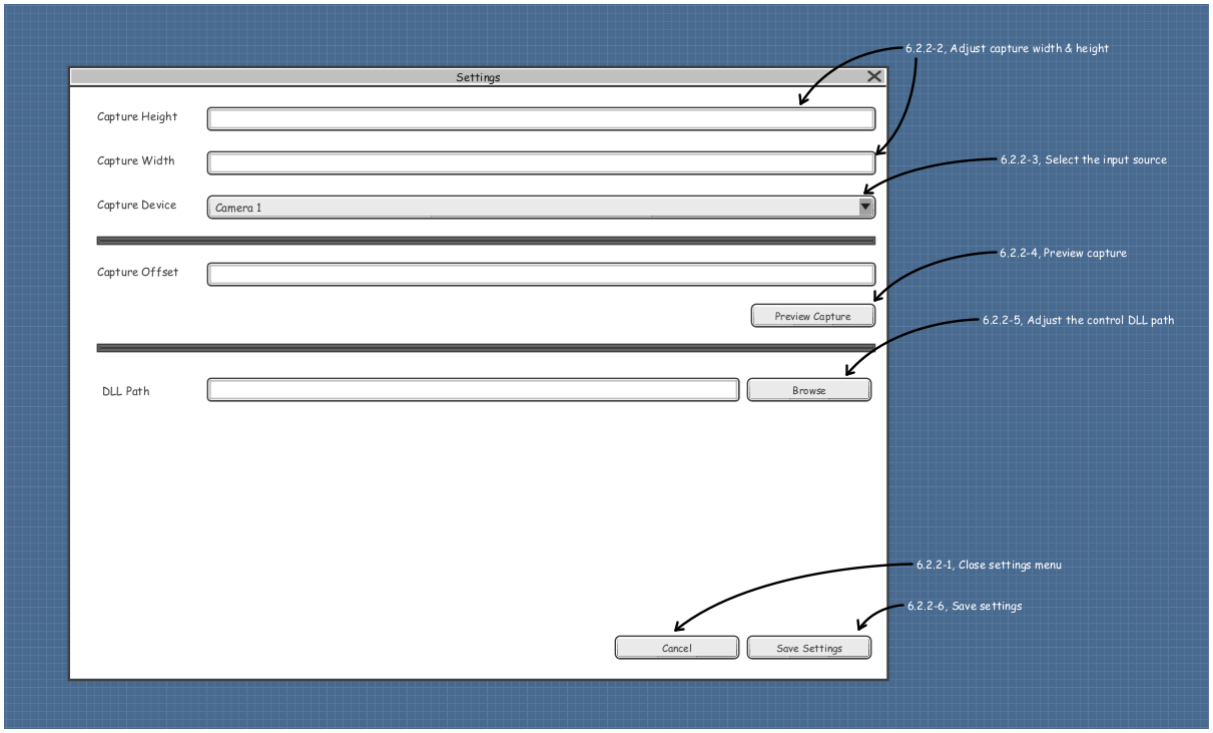

Figure 11 – Settings Menu Wireframe.

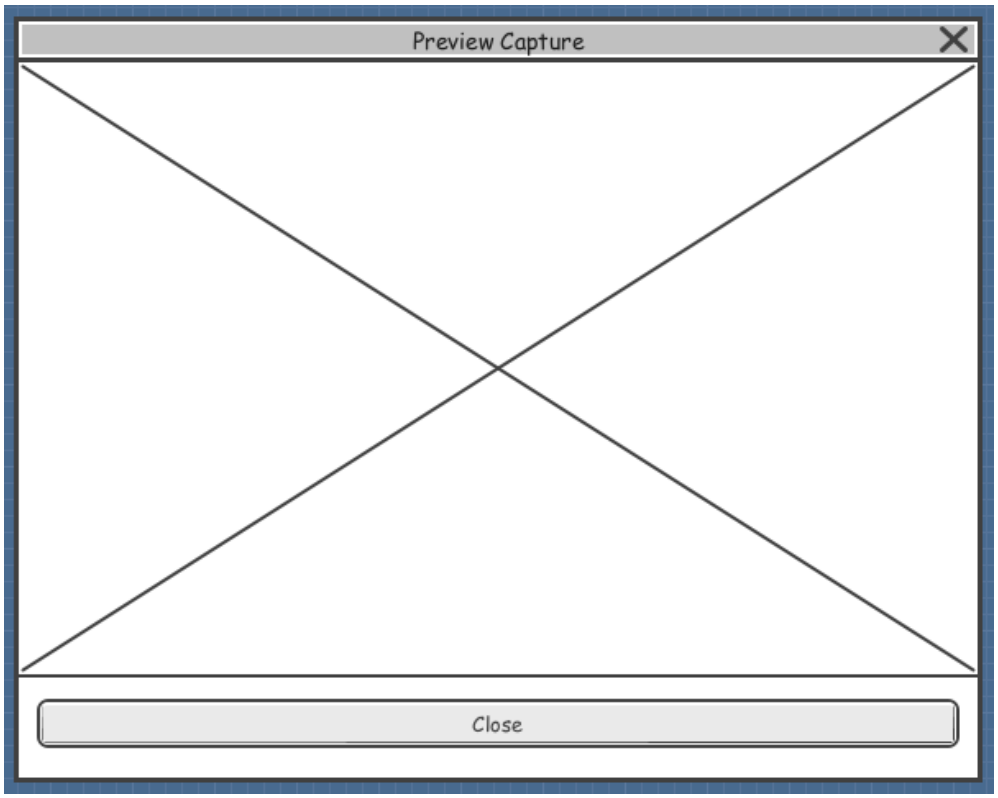

Figure 12 – Preview Capture Wireframe.

7.2.2 Class Diagram

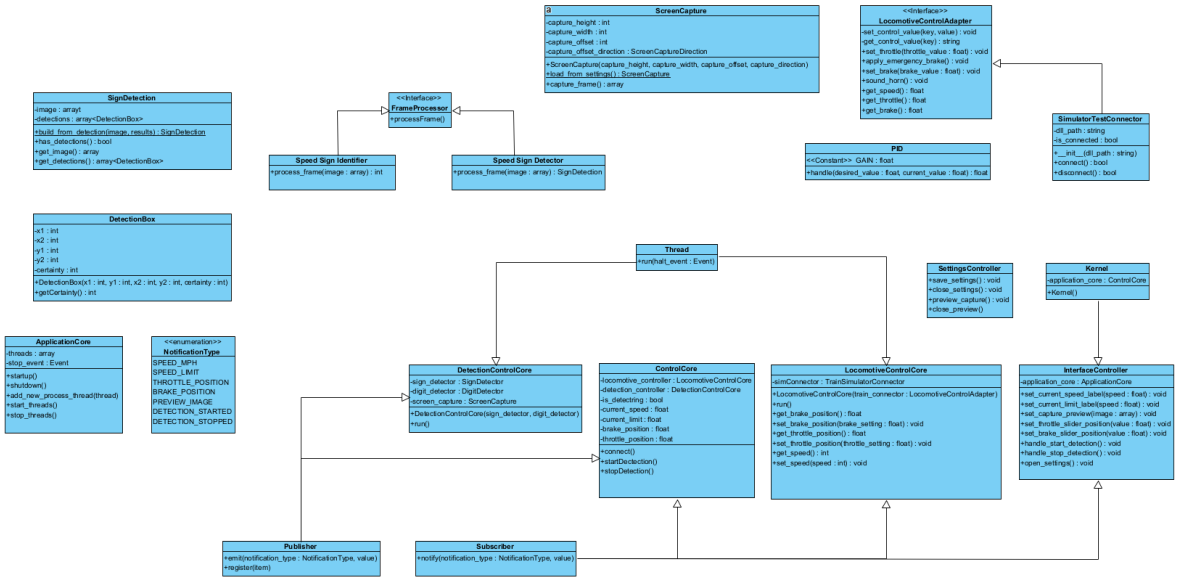

Figure 13 – UML class diagram representing the control structure of the application.

7.2.3 Activity Diagrams

As can be seen below some of the applications functionality has been outlined using activity diagrams, this expresses some of the more complex pathing and functionality and was used as a planning tool to ensure that the latter sequence diagrams could be as details and close to the true implementation as possible.

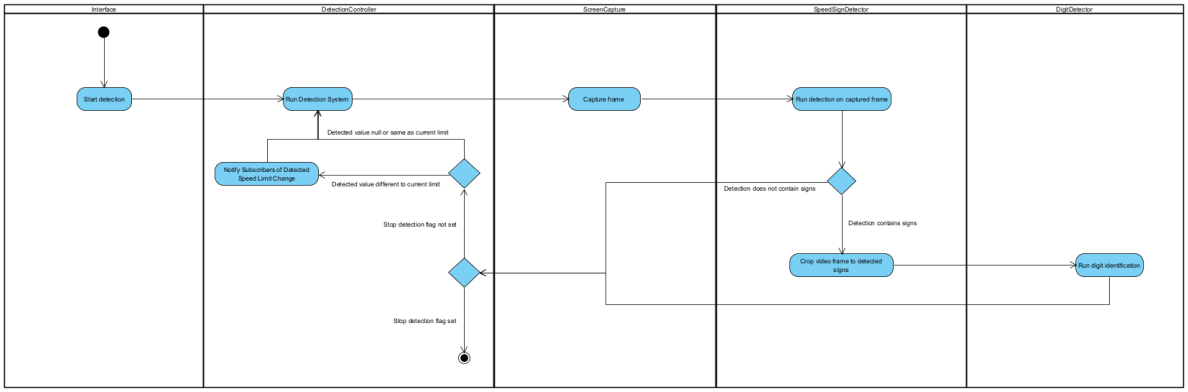

Figure 14 – Top-Down View of how each component within the system will interact when the detection system is running.

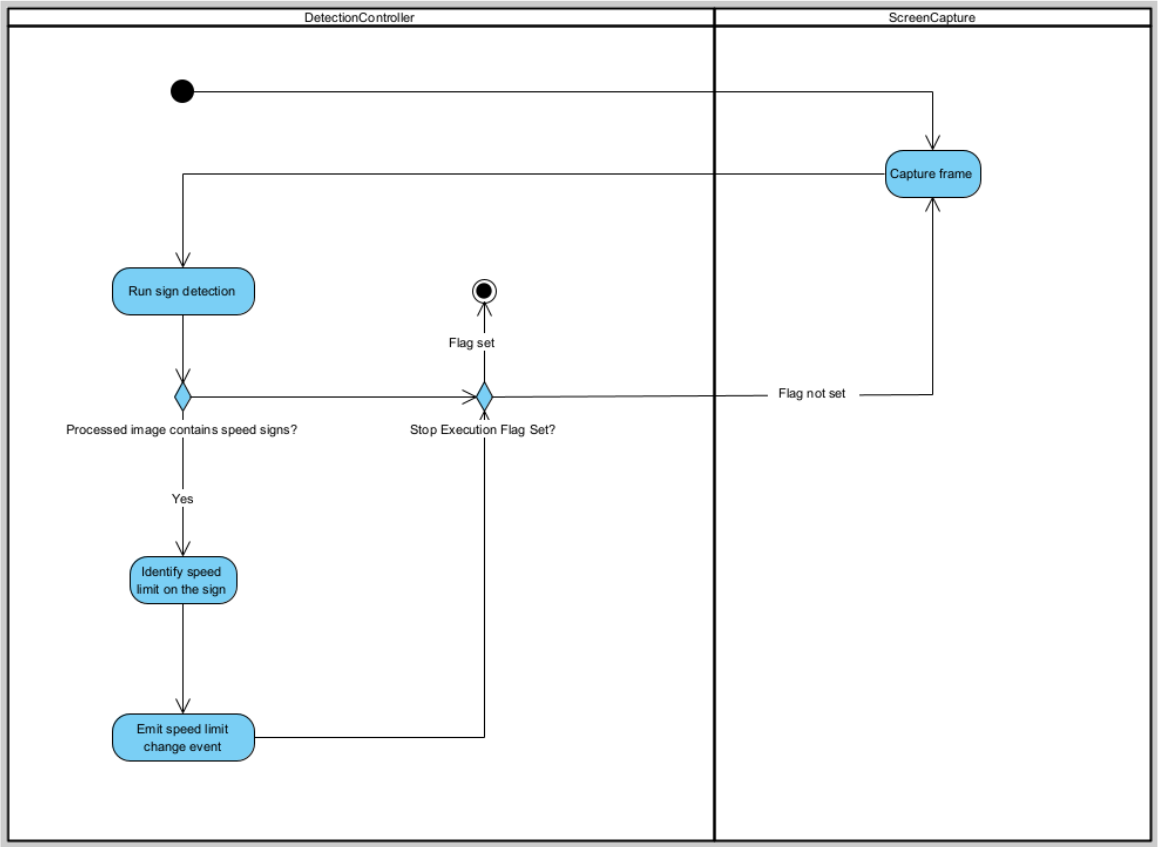

Figure 15 – An activity diagram to document the execution order of events whilst the detection system is running.

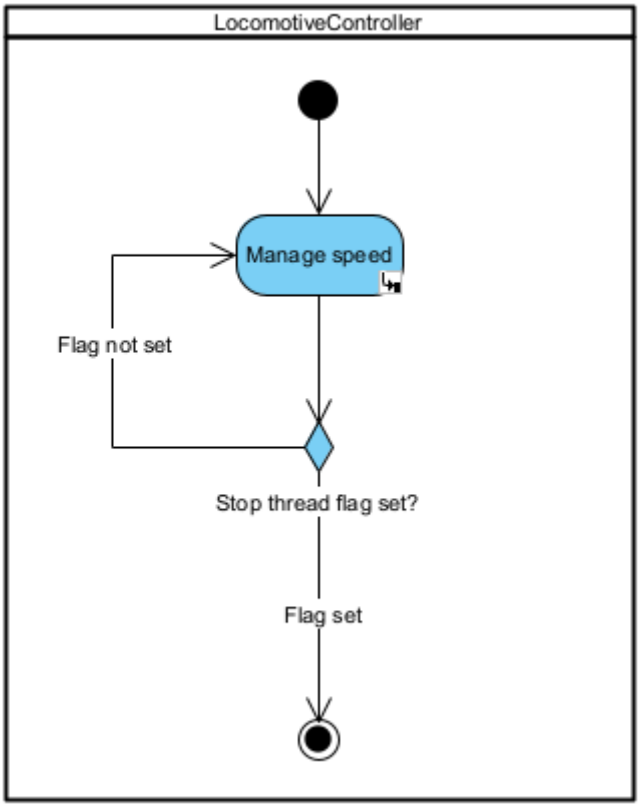

Figure 16 – An activity diagram showing the internal loop used within the locomotive control thread to manage locomotive speed.

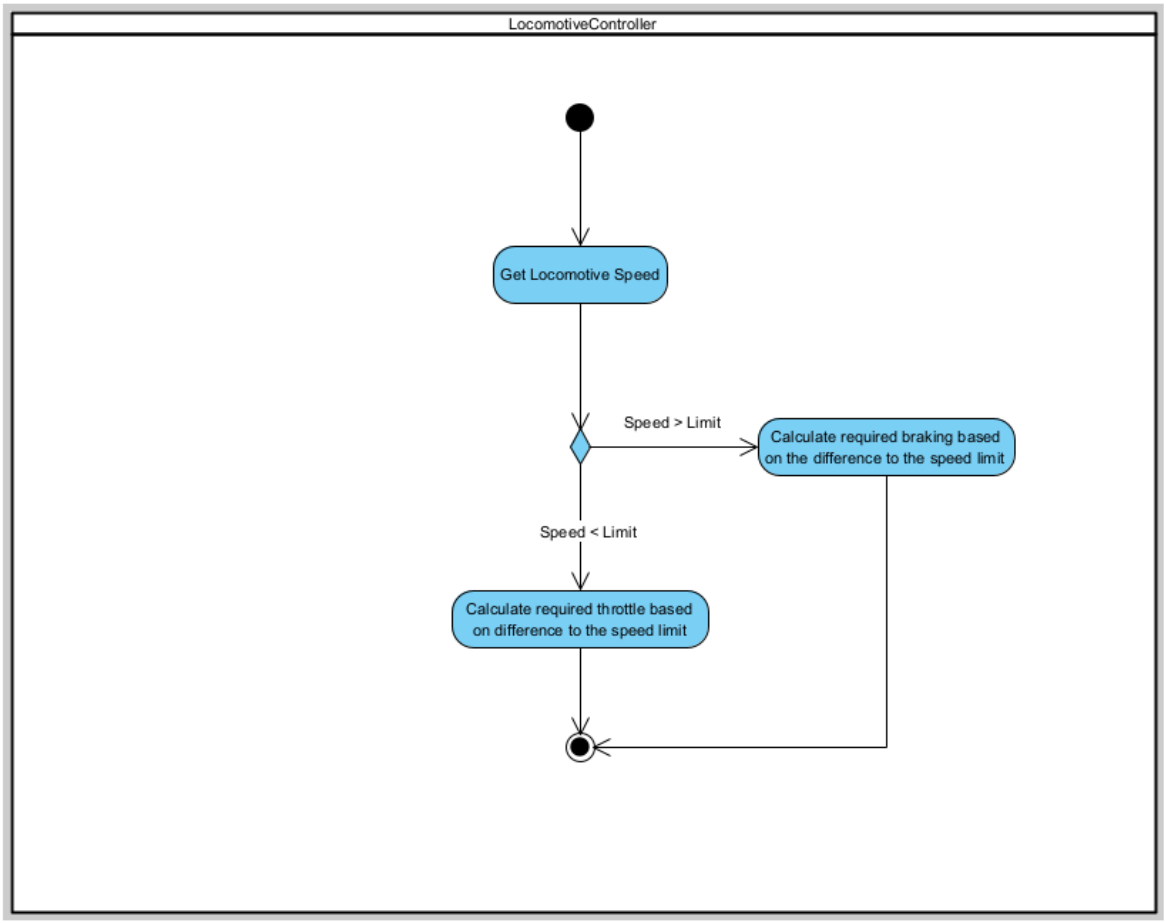

Figure 17 – A sub diagram of the manage speed activity displayed within

Figure 16.

7.2.4 Sequence Diagrams

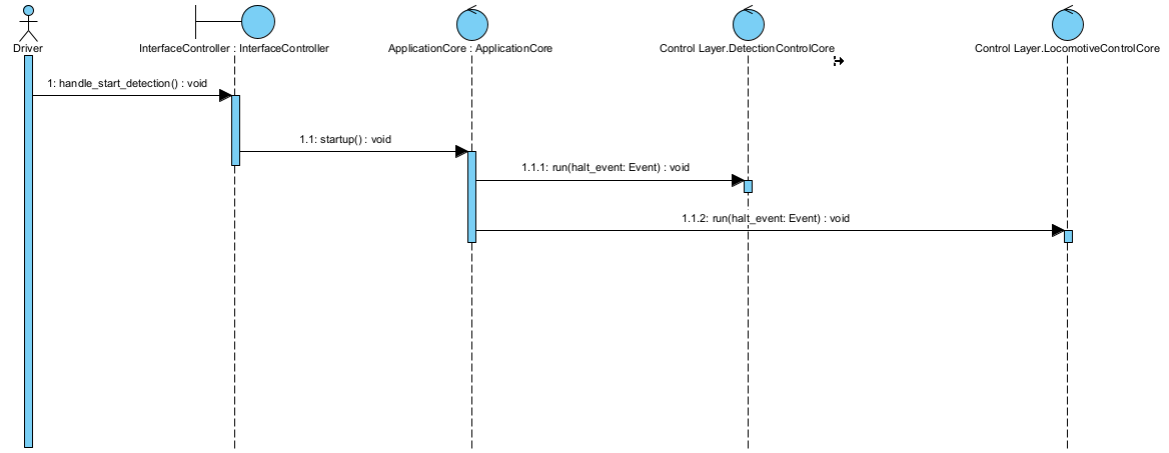

Figure 18 – Sequence diagram demonstrating the method call waterfall when the driver begins the speed sign detection system.

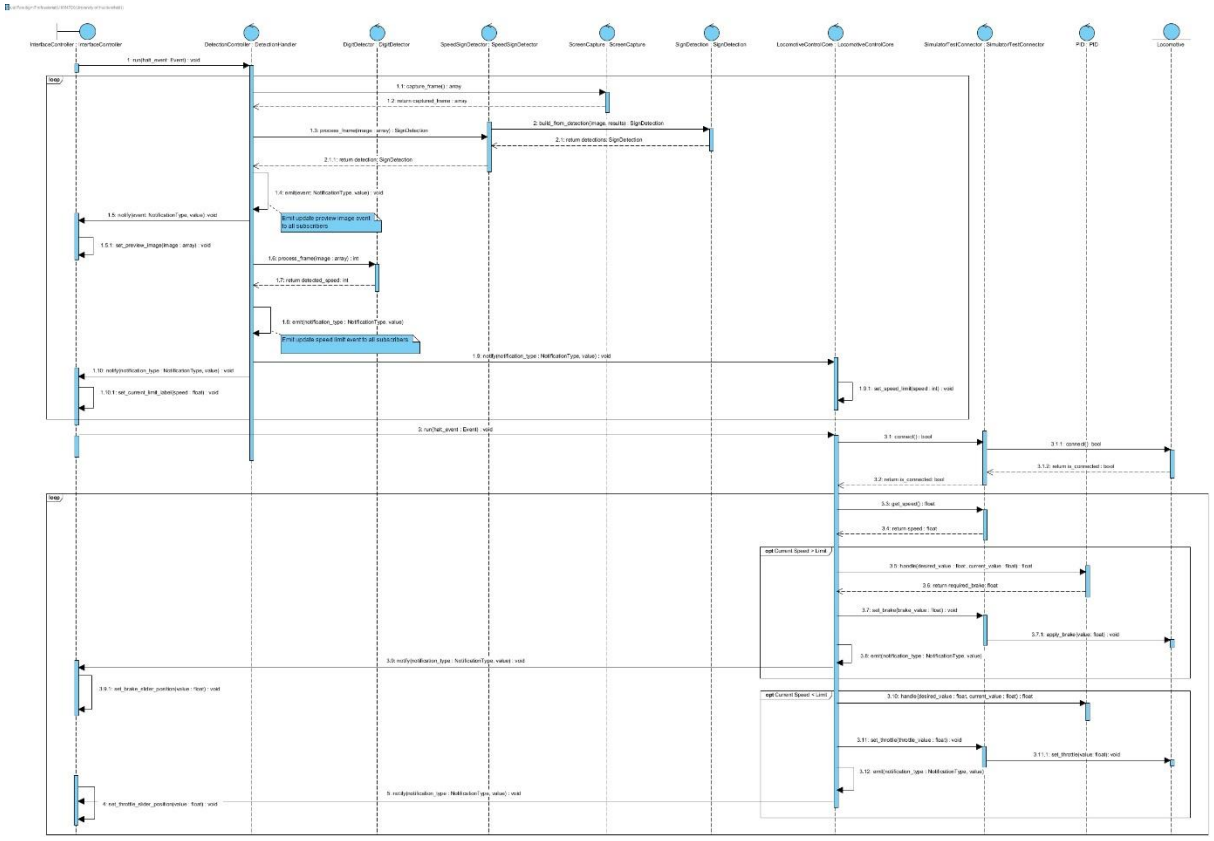

Figure 19 - Sequence diagram mapping method calls to the 'Detection' and 'Control' threads whilst the detection system is active.

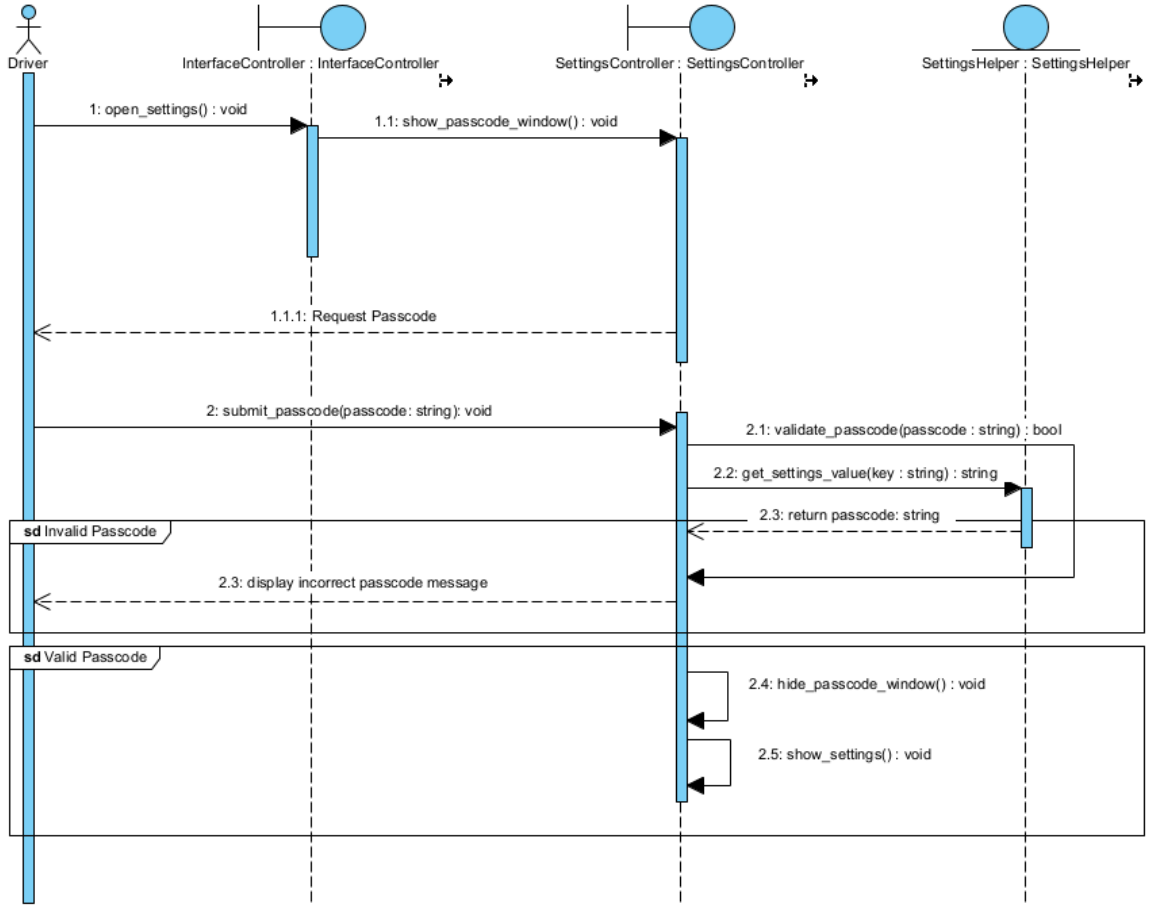

Figure 21 – Sequence diagram demonstrating the 'manage settings' use case.

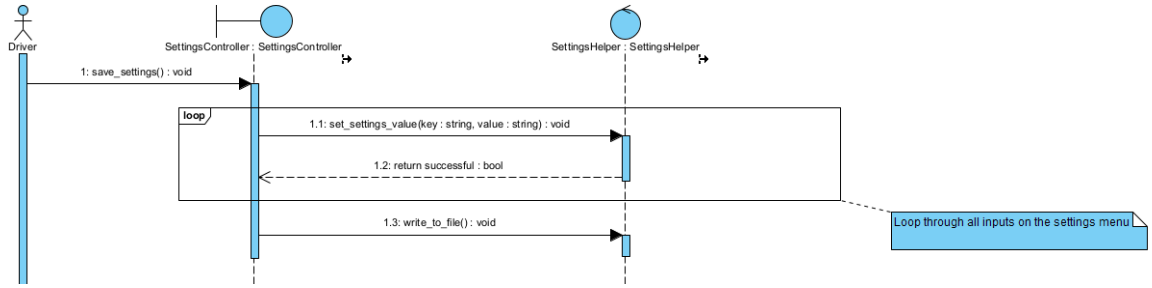

Figure 20 - Sequence diagram detailing the 'Save Settings' use case.

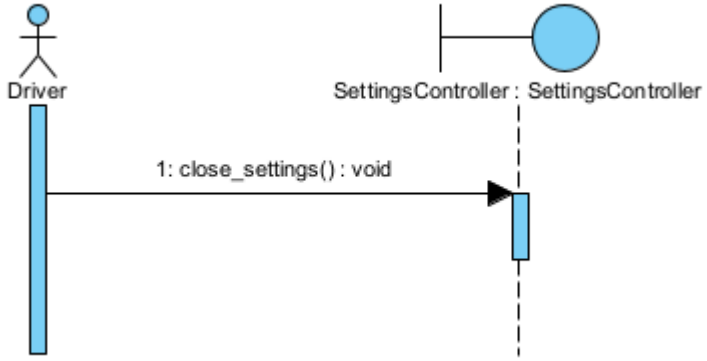

Figure 22 - Sequence diagram detailing the 'Close Settings' use case.

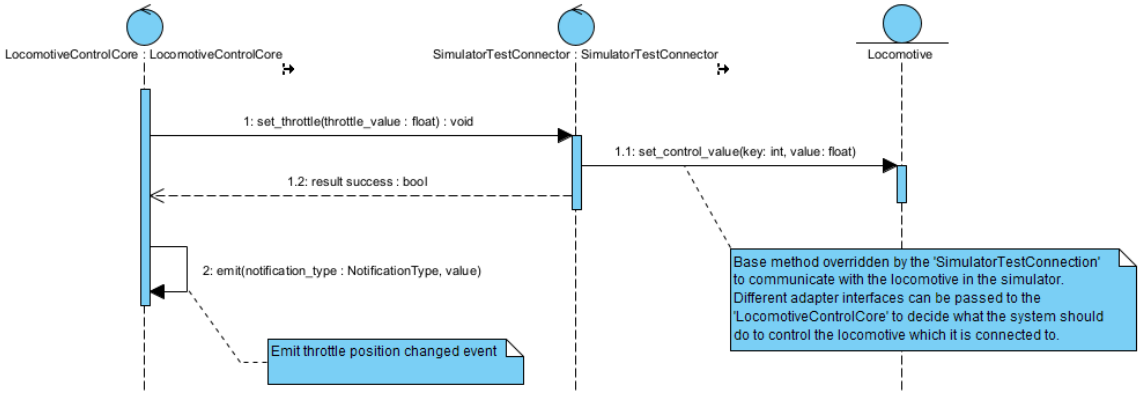

Figure 23 – 'Set Throttle' use case.

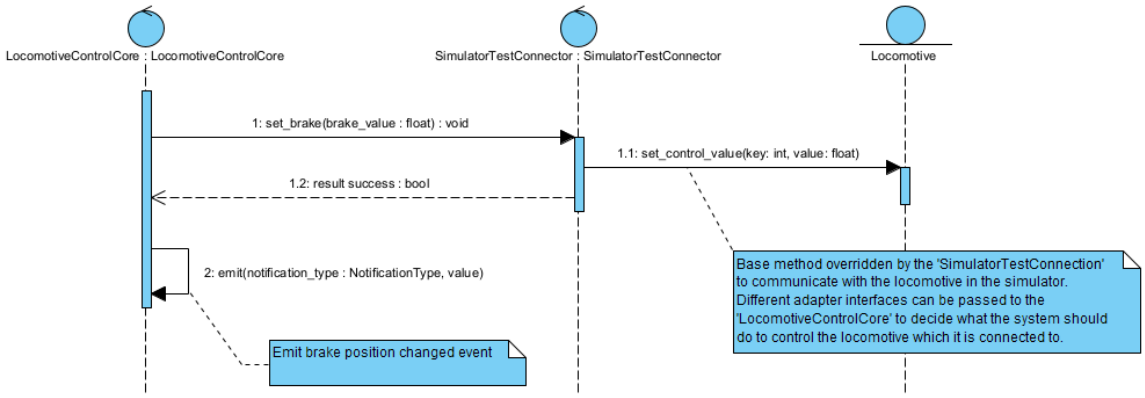

Figure 24 – 'Set Brake' use case.

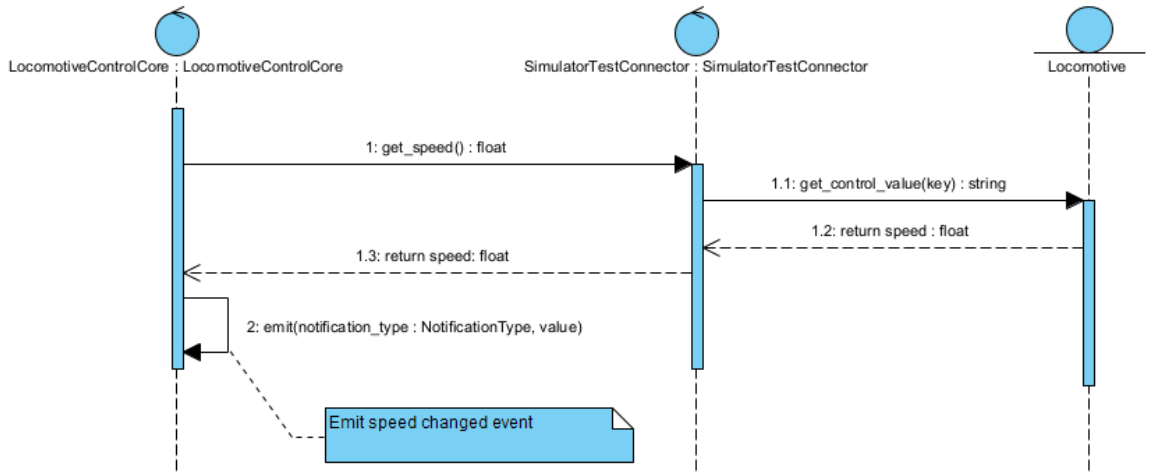

Figure 25 – 'Get Speed' use case.

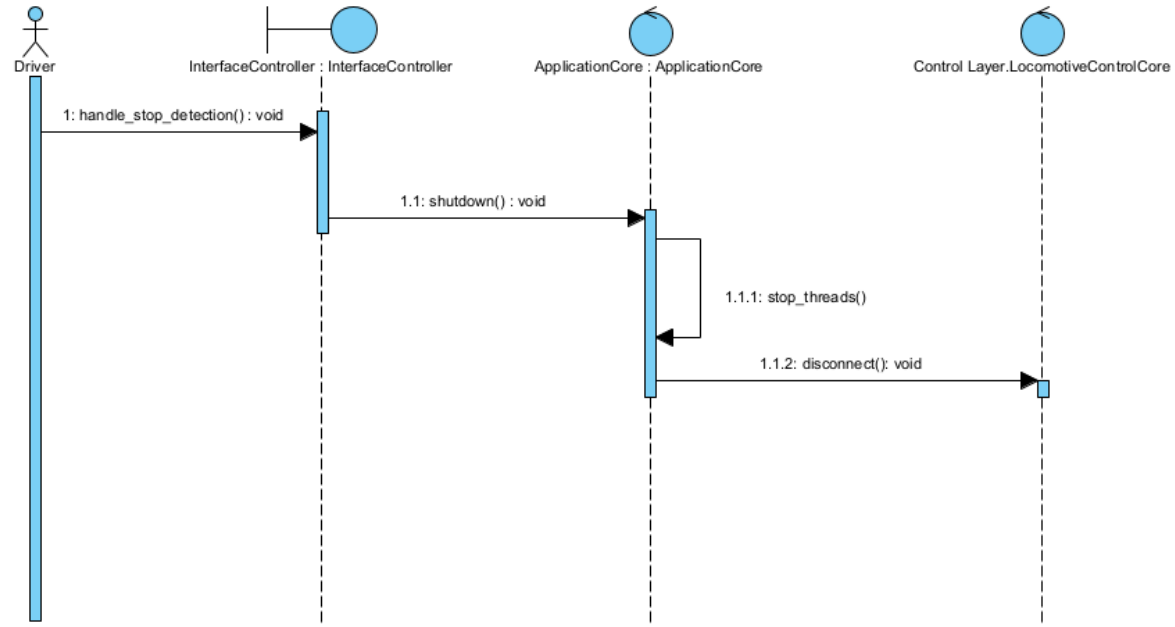

Figure 26 – 'Stop Detection' use case.

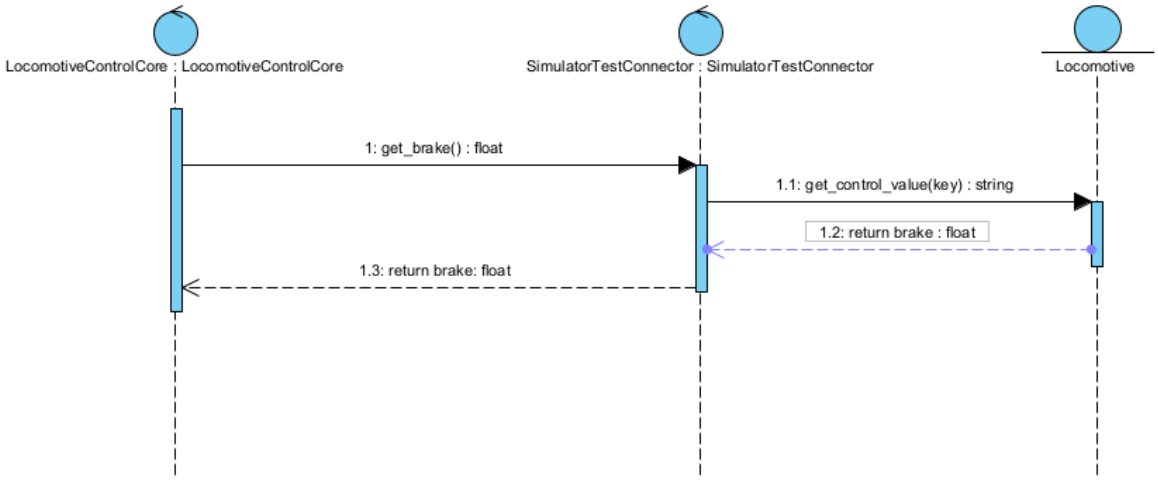

Figure 27 – 'Get Brake Position' use case.

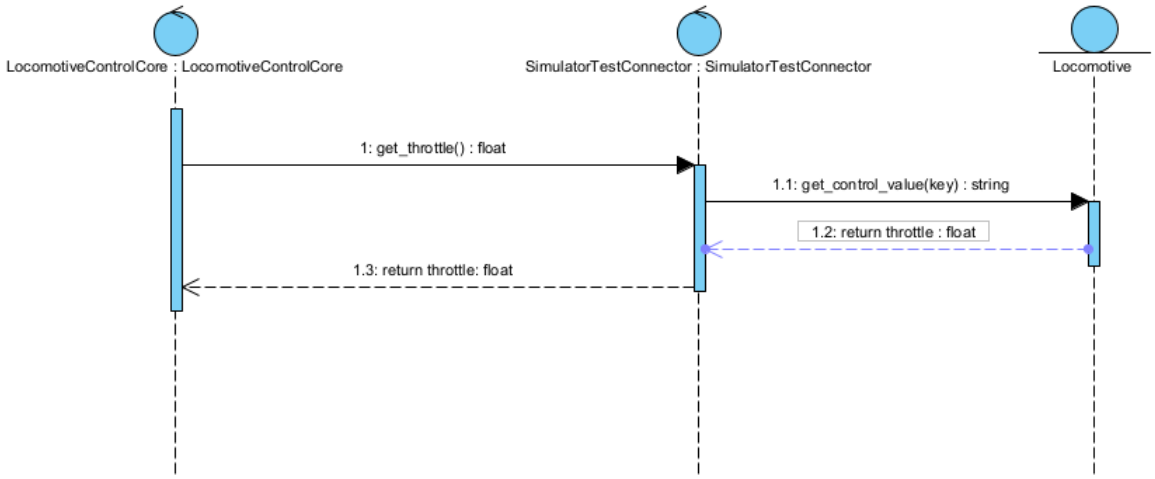

Figure 28 – 'Get Throttle' use case.

7.3 Architecture Modelling

As was discussed within the methodology a decomposition of the planned modules of the project can be seen within Figure 29 – Package Diagram, an overview of how each component within the application will interact and what planned dependencies each component of the application will have.

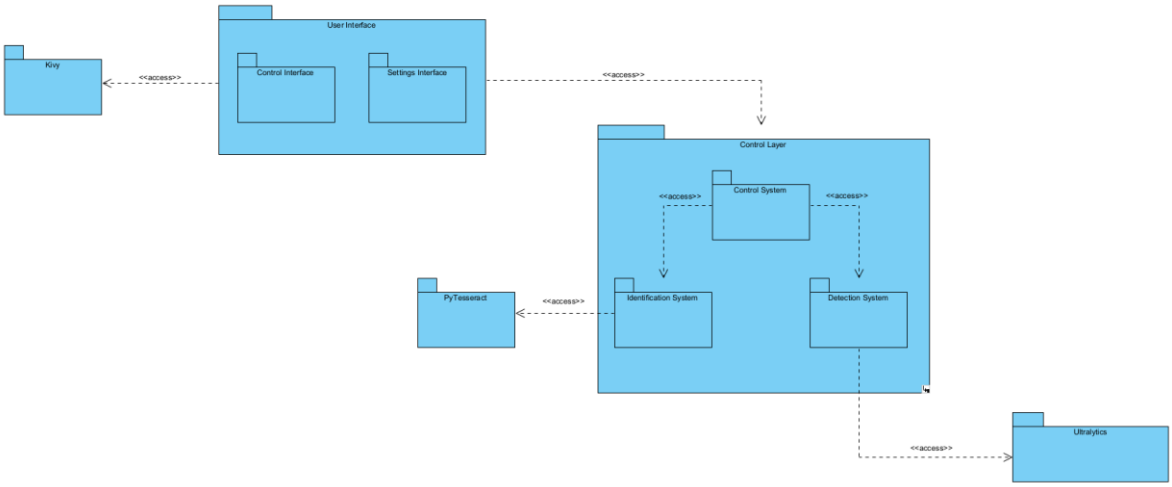

Figure 29 – Package Diagram.

7.4 Software Testing Plan

This project will be divided into three separate testing phases: application testing, model testing and system testing. The former focuses on the user interface and all other modules which will interact with the machine learning core. Model testing will ensure that the model being built is robust and accurate. Where system testing is where we will establish and evaluate the capability of the system which has been produced.

7.4.1 Application Testing

As has already been discussed within Section 5.5 of this document unit testing will be employed to ensure that each component within the software is working as intended, this also extends to the development of end-to-end tests which will ensure that the full detection and control cycle works as intended.

7.4.1.1 Model Testing

During the development of this project, a model will have to be created and trained using the selected object detection library, the selection and evaluation process of which has been discussed previously within the methodology.

7.4.1.2 System Testing

During this phase, the entire application will be tested for functionality within the selected simulator, and a test track will be developed based on the criteria listed below.

- Contain a range of speed limits ranging from 5-90mph.
- Contain a range of different background scenery and clutter.

The system will be measured based on the below metrics:

- How many times the system correctly identifies a speed sign.

To provide an overall better and more rounded evaluation of the system the tests will also be repeated to test the system in different conditions including:

- Adverse weather conditions. (Rain, Snow, Fog)
- Various times of day (Dawn, Day, Dusk, Night)

7.4.1.2.1 Test Track Design

Due to the relative complexity of the train simulator in the game editor, we will re-purpose the 'Falmouth Branch Line' (Dovetail Games, 2010) route for testing by using the in-game editor to adjust the line limits and place a range of different speed limit signs along the route. This route is ideal for testing as is has extensive background clutter including foliage and houses.

7.4.2 Test Plans

TEST PLAN NUMBER	**TP-001**
ASSOCIATED REQUIREMENTS	REQ-001, REQ-030, REQ-032, REQ-004, REQ-017, REQ-031, REQ-032, REQ-034, REQ-035, REQ-036, REQ-037
TESTING STEPS	1) Open the Application. 2) Load Test Route. 3) Configure Application Settings to Point DLL to the Simulator. 4) Ensure Capture Width is the same as the monitor. 5) Click the 'Start Detection'. 6) Allow the application to complete the test route. 7) Click the 'Stop Detection' button.

TEST PLAN NUMBER	**TP-002**
ASSOCIATED REQUIREMENTS	REQ-006
TESTING STEPS	1) Open the Application. 2) Load Test Route. 3) Configure Application Settings to Point DLL to the Simulator. 4) Enable the application debug output. 5) Click the 'Start Detection' 6) View the output in the Python console.

TEST PLAN NUMBER	TP-003
ASSOCIATED REQUIREMENTS	REQ-007
TESTING STEPS	1) Open the application. 2) Install ‘radon’ (rubik, n.d). 3) Run the cyclomatic complexity checker.

TEST PLAN NUMBER	TP-004
ASSOCIATED REQUIREMENTS	REQ-008
TESTING STEPS	1) Open the Application. 2) Load Test Route. 3) Configure Application Settings to Point DLL to the Simulator. 4) Ensure Capture Width is the same as the monitor. 5) Click the ‘Start Detection’. 6) Allow the locomotive to accelerate. 7) Click the ‘Stop’ button.

TEST PLAN NUMBER	TP-005
ASSOCIATED REQUIREMENTS	REQ-040, REQ-041
TESTING STEPS	1) Open the application. 2) Click the ‘Settings’ button. 3) Enter ‘1111’ 4) Click submit. 5) Enter ‘1234’ 6) Click Submit

TEST PLAN NUMBER	TP-006
ASSOCIATED REQUIREMENTS	REQ-059
TESTING STEPS	1) Open the application. 2) Install ‘radon’ (rubik, n.d). 3) Run the maintainability index command.

TEST PLAN NUMBER	TP-007
ASSOCIATED REQUIREMENTS	REQ-020, REQ-021, REQ-029

TESTING STEPS	1) Plug in a webcam to the computer. 2) Open the application. 3) Click the ‘Settings’ button. 4) Enter ‘1234’. 5) Click Submit. 6) Click the ‘Capture Preview’ button. 7) Close the ‘Capture Preview’. 8) Change the capture device to the webcam. 9) Click the ‘Capture Preview’ button. 10) Close the application.

TEST PLAN NUMBER	**TP-008**
ASSOCIATED REQUIREMENTS	REQ-025, REQ-027
TESTING STEPS	1) Open the application. 2) Click the ‘Settings’ button. 3) Enter ‘1234’ 4) Click Submit 5) Click the ‘Capture Preview’ button. 6) Close the ‘Capture Preview’ 7) Change the capture height to 300px. 8) Click the ‘Capture Preview’ button. 9) Change the capture height to 1000px. 10) Close the application.

TEST PLAN NUMBER	**TP-009**
ASSOCIATED REQUIREMENTS	REQ-026
TESTING STEPS	1) Open the application. 2) Click the ‘Settings’ button. 3) Enter ‘1234’ 4) Click Submit 5) Click the ‘Capture Preview’ button. 6) Close the ‘Capture Preview’ 7) Change the capture width to 300px. 8) Click the ‘Capture Preview’ button. 9) Change the capture width to 1000px. 10) Click the ‘Capture Preview’ button. 11) Close the application.

TEST PLAN NUMBER	**TP-010**
ASSOCIATED REQUIREMENTS	REQ-028
TESTING STEPS	1) Open the application. 2) Click the ‘Settings’ button. 3) Enter ‘1234’ 4) Click Submit 5) Change the capture width to ‘abc’. 6) Change the capture height to ‘def’. 7) Click the ‘Capture Preview’ button. 8) Close the application.

TEST PLAN NUMBER	TP-011
ASSOCIATED REQUIREMENTS	REQ-033, REQ-044
TESTING STEPS	1) Open the application. 2) Click the 'Settings' button. 3) Enter '1234' 4) Click Submit 5) Change the capture width to '50. 6) Change the capture height to '50'. 7) Click the 'Save' button. 8) Close the application. 9) Open the application. 10) Click the 'Settings' button. 11) Enter '1234' 12) Click Submit

TEST PLAN NUMBER	TP-012
ASSOCIATED REQUIREMENTS	REQ-010
TESTING STEPS	1) Open the python console. 2) Run the python validation script from the project directory located in 13.6.

TEST PLAN NUMBER	TP-013
ASSOCIATED REQUIREMENTS	REQ-022, REQ-024
TESTING STEPS	1) Open the Application. 2) Load Test Route. 3) Configure Application Settings to Point DLL to the Simulator. 4) Ensure Capture Width is the same as the monitor. 5) Click the 'Start Detection'. 6) Allow the application to complete the test route. 7) Whilst the application is completing the route record how many signs are correctly identified. 8) Click the 'Stop Detection' button. 9) Close the application. 10) Repeat for each weather condition and time of day as required.

7.5 DATASET SPECIFICATION

As part of the Object detection subnet, a range of training and testing images will have to be collected from the system test environment. A set of requirements has been created to ensure that the model is provided with consistent images to produce an optimal result, the dataset created should:

- Contain 750 images.
- Be split into training and testing subsets based on an 80:20 split.
- Contain at least 1 speed limit sign per image.
- Only contain UK speed limit signs.
- Contain a range of realistic speed limit signs ranging from 5 - 125mph.
- Contain speed signs captured from a range of angles including but not limited to the angle at which a sign would be captured from the cab of a locomotive.
- Focus on capturing signs from a distance to ensure that the trained model can identify signs from a distance.
- Only contain images of signs where the digits are still legible.

Figure 30 - Example Training image as per the specification listed above.

The dataset will not contain any augmented images, this is a common approach used to increase the size of smaller datasets to increase the overall accuracy however given that the system should only be used to identify signs from the position of a locomotive this technique will not be employed.

7.6 Model Training Methodology

As part of the training approach for this application a series of steps have been defined which will be followed when training the model, these are:

1) Train a YOLOv8 model using the dataset created using transfer learning from the YOLOv8n model.
2) Continue the training until loss levels have not decreased significantly for at least 10 epochs.
3) Validate the generated model against the testing set and record accuracy.

If the accuracy of the model does not appear to be able to meet a testing accuracy of at least 90% by the end of the 5th generation, additional images will be added to the dataset up to a maximum of 1,500 images. If the model is still not appearing to reach the accuracy threshold results will be accepted as is, thus it will also be discussed in the system evaluation.

8 IMPLEMENTATION

8.1 SPRINT PLANNING

As was previously discussed, this project will employ a scrum-based approach to development, the first and key aspect of this is splitting the tasks to be undertaken during the development of the project, these are linked back to the requirements specified within section 6 of this document. As can be seen in Figure 31, the requirements specification was split into individual tasks. These were then organised into individual sprints, as can be seen on Figure 32. Any incomplete tasks at the end of a given sprint are placed onto the backlog to be re-assigned to later sprints.

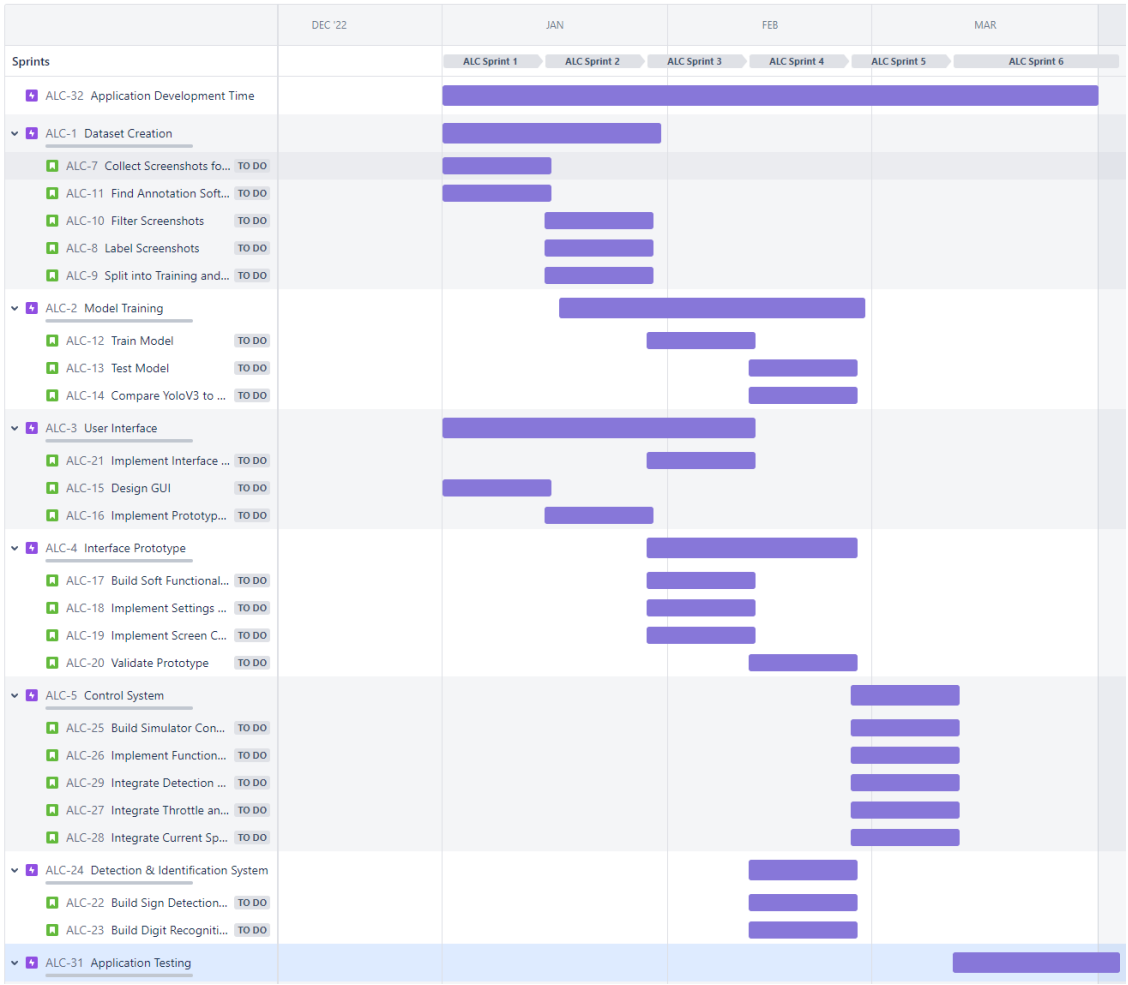

Figure 31 – Development Roadmap Based on Planned Sprints.

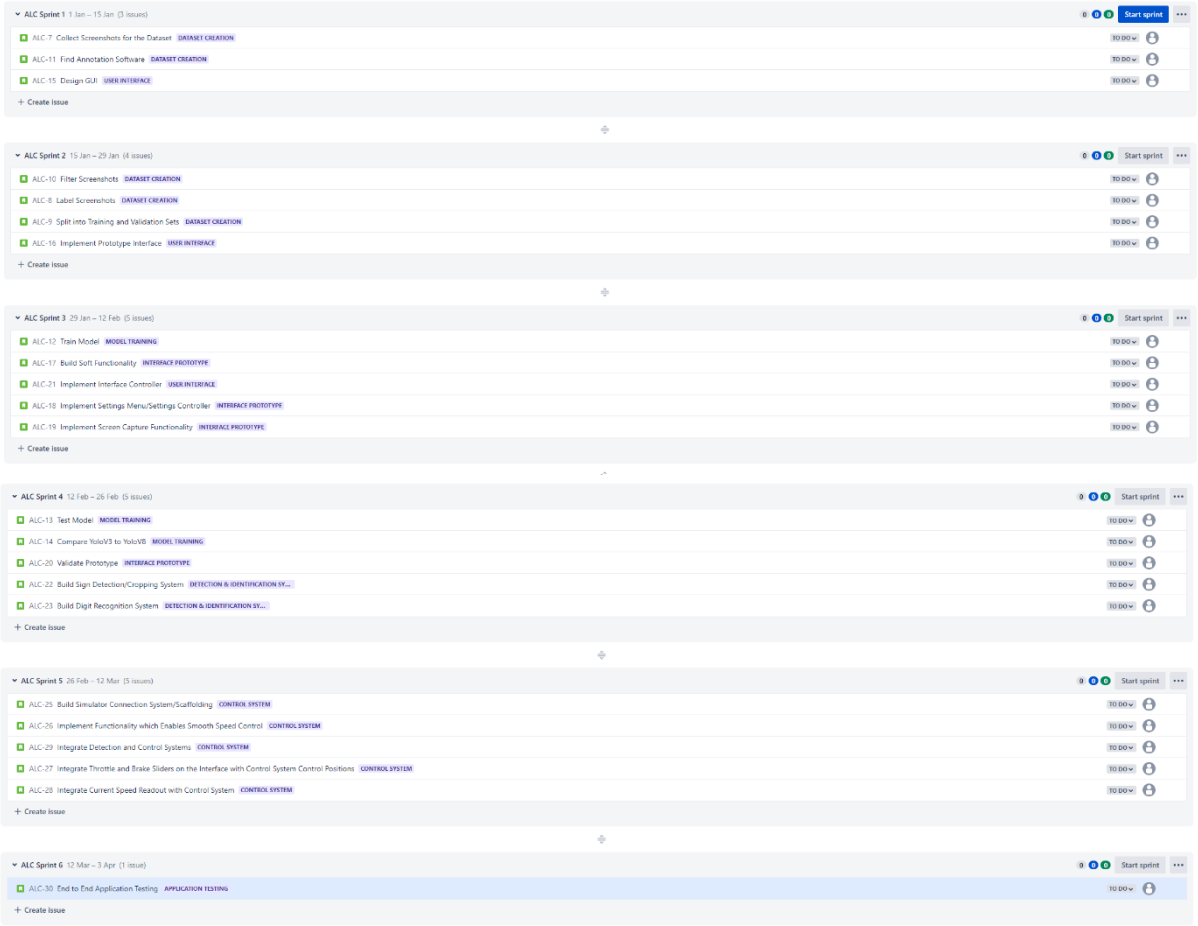

Figure 32 – Details of All 6 Planned Sprints.

8.2 Sprint 1 – Dataset Creation

Some of the time allocated to this sprint was intended for the development of wireframes related to the project however as can be seen in previous sections these have already been completed as part of the design of the application therefore additional time was allocated to the collection of the dataset. Following the methodology outlined within Section 7.5, 750 images were collected from a range of routes within Train Simulator Classic. The screenshot function built into the Steam in-game overlay was used extensively to capture these images. A snapshot of the full dataset can be seen in Figure 33 – Captured Dataset. The number of tasks contained within the first sprint was significantly less than there could be as the sprint was completed ½ a week ahead of schedule. As a result, the tasks within the subsequent sprints were moved forward accordingly. Software for annotation of the images was also found in the form of LabelStudio (HartexLabs, 2023), an open-source data labelling tool which supports the correct annotation format as used by Ultralytics.

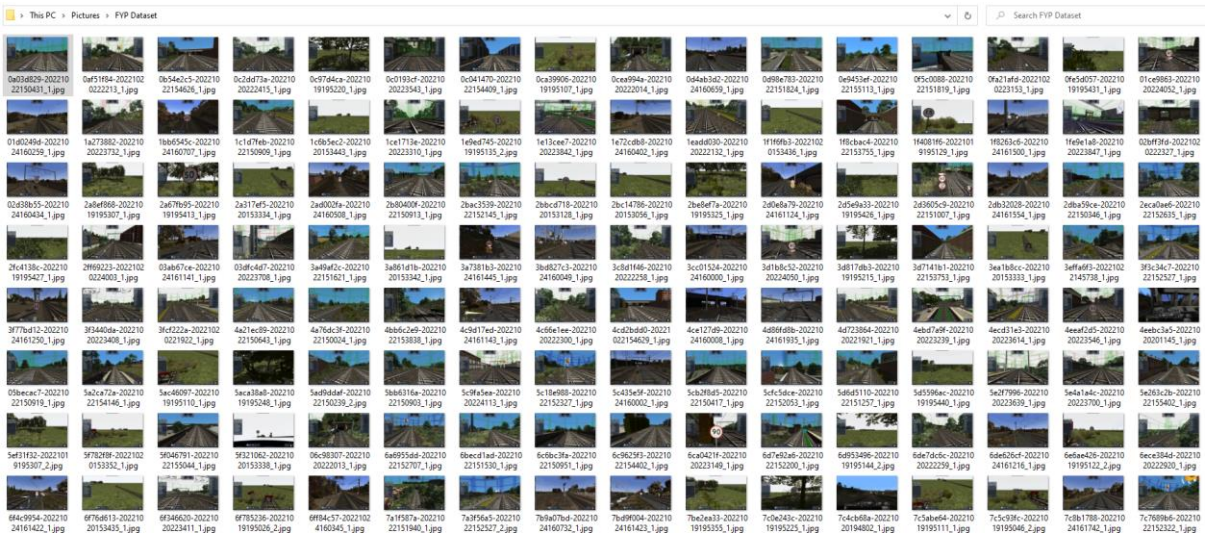

Figure 33 – Captured Dataset.

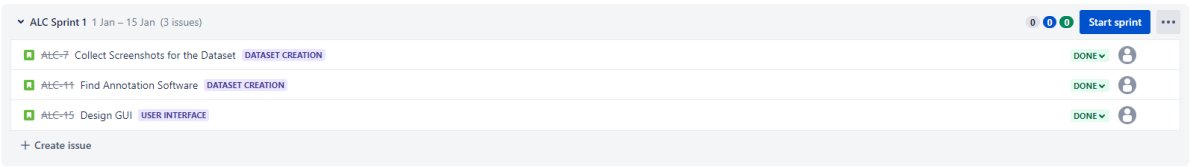

Figure 34 – Completed Sprint.

8.3 Sprint 2 – Dataset Creation Cont. & Interface Prototypes

As planned, within this sprint a user interface prototype was created using Kivy, the rationale for this has been discussed previously within the methodology of this document. One of the key issues encountered during development was the default background colour, black. This made it difficult to style the application. During the first user interface prototype, it was also made obvious that due to how Kivvy is structured it would not be possible to customise certain elements such as sliders or disabled buttons.

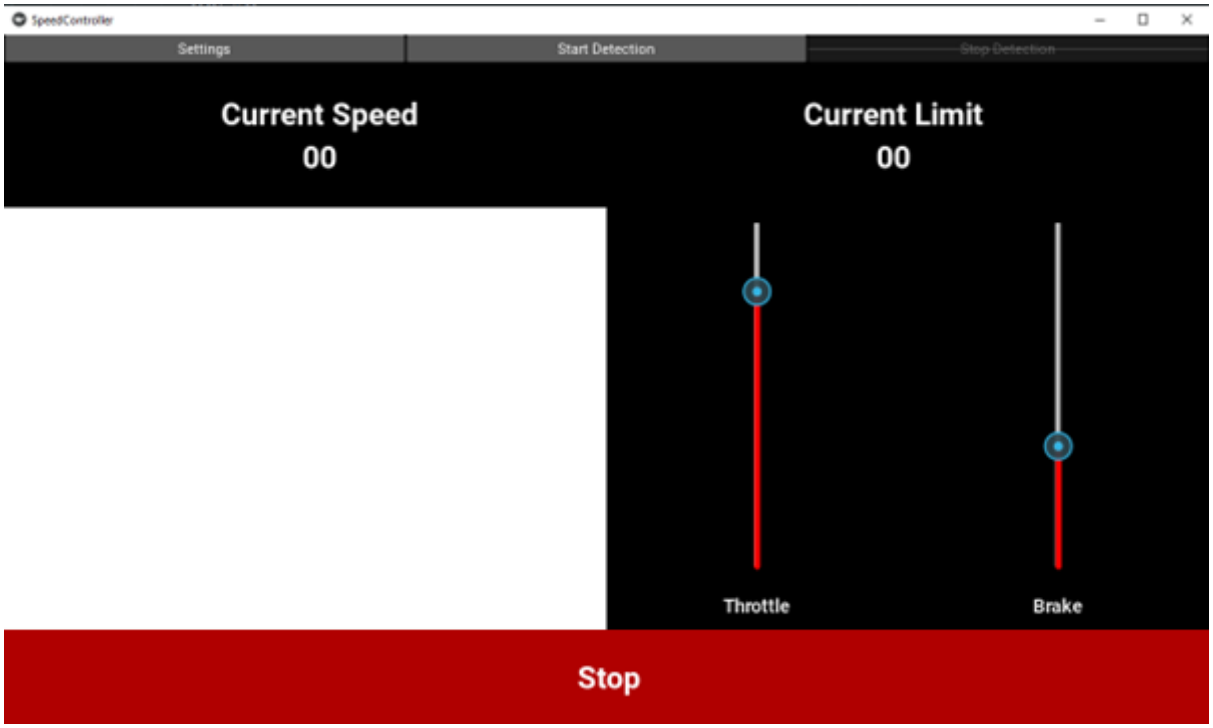

Figure 35 – Interface First Prototype.

KivyMD (kivymd, n.d) was found as an alternative offering. This extension is based on Kivvy however extends many of the widgets to be fully material design compliant thus making them more customisable. The change between the two packages was extremely quick as was discussed within the methodology the prototype system would only contain a non-functional interface thus allowing elements to be changed.

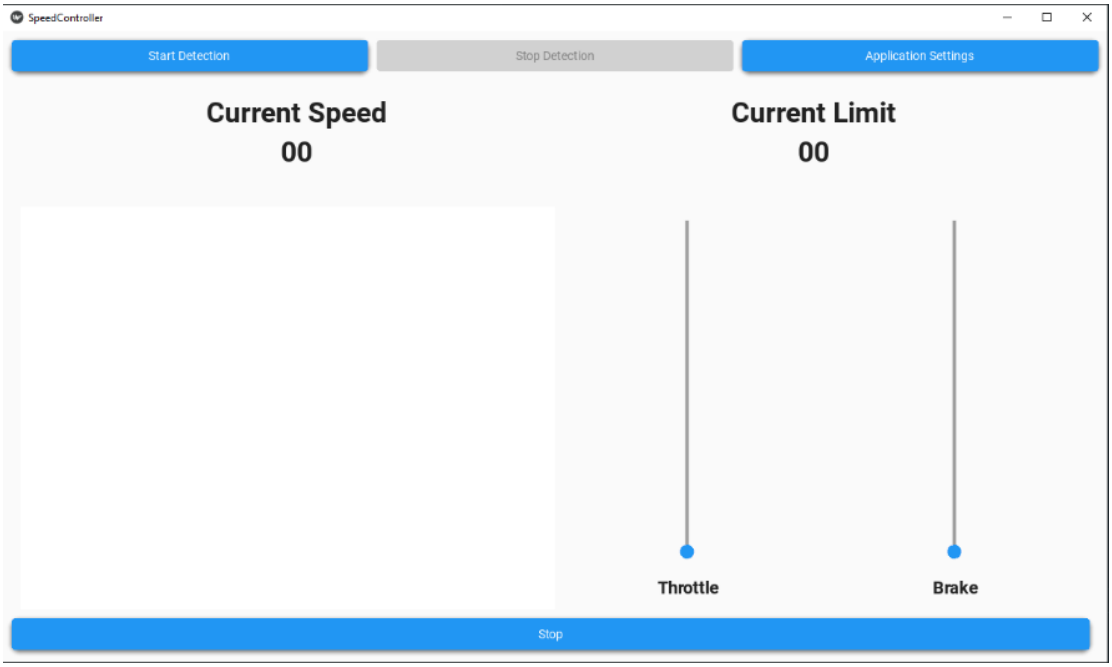

Figure 36 – Interface Second Prototype.

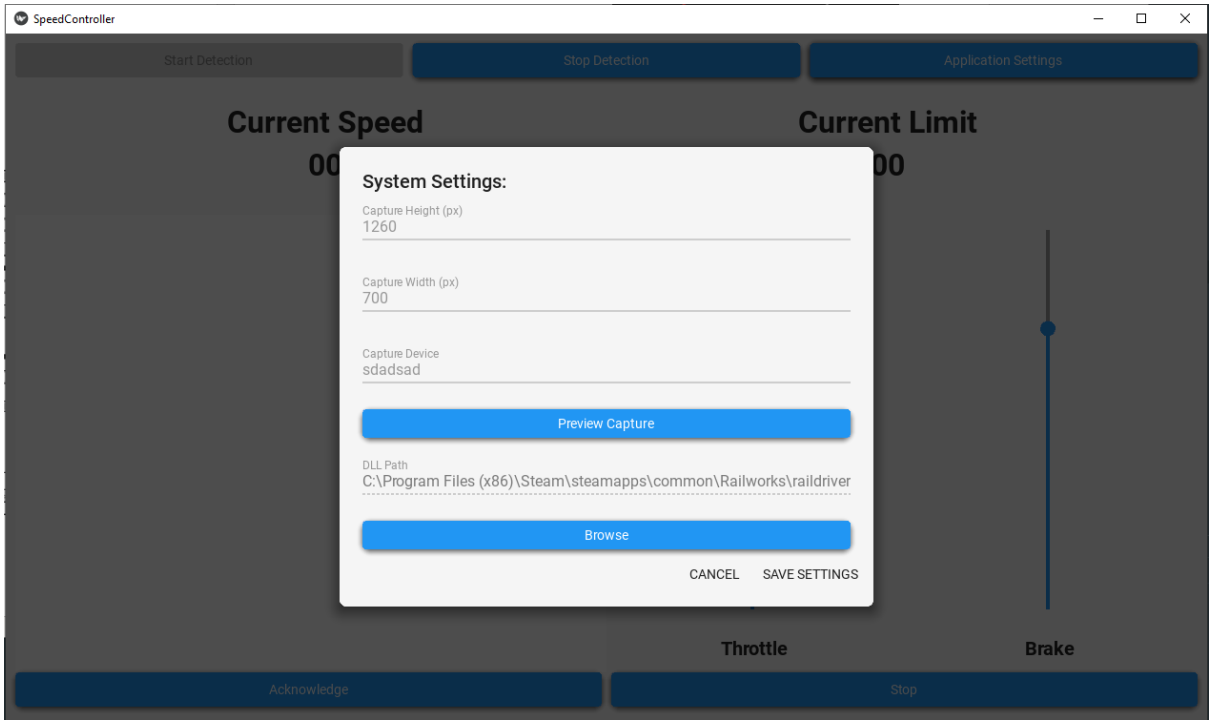

Figure 37 – Settings Menu Second Prototype.

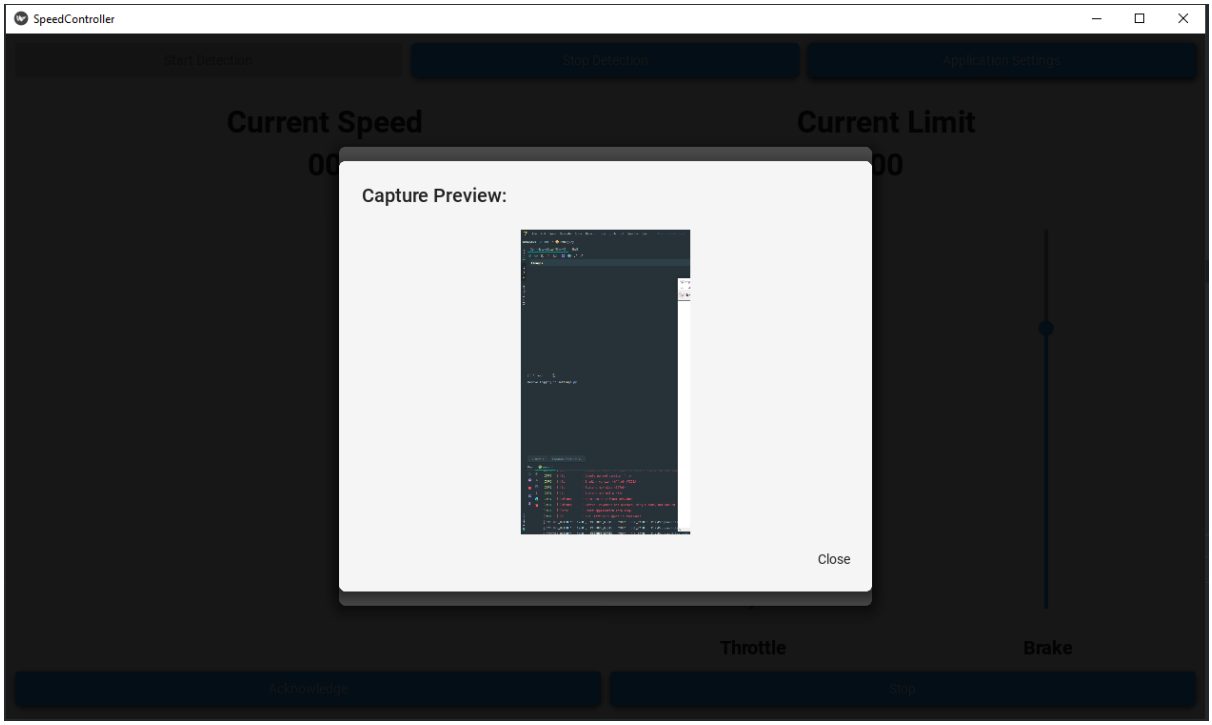

Figure 38 – Capture Preview Prototype.

The secondary task being dataset creation, this is key as the next sprint focuses on training the model using Ultralytics. Images within the dataset were filtered accordingly, some signs were not as visible as they appeared within the game, some images were removed. At the same time, the dataset was split 80:20, training, and testing. With 750 just over 750 images in the dataset 600

training images and 150 validation images. These were then labelled; this was a fully manual process which required the developer to individually draw bounding boxes around each speed sign Examples of this can be seen below in Figure 39.

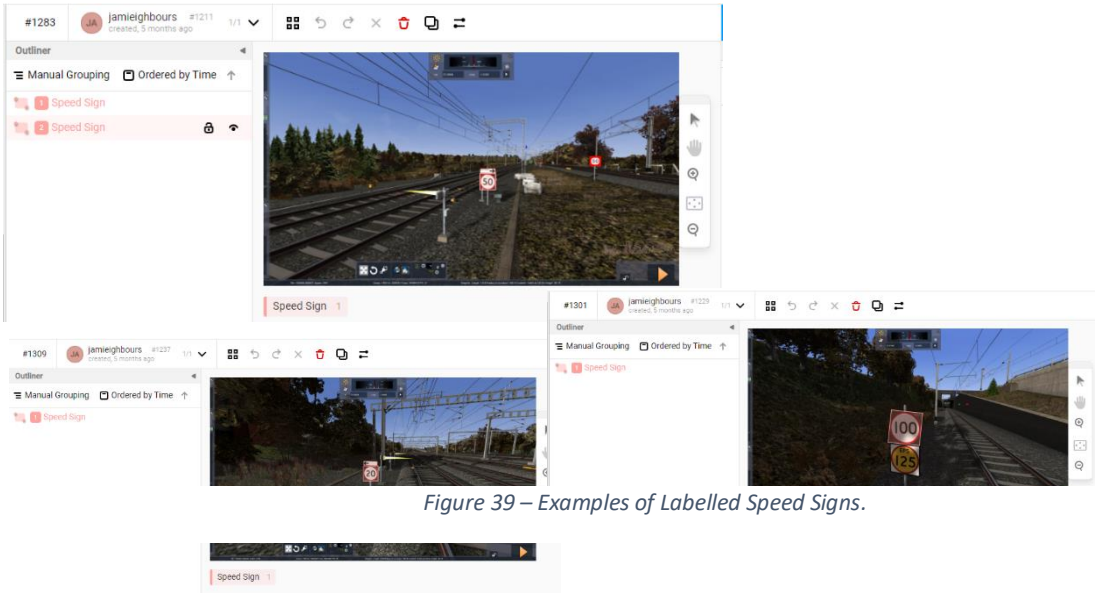

Figure 39 – Examples of Labelled Speed Signs.

Upon completion all associated tasks were then marked as complete on Jira.

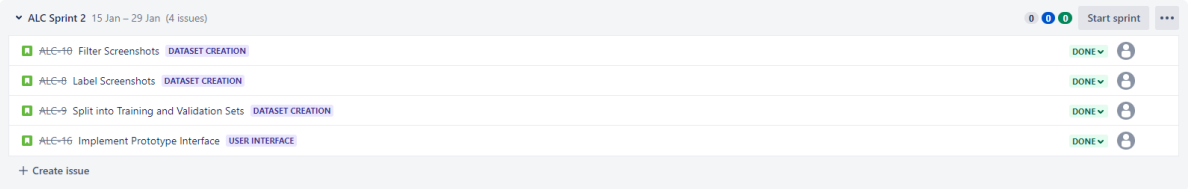

Figure 40 – Completed Sprint 2.

8.4 Sprint 3 – Model Training and Interface Functionality

The focus of this sprint is developing a script to train the model using the Ultralytics package. The script used to train the mode can be seen within the appendix of this document under 13.5.

```
Starting training for 50 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/50         0G        1.1      4.734     0.9598         35        640:   5%|▌         | 2/39 [00:15<04:37,  7.49s/it]
```

Figure 41 – Model Training Output.

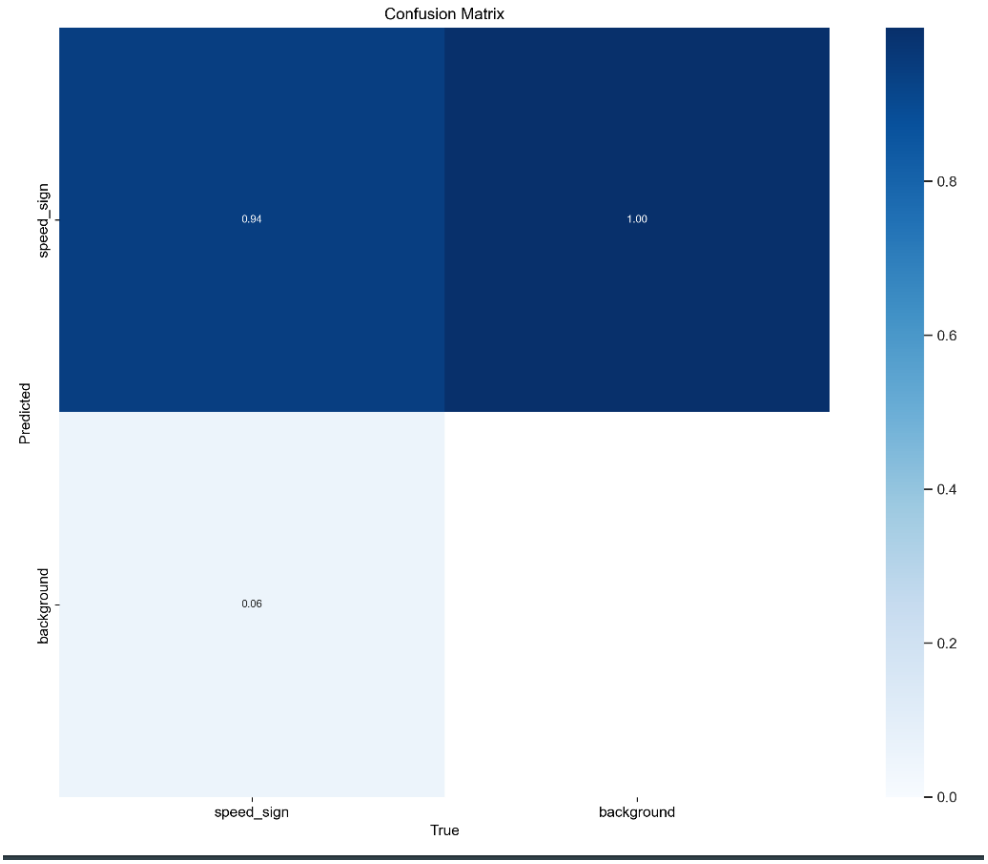

Figure 42 – Confusion Matrix for the trained model, this was automatically generated by the Ultralytics library (Ultralytics, n.d). Accuracy of 94% was achieved after completion of 50 epochs.

The second focus within this sprint was to develop basic interface functionality and implement the methods outlined within Figure 13 – UML class diagram representing the control structure of the application Figure 13 under the 'Interface Controller' and 'Settings Controller'. This consisted of basic functionality such as disabling the 'stop detection' button when the start detection button was pressed and so forth. At this stage the full settings menu was also implemented, reading, and writing to a JSON file stored within the project, this gives the system the ability to permanently store data simply, the alternative would be to store the settings data as a CSV however the JSON file format is better for storing keyed data such as system settings. A database was not an option as was discussed within the requirements of the project; the developed software should be as lightweight as possible.

This sprint also encompasses the first stage of the detection system, the screen capture system, this was implemented using the MSS python package. The screen capture system is currently used by the settings menu controller to provide a capture preview window, it was also noted that the ability to set a capture offset would not be required as the capture system automatically defaults to the left side of the monitor when capturing a screenshot, therefore reducing the capture width has the same effect as a manual offset. On a railway line, it can also be noted that speed signs relevant to the locomotive travelling along a track are always located on the lefthand side, therefore this functionality will be dropped from the product due to its redundancy.

8.4.1 Training Issues

One issue encountered during the training of the model was caused by mislabelled signs, some of the annotations had been corrupted when exporting from label studio, this caused the first model to identify every pixel within a frame as a speed sign, upon re-exporting the annotations and re-training the model this error disappeared.

8.5 Sprint 4 – Detection System

Implementation of the detection system was a straightforward process, this consisted of talking the pre-trained model from the previous sprint and building the methods to pass data to and interrogate the model. As has been discussed when planning the application architecture, the software will make use of two subnets the first will be used to detect speed signs within a given frame, the second is used to identify the digits within the frame. This sprint also consisted of building the scaffolding to enable this.

8.6 Sprint 5 - Control System & Integration

8.6.1 Control Prototype

As was previously discussed within this document, the application will be tested using Train Simulator Classic, this is due to the availability of the control API, documentation on offer and flexibility of the software.

One of the limitations to this was the lack of API documentation, only minimal reference was made to the API using a basic google search, a document which had likely been previously attached to a forum post by C. Gamble (2014) was found on the 3rd page of google which contained some information on the methods which could be used to integrate with the train simulator API.

In addition to this, an application called ‘Dependencies’ (lucasg, 2021) was also made use of to establish what other functions could be accessed.

E	Ordinal	Hint	Function	VirtualAddress	Demangler
	1 (0x0001)	N/A	ClearChanged	0x00001100	None
	2 (0x0002)	N/A	ControllerChanged	0x00001160	None
	3 (0x0003)	N/A	GetControllerList	0x00001050	None
	4 (0x0004)	N/A	GetControllerValue	0x00001060	None
	5 (0x0005)	N/A	GetCurrentControllerValue	0x000010d0	None
	6 (0x0006)	N/A	GetLocoName	0x00001040	None
	7 (0x0007)	N/A	GetNextRailDriverId	0x00001390	None
	8 (0x0008)	N/A	GetRailDriverConnected	0x00001280	None
	9 (0x0009)	N/A	GetRailDriverGetId	0x000013f0	None
	10 (0x000a)	N/A	GetRailDriverGetType	0x00001410	None
	11 (0x000b)	N/A	GetRailDriverValue	0x000013c0	None
	12 (0x000c)	N/A	GetRailSimCombinedThrottleBrake	0x00001270	None
	13 (0x000d)	N/A	GetRailSimConnected	0x00001030	None
	14 (0x000e)	N/A	GetRailSimLocoChanged	0x00001250	None
	15 (0x000f)	N/A	GetRailSimValue	0x000011c0	None
	16 (0x0010)	N/A	IsLocoSet	0x000012c0	None
	17 (0x0011)	N/A	SetControllerList	0x000012f0	None
	18 (0x0012)	N/A	SetControllerValue	0x00001130	None
	19 (0x0013)	N/A	SetLocoName	0x00001290	None
	20 (0x0014)	N/A	SetRailDriverConnected	0x00001010	None
	21 (0x0015)	N/A	SetRailDriverLocoChanged	0x00001440	None
	22 (0x0016)	N/A	SetRailDriverValue	0x00001420	None
	23 (0x0017)	N/A	SetRailSimConnected	0x00001380	None
	24 (0x0018)	N/A	SetRailSimControllerMinMax	0x00001350	None
	25 (0x0019)	N/A	SetRailSimControllerValue	0x00001320	None
	26 (0x001a)	N/A	SetRailSimValue	0x00001190	None

Figure 43 – Output from the 'Dependencies' (lucasg, 2021) application.

Based on the documentation provided by Dovetail Games I was also able to establish the full range of control inputs offered by the simulator, not all locomotives implemented all inputs however this documentation was enough to enable me to control a locomotive.

Once the prototype had been refined appropriately it was then integrated with the detection system which we developed within sprint 3.

8.6.2 Multithreading Issues

One of the issues which arose during the integration of the components was caused by the disconnected nature of them. For example, when an event was emitted to update the speed limit on the interface, this would be called by the detection thread, this caused the interface and detection system to hang. The solution to this was to add an annotation to the update methods of the interface which instructed them to always be run using the main thread (the interface thread).

8.6.3 Smooth Acceleration & Braking

An unplanned issue which was encountered due to lack of forethought was the implementation of the acceleration and braking control, initially it was to be implemented using a look up table containing braking values at set speed differences however this was proven to be quite inefficient and difficult to program, instead a PID system was implemented which based on a gain value proportionally controlled the locomotives speed.

8.7 Sprint 6 – Application Testing

The duration of this sprint was extended to allow for additional time to test the application, the results of which can be seen below in section 9.

9 SOFTWARE TESTING

9.1 APPLICATION TESTING

As has been discussed previously within the methodology of this project, testing of the application has been conducted throughout the development process using the test plan defined below, this section outlines the final iteration of the test plan run against the finished product. All requirements outlined within the requirements diagrams in 7.1.1 have been translated into the tables below to ensure that all requirements initially requested have been met.

9.1.1 Automatic Locomotive Control System

REQUIREMENT	DESCRIPTION	TEST PLAN	PASS/FAIL	EVIDENCE	NOTES
REQ-001	The system should be able to detect railway speed signs within a video frame.	**TP-001**	Pass	*See Appendix -13.7, Figure 46* – REQ-001	Start Detection button is disabled when detection is running.
REQ-006	The system must process speed signs in real time, the system should be able to process a video frame in 200ms (5FPS).	**TP-002**	Pass	Average detection cycle time over 210 detection cycles 271.4ms, significantly lower when not running in parallel with the simulator due to the processing overhead. *Appendix - 13.7,* Figure 51	When debug mode is enabled, and the application is running framerate debug information is printed to the Python console.
REQ-007	The system should have a cyclomatic complexity less than 10.	**TP-003**	Pass	*Appendix - 13.7,* Figure 52	None

REQ-008	The system should have the facility to enact an emergency stop.	**TP-004**	Pass	*See Appendix -13.7, Figure 46* – REQ-001	‘Stop’ button is present and accessible within the interface.
REQ-030	The application should have a user interface.	**TP-001**	Pass	*See Appendix -13.7, Figure 46* – REQ-001	None
REQ-032	Due to cost and safety requirements the system should be able to connect to a rail simulator for testing.	**TP-001**	Pass	*See Appendix -13.7,* Figure 67	Simulator connection tested during test runs.
REQ-040	The settings menu should have a password to prevent unauthorised access.	**TP-005**	Pass	*Appendix - 13.7,* Figure 51	Settings Menu is only shown when the password is correctly entered, or the debug mode is enabled.
REQ-041	If a password is entered incorrectly the user should not be allowed into the settings menu.	**TP-005**	Pass	*Appendix - 13.7,* Figure 48	None.
REQ-043	The password should be encrypted and stored securely.	**N/a**	Pass	*Appendix - 13.7,* Figure 49	Password is hashed and stored in the settings file using argon.
REQ-045	All code written should conform to	**N/a**	Pass	N/a	N/a

	PEP8 coding standards.				
REQ-059	The system should have an average maintainability index above 70.	**TP-006**	Pass	*Appendix - 13.7,* Figure 50	Average Index Value, 72.98

9.1.2 User Interface

REQUIREMENT	DESCRIPTION	TEST PLAN	PASS/FAIL	EVIDENCE	NOTES
REQ-004	The driver should be able to view the control inputs of the system.	**TP-001**	Pass	*Appendix - 13.7,* Figure 46	Throttle and Brake control inputs are displayed on the interface.
REQ-017	The driver should be able to control the detection system.	**TP-001**	Pass	*Appendix - 13.7,* Figure 46 *&* Figure 53	Evidence displays the system when detecting and when stopped.
REQ-020	Change the capture device from a computer monitor to a video camera.	**TP-007**	Pass	*Appendix - 13.7,* Figure 54 *&* Figure 55	None
REQ-021	Preview what the detection system will see, this should reflect the current settings.	**TP-007**	Pass	*Appendix - 13.7,* Figure 57	The appearance of the GUI preview window and preview capture window match.
REQ-025	Change the height of the area of the captured frame which is processed.	**TP-008**	Pass	*Appendix - 13.7,* Figure 58 *&* Figure 59	None
REQ-026	Change the width of the area of the captured frame which is processed.	**TP-009**	Pass	*Appendix - 13.7,* Figure 60 & Figure 61	None

REQ-027	The option used to select the input device should only show detected, compatible devices.	**TP-008**	Pass	*Appendix - 13.7,* Figure 54	Only devices recognised as video devices are displayed in the list.
REQ-028	The inputs should only accept non-zero integer values.	**TP-010**	Pass	N/a	Cannot be expressed using screenshots but the capture height and width inputs have an integer-only filter applied which ignores all non-integer keypresses.
REQ-029	Cameras and webcams should be selectable.	**TP-007**	Pass	*Appendix - 13.7,* Figure 54*,* Figure 55 *&* Figure 56	Video is captured in the same way as when capturing from a screen, the application defaults back to screen capture if the capture device cannot be read from.
REQ-031	The driver should be able to enable detection at any time.	**TP-001**	Pass	*Appendix - 13.7,* Figure 46	The detection system was capable of halting and restarting as required.

REQ-032	The driver should be able to stop the detection system at any time.	**TP-001**	Pass	*Appendix - 13.7,* Figure 53	The detection system halted as expected.
REQ-033	The driver should be able to view the current line speed.	**TP-011**	Pass	*Appendix - 13.7,* Figure 62 *&* Figure 63	The line speed was updated on the GUI as different speed limits were detected.
REQ-034	The driver should be able to view the locomotives current speed.	**TP-001**	Pass	*Appendix - 13.7,* Figure 67	None
REQ-035	The driver should be shown what the camera is currently seeing displayed on the interface.	**TP-001**	Pass	*Appendix - 13.7,* Figure 46, Figure 62 & Figure 63	A preview of what the system was capturing was displayed on the GUI and updated. when new frames had been processed.
REQ-036	The driver should be displayed the bounding boxes of any detections made.	**TP-001**	Pass	*Appendix - 13.7,* Figure 46, Figure 62, Figure 63 & Figure 64	Bounding boxes are displayed around all detections including false positives.
REQ-037		**TP-001**		*Appendix - 13.7,* Figure	The level of certainty is

	The driver should be shown the level of certainty when a detection is made.			46, Figure 62, Figure 63	displayed above the bounding boxes.
REQ-044	The system shall store settings in a file.	**TP-011**	Pass	*Appendix - 13.7,* Figure 65 & Figure 66	Updated settings were saved as expected.

9.2 Sign Detection System

REQUIREMENT	DESCRIPTION	TEST PLAN	PASS/FAIL	EVIDENCE	NOTES
REQ-010	In ideal conditions, the accuracy of the system should be 90%.	**TP-012**	Pass	*Appendix - 13.7,* Figure 68	When evaluating the model against the validation set the mAP50 is 97%.
REQ-022	The system should be able to detect speed signs in a range of different weather conditions.	**TP-013**	Pass	*Appendix - 13.7,* Table 12.	None
REQ-024	The system should be able to detect speed signs regardless of the time of day.	**TP-013**	Pass	*Appendix - 13.7,* Table 12	

9.3 End-to-End Testing

Testing of the software will follow the methodology described within 5.5.1. Examples of all speed signs on the test route can be seen in 13.8. All collected results from the testing process can also be viewed under 13.9, End-to-End Test Results. Whilst carrying out the testing process it was noted that the ambient light levels for dawn and dusk within the simulator were nearly identical, therefore the decision was taken to test the application with the time set to 05:00 and would be categorized as dawn/dusk testing.

	CLEAR	FOG	RAIN	SNOW
DAWN/DUSK	21	21	19	9
DAY	20	18	20	20
NIGHT	12	21	21	10

Table 9 – Signs correctly identified, see full results in 13.7.

	CLEAR	FOG	RAIN	SNOW	AVERAGE
DAWN/DUSK	100%	100%	90.50%	42.88%	83.35%
DAY	95.24%	85.71%	95.24%	95.24%	92.86%
NIGHT	57.14%	100%	100%	47.62%	76.19%
AVERAGE	84%	95%	95%	62%	84.06%

Table 10 – Percentage accuracy of application during testing.

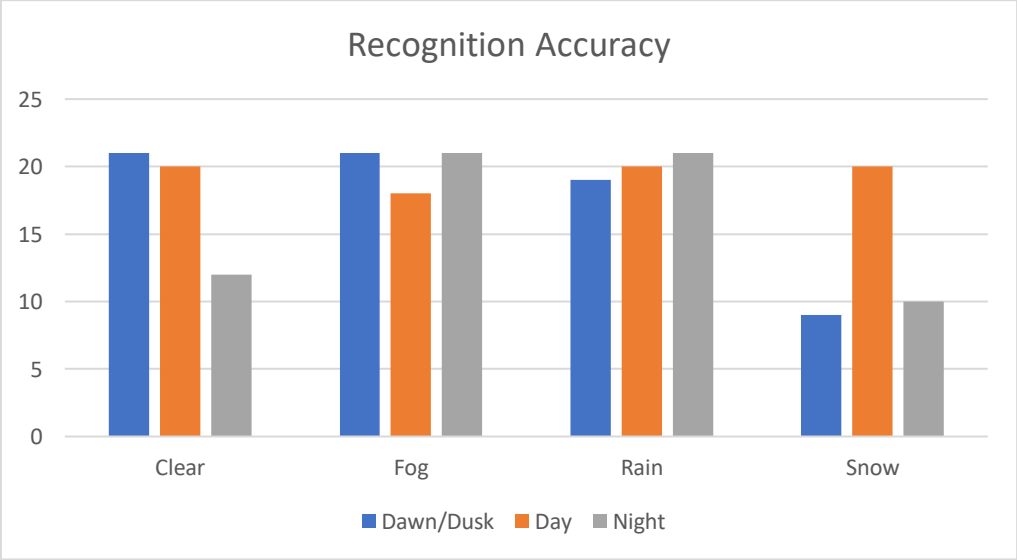

Table 11 – Accuracy across all weather conditions.

9.4 Unit Testing

Due to the nature of the project, it was not possible to unit test every single component integrated however where it was feasible unit tests have been written, which at the time of project completion are all passing as can be seen below in. Mocking is a common technique used within unit testing to emulate the functionality of external components; this was used extensively to emulate the control API. Within the unit tests approximately 82% of the lines of code have been tested.

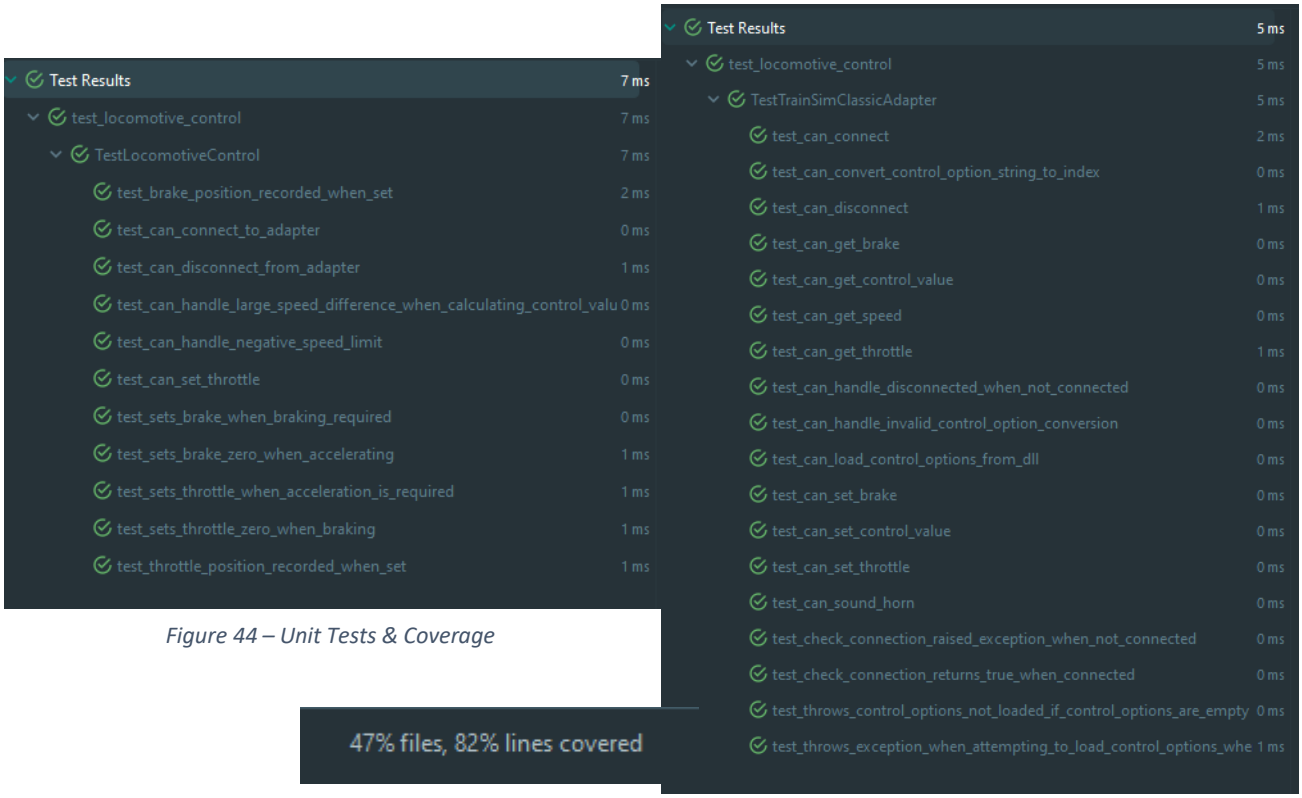

Figure 44 – Unit Tests & Coverage

10 Discussion: Review and Reflections

10.1 Evaluation of the Product

As has been previously discussed, during the testing sprint all application requirements which were initially requested have been tested against. All of these have passed, despite this, some limitations have been noted with the current implementation which we will discuss later in this section. Workarounds and enhancements will also be suggested which could be used to mitigate these limitations.

10.2 Limitations

One of the main limitations which was highlighted during the testing of the applications was the detection range of the detection and identification subnets. This was caused by two factors, the first being a limitation of the simulator and the second a limitation of the detection system. Assumedly as part of the optimization procedure implemented by the developers of the simulator at long distances the level of detail within the scene is reduced, this meant that in many situations at long distances, speed signs looked correct to the eye however when zooming in they were blurry. This

assists with increasing FPS within the game however prevented the system from recognising signs at ranges. One of the methods that could be used to overcome this would be to increase the zoom of the camera used to capture frames.

The second issue which was highlighted during testing was the sub-optimal night performance, this could be seen within the results where the system averaged 75% accuracy, the main cause of this was the images captured were too dark, in future image enhancement techniques could be used such as increasing the contrast of the image to make the sign more visible to the detection system, alternatively a night vision sensor could be used however this would require a different recognition model to be trained to recognise signs due to the difference in colours.

The final limitation which was forced due to the implementation of the screen capture system, the application only supports windows, this is due to the underlying system used by OpenCV to connect to external video inputs such as webcams.

10.3 Evaluation of the Project & Personal Performance

During the development of this project, the plan was followed efficiently and effectively to deliver a fully tested product, during the design and planning stages it was ensured that all modern software design procedures were followed through the reference of SWEBOK. Despite this, a repeat of this project would be undertaken differently. Specifically, the requirements analysis phase, primarily concerning the elicitation of requirements. A more in-depth version of this project would have been created had a real client been involved due to the additional feedback which would have been received.

Another consideration is the creation and implementation of machine learning models. There is a limitation to the design and usage of these as has been seen during the development of the project as it became quickly apparent that designing and writing an accurate object detection model would not be possible during the short space of time allocated to the development of this project. A limitation of personal knowledge was also a contributing factor to this, despite a base understanding of machine learning and neural networks.

With regards to writing the report and documenting the outcomes, more notes should have been taken during the development phase. It was commonplace that during a sprint code would be written continuously without pause therefore some of the issues encountered were lost, these would only be minor issues however it meant that additional detail which could have been added to this section was lost.

The online resources provided were satisfactory and meetings with the assigned tutor were also beneficial and sometimes incredibly insightful, a wealth of knowledge, recommendations and resources were provided enabling enhancements to the project to be made in areas which were not well understood beforehand.

The overall understanding of machine learning techniques has increased during the research and development phases of this project, especially my understanding of the capabilities of object detection algorithms as can be seen during the development of this project. There is still plenty of room for additional understanding into the low level architecture of machine learning algorithms and their development such would enable this project to be developed with a custom built neural network.

Using Git/GitHub to version the application was beneficial to the product as it allowed code from each sprint to be maintained separately and merged into the main branch at the conclusion of the sprint, this also enabled code reviews which enhanced the overall quality. One of the main issues with this type of development was the number of branches involved which regularly became out of sync, this could be remedied by developing a procedure to ensure that all current working branches were rebased each time a push was made to the main or development branches.

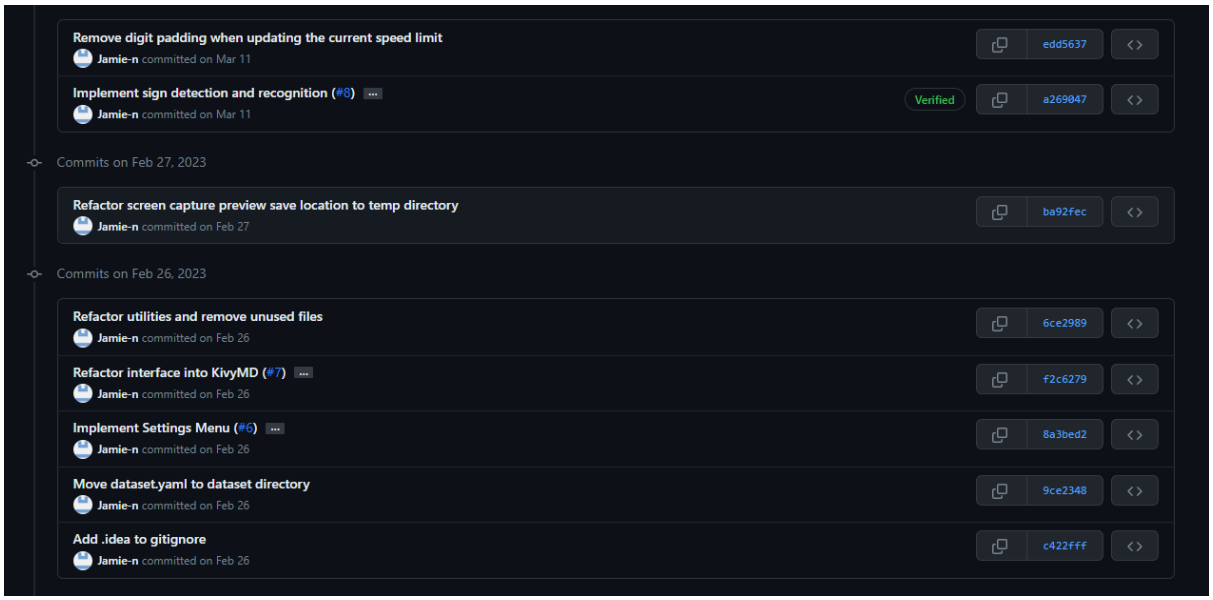

Figure 45 – Snapshot of the GitHub commit history.

11 Summary and Conclusions

The aim of this project was to *'Investigate the efficacy of using object detection techniques to develop a system capable of autonomously controlling a locomotive.'* This has been met whilst following correct software development practices including requirements elicitation, modelling, application design and wireframing to the implementation of a software development methodology. The application produced is provably maintainable and follows Python coding standards. This application shows promise in developing a system which is capable of fully autonomously controlling a locomotive. As has been discussed in previous sections, some shortcomings have been noted such as the night performance. Despite this during the day the application was capable of safely operating on the route.

11.1 Future Plans

There would be several aspects which I would aim to expand upon to extend this project which the current timescale did not allow, as has already been discussed a method of developing a neural network from scratch would be of interest to develop. This also ties into implementing a system which could self-teach based on frames captured during the operation of the system. This would still require a user to manually annotate the images however could enable the system to increase the size of the dataset and thus accuracy.

Another additional feature would be accommodation of locomotive metrics such as length or weight calculation as currently when a speed sign with a greater limit is detected the throttle is increased

immediately whereas in real life a driver should not accelerate until the back of the locomotive has passed the sign, a system where the driver inputs locomotive length would enable this to be added as the distance travelled by the train can easily be calculated based on the speed. The same approach can be taken by creating a system were the driver inputs the weight of the locomotive, this would enable the system to adjust the throttle inputs made to the locomotive. Theoretically, fewer throttle inputs would be required for a lighter locomotive, this would also help enhance passenger comfort, as ideally, all trains should exhibit the same rate of acceleration regardless.

The final aspect of this project would be a real-world implementation on a small scale, possibly a light rail system or model railway to test the real-world performance and application of this system as it has been proven to work in theory.

12 REFERENCES

Adams, C. (2019, August 2). Train driver went more than 100mph over speed limit, investigation finds. *The Independent*. https://www.independent.co.uk/travel/news-and-advice/train-driver-speed-limit-national-rail-kings-cross-a9037006.html

Agile Alliance. (2023). The 12 Principles behind the Agile Manifesto. Agile Essentials: The 12 Principles behind the Agile Manifesto. https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

Agudo, D., Sánchez, A., Vélez, J. F., & Belén Moreno, A. (2016). Real-time railway speed limit sign recognition from video sequences. *2016 International Conference on Systems, Signals and Image Processing (IWSSIP)*, Bratislava, Slovakia, 1-4. https://doi.org/10.1109/IWSSIP.2016.7502716

Allen, G. (n.d). Lecture 4 - Use Case Diagrams [Leaflet]. School of Computing and Engineering. https://brightspace.hud.ac.uk/d2l/le/content/65771/viewContent/328083/View

Barnes, N., Zelinsky, A., & Fletcher, L. S. (2008). Real-Time Speed Sign Detection Using the Radial Symmetry Detector. *IEEE Transactions on Intelligent Transportation Systems, 9*(2), 322-332. https://doi.org/10.1109/TITS.2008.922935

Bass, L., Clements, P., & Kazman, R. (2021). Software Architecture in Practice (4th ed.). Pearson Education, Limited. https://ebookcentral.proquest.com/lib/hud/detail.action?docID=7116234

Biswas, R., Fleyeh, H., & Mostakim, M. (2014). Detection and classification of speed limit traffic signs. *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, Hammamet, Tunisia, 1-6. https://doi.org/10.1109/WCCAIS.2014.6916605

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61-72. https://doi.org/10.1109/2.59

Bourque, P., & Fairley, R. E. (2013). Guide to the Software Engineering Book of Knowledge (SWEBOK V3.0). IEEE Computer Society.

Coleman, D., Ash, D., Lowther, B., & Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer, 27*(8), 44-49. https://doi.org/10.1109/2.303623

Damavandi, Y. B., & Mohammadi, K. (2004). Speed limit traffic sign detection and recognition. *IEEE Conference on Cybernetics and Intelligent Systems*, 2004, Singapore, 2, 797-802. https://doi.org/10.1109/ICCIS.2004.1460690

data.europa.eu. (2020). *Special Eurobarometer 496: Expectations and Concerns from a Connected and Automated Mobility*. data.europa.eu - The official portal for European data. https://data.europa.eu/data/datasets/s2231_92_1_496_eng?locale=en

Dovetail Games. (2009). Train Simulator Classic. Steam. https://store.steampowered.com/app/24010/Train_Simulator_Classic/

Dovetail Games. (2010). Train Simulator: Falmouth Branch Route Add-On. Steam. https://store.steampowered.com/app/24096/Train_Simulator_Falmouth_Branch_Route_AddOn/

Enable Law. (2022). *Driverless Vehicles – Who Is Responsible If They Crash?*. https://www.enablelaw.com/news/expert-opinion/driverless-vehicles-who-is-responsible-if-they-crash/

Gamble, C [CobraOne]. (2014). TS2015 Raildriver Interface. [Online Forum Post] Atomic Systems. https://forums.uktrainsim.com/viewtopic.php?f=361&t=139830

GeeksforGeeks. (2023). Adapter Pattern [Diagram]. https://www.geeksforgeeks.org/adapter-pattern/

Gomes, S. L., Rebouças, E. d. S., Neto, E. C., Papa, J. P., de Albuquerque, V. H. C., Rebouças Filho, P. P., & Tavares , J. M. R. S. (2017). Embedded real-time speed limit sign recognition using image processing and machine learning techniques. *Neural Computing and Applications*, 28, 573–584. https://doi.org/10.1007/s00521-016-2388-3

Greenhalgh, J., & Mirmehdi, M. (2012). Real-Time Detection and Recognition of Road Traffic Signs. *IEEE Transactions on Intelligent Transportation Systems, 13* (4), 1498 - 1506. https://doi.org/10.1109/TITS.2012.2208909

Guerrieri, M., & Parla, G. (2019). Automated Railway Signs Detection. Preliminary Results. *Transport and Telecommunication Journal, 20* (1), 12-21. https://doi.org/10.2478/ttj-2019-0002

Guibert, L., Keryer, G., & Attia, M. (1993). Optical roadsign recognition to improve active safety features. *Optical Engineering and Photonics in Aerospace Sensing*, Orlando, FL, United States, https://doi.org/10.1117/12.160292

HartexLabs. (2023). LabelStudio [Computer Software]. (1.7.2). https://github.com/heartexlabs/label-studio

Huang, G. B., Zhu, Q. Y., & Siew, C. K. (2006). Extreme learning machine: Theory and applications. *Neurocomputing, 70* (1-3), 489-501. https://doi.org/10.1016/j.neucom.2005.12.126

Joshi, M., & Vyas, A. (2014). Comparison of Canny edge detector with Sobel and Prewitt edge detector using different image formats. *International Conference on Emerging Trends of Research in Applied Sciences & Computational Techniques*, 2, https://doi.org/10.17577/IJERTCONV2IS03009

kivy. (n.d). Kivy. GitHub. https://github.com/kivy/kivy

kivymd. (n.d). KivvyMD. GitHub. https://github.com/kivymd/KivyMD

Kundu, S. K., & Mackens, P. (2015). Speed Limit Sign Recognition Using MSER and Artificial Neural Networks. *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Gran Canaria, Spain, 1849-1854. https://doi.org/10.1109/ITSC.2015.300

Kyriakidis, M., Pak, K. T., & Majumdar, A. (2015). Railway Accidents Caused by Human Error: Historic Analysis of UK Railways, 1945 to 2012. *Transportation Research Record: Journal of the Transportation Research Board*, 2476 (1), 126–136. https://doi.org/10.3141/2476-17

Li, G., Hamilton, I. W., Morrisroe, G., & Clarke , T. (2006). Driver detection and recognition of lineside signals and signs at different approach speeds. *Cognition, Technology & Work, 8* (1), 30-40. https://doi.org/https://doi.org/10.1007/s10111-005-0017-5

Li, J., & Wang, Z. (2019). Real-Time Traffic Sign Recognition Based on Efficient CNNs in the Wild. *IEEE Transactions on Intelligent Transportation Systems, 20* (3), 975-984. https://doi.org/10.1109/TITS.2018.2843815

Liu, W., Liu, Y., Yu, H., Yuan, H., & Zhao, H. (2010). Real-Time Speed Limit Sign Detection and Recognition from Image Sequences. *2010 International Conference on Artificial Intelligence and Computational Intelligence*, Sanya, China, 1, 262-267. https://doi.org/10.1109/AICI.2010.62

Losio, R. (2022). *First Open Source Copyright Lawsuit Chal-lenges GitHub Copilot*. InfoQ. https://www.infoq.com/news/2022/11/lawsuit-github-copilot/

lucasg (2021). Dependencies [Computer Software]. https://github.com/lucasg/Dependencies

Lucidchart. (n.d). UML diagrams vs. SysML diagrams. https://www.lucidchart.com/blog/uml-diagrams-vs-sysml-diagrams

madmaze. (2022). Python Tesseract. GitHub. https://github.com/madmaze/pytesseract

Mata-Carballeira, O., del Campo, I., Martínez, V., & Echanobe, J. (2018). Deep Extreme Learning Machines with Auto Encoder for Speed Limit Signs Recognition. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI, USA, 965-972. https://doi.org/10.1109/ITSC.2018.8569428

MasterClass. (2021). Quick Guide to Wireframing: 3 Tips for Creating Wireframes. MasterClass. https://www.masterclass.com/articles/wireframing-guide

NV3 Games. (2009). Trainz Railroad Simulator 2022. Steam. https://store.steampowered.com/app/1784570/Trainz_Railroad_Simulator_2022/

Open Rails. (n.d). Open Rails - Free train simulator. http://openrails.org/

Python Software Foundation. (2023). unittest — Unit testing framework. https://docs.python.org/3/library/unittest.html

Python Software Foundation. (n.d). tkinter — Python interface to Tcl/Tk. Python Foundation. https://docs.python.org/3/library/tkinter.html

rubik. (n.d). Radon. GitHub. https://github.com/rubik/radon

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 779-788. https://doi.org/10.1109/CVPR.2016.91

Regan, H., Wright, R., & Field, A. (2018, December 1). The scientist, the twins and the experiment that geneticists say went too far. *CNN Health*. https://edition.cnn.com/2018/11/30/health/gene-edited-babies-he-jiankui-intl/index.html

Regalado, A. (2019, December 30). He Jiankui faces three years in prison for CRISPR babies. *MIT Technology Review*. https://www.technologyreview.com/2019/12/30/131061/he-jiankui-sentenced-to-three-years-in-prison-for-crispr-babies/

Riverbank Computing. (n.d). What is PyQt?. Riverbank Computing. https://www.riverbankcomputing.com/software/pyqt/

TechPowerUp. (n.d). *NVIDIA GeForce GTX TITAN* X. https://www.techpowerup.com/gpu-specs/geforce-gtx-titan-x.c2632

Tesla. (n.d). *Autopilot and Full Self-Driving Capability*. https://www.tesla.com/support/autopilot

Thomson, J. J. (1976). Killing, Letting Die and the Trolley Problem. The Monist, 59(2), 204-217. https://www.jstor.org/stable/27902416

Torresen, J., Bakke, J. W., & Sekanina, L. (2004). Efficient recognition of speed limit signs. *The 7th International IEEE Conference on Intelligent Transportation Systems*, Washington, WA, USA, 652-656. https://doi.org/10.1109/ITSC.2004.1398978

Train Driver's "Lapse." (1955, September 14). *The Times*, 4.

ultralytics. (n.d). *Ultralytics*. GitHub. https://github.com/ultralytics/ultralytics

Slintel. (2023). Git Market Share. https://www.slintel.com/tech/version-control/git-market-share

Vats, R. (2023). GitHub vs GitLab: Difference Between GitHub and GitLab. UpGrad. https://www.upgrad.com/blog/github-vs-gitlab-difference-between-github-and-gitlab/

Wang, H., Yuan, R., Pan, H., Liu, W., Xing, Z., & Huang, J. (2020). Speed sign recognition in complex scenarios based on deep cascade networks. *IET Intelligent Transport Systems, 14* (6), 628-636. https://doi.org/10.1049/iet-its.2019.0620

Zhang, J., Wang, W., Lu, C., Wang, J., & Sangaiah, A. K. (2020). Lightweight deep network for traffic sign classification. *Annals of Telecommunications, 75* (7-8), 369–379. https://doi.org/10.1007/s12243-019-00731-9

13 APPENDIX

13.1 PROJECT PROPOSAL FORM

13.1.1 Summary of the Problem

One of the main causes of accidents on railways are derailments, this can commonly be caused by over speeding. In some cases, this can have catastrophic consequences. The 2016 Croydon tram disaster was one such example, caused by a tram exceeding the allowed line speed due to a fatigued driver becoming incapacitated. This project aims to prevent this type of issue by developing some software which can integrate into a trains system and automatically control the speed accordingly. Another area of interest in railway transport is 'automatic train operation' (ATO) this currently requires costly modification of the route and additional lineside equipment to enable safe operation however a system which is capable of interpreting speed signs would be a step towards having fully autonomous trains on any route without extensive engineering works.

13.1.2 Project Objective

Investigate the feasibility of using object detection and optical character recognition to increase the safety on British railways by automatically controlling locomotive speed.

1) Complete a comparative analysis of existing open-source object detection and character recognition libraries.
2) Design and model a system capable of meeting the project objective.
3) Development of the interface and software
4) Testing the accuracy, identification speed and overall effectiveness of the system in application.
5) Evaluation of the software and further research which could be completed.

13.1.3 The Product

By the end of this project, I will have developed a system which is capable of reading and interpreting railway speed boards in a simulator and appropriately applying throttle and brakes as required. The system will be able to operate on a test track set up with multiple speed boards, controlling the locomotives speed as demanded. The system will have an interface which will remind the driver of the current line speed, this will also give the driver the opportunity to dismiss the identified speed sign in the case of a false positive. The aim will be to use a rail simulator and it is API to poll speed, throttle and braking percentage and apply throttle and brakes as required. This emulates a 1 integrated system in the real locomotive which would be capable of reading the same data.

13.1.4 The Clients

This is something which could have potential global application, the ability to implement automatic train operation on lines which have not yet received costly modification work to enable this type of operation. As this project focusses on UK railway signage, network rail would be the main potential customer alongside many other UK rail operators such as LNER, Northern, TransPennine or Thameslink to name a few.

13.1.5 Activities to Solve the Problem

During this project there are several steps which will be taken to solve this problem as follows:

- Literature Review of existing research/technologies.
- Research into machine learning & python development environments (Anaconda, Jupyter)
- Investigation of open-source object recognition libraries • Investigate open-source optical character recognition libraries.
- Curate a sizeable, annotated training and testing datasets.
- Train a model capable of recognising speed boards to a high level of accuracy.
- Analyse the accuracy of the model during testing based on different environments.
- Wireframing and developing an intuitive interface.
- Testing the model and software in a simulated environment to establish its potential effectiveness for real life applications.
- Evaluation of the finished software and exploration of future expansion of the project are.

13.1.6 Time Plan

Throughout the project there are several predefined deadlines as dictated by the module these are as follows:

- Project Proposal: 27th October 2022
- Literature Review: 27th December 2022
- Poster Session: Consolidation Week January 2023
- Final Report Deadline: 2nd May 2023
- 30 Minute Presentation: 19th May 2023

13.2 ETHICAL REVIEW FORM

PROJECT ETHICAL REVIEW FORM

Applicable for all research, masters and undergraduate projects

Project Title:	Investigate the feasibility of using object detection and optical character recognition to increase the safety on British railways by automatically controlling locomotive speed.
Student:	U1854720 – James Neighbours
Course/Programme:	Software Engineering BSc
Department:	School of Computing and Engineering
Supervisor:	Colin Venters
Project Start Date:	01/09/2022

ETHICAL REVIEW CHECKLIST

	Yes	No
1. Are there problems with any participant's right to remain anonymous?	☐	☒
2. Could a conflict of interest arise between a collaborating partner or funding source and the potential outcomes of the research, e.g. due to the need for confidentiality?	☐	☒
3. Will financial inducements be offered?	☐	☒
4. Will deception of participants be necessary during the research?	☐	☒
5. Does the research involve experimentation on any of the following?		
(i) animals?	☐	☒
(ii) animal tissues?	☐	☒
(iii) human tissues (including blood, fluid, skin, cell lines)?	☐	☒
6. Does the research involve participants who may be particularly vulnerable, e.g. children or adults with severe learning disabilities?	☐	☒
7. Could the research induce psychological stress or anxiety for the participants beyond that encountered in normal life?	☐	☒
8. Is it likely that the research will put any of the following at risk:		
(i) living creatures?	☐	☒
(ii) stakeholders (disregarding health and safety, which is covered by Q9)?	☐	☒

(iii) the environment? ☐ ☒

(iv) the economy? ☐ ☒

9. Having completed a health and safety risk assessment form and taken all reasonable practicable steps to minimise risk from the hazards identified, are the residual risks acceptable (Please attach a risk assessment form – available at the end of this document) ☒ ☐

STATEMENT OF ETHICAL ISSUES AND ACTIONS

If the answer to any of the questions above is yes, or there are any other ethical issues that arise that are not covered by the checklist, then please give a summary of the ethical issues and the action that will be taken to address these in the box below. If you believe there to be no ethical issues, please enter "NONE".

NONE

STATEMENT BY THE STUDENT

I believe that the information I have given in this form on ethical issues is correct.

Signature: J Neighbours Date: 27/11/2022

AFFIRMATION BY THE SUPERVISOR

I have read this Ethical Review Checklist and I can confirm that, to the best of my understanding, the information presented by the student is correct and appropriate to allow an informed judgement on whether further ethical approval is required.

Signature: C Venters Date: 28/11/2022

SUPERVISOR RECOMMENDATION ON THE PROJECT'S ETHICAL STATUS

Having satisfied myself of the accuracy of the project ethical statement, I believe that the appropriate action is:

The project proceeds in its present form	X
The project proposal needs further assessment by an Ethical Review Panel. The Supervisor will pass the form to the Ethical Review Panel Leader for consideration.	

13.3 The Agile Manifesto

Listed below are the 12 agile principles as outlined by Agile Alliance (2023).

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Businesspeople and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

13.4 Maintainability Index

It is key during software development to ensure that the systems developed are maintainable. This ensures their longevity and lowers the long-term maintenance and running costs of the system and can also optimize the performance. There are many different metrics to measure how maintainable a system is such as the tightness of coupling of the modules or the number of lines of code the project contains. These metrics only examine a single aspect and can sometimes be misleading therefore combined metrics have become more popular which look at a range of factors and present the user with an overall score. First proposed by Coleman et al (1994), a mathematical formula based on a range of metrics was proposed as a method of establishing how maintainable a system is and has been widely used since. The formula will score the measured application from 0-100, the higher the score the better the maintainability.

13.5 Model Training Python Script

```
from ultralytics import YOLO

model = YOLO("../pretrained/yolov8n.pt")

results = model.train(data="dataset.yaml", epochs=150, patience=10, cache="ram",
pretrained=True)
results = model.val()
success = model.export()
```

13.6 Model Validation Python Script

```
from ultralytics import YOLO

model = YOLO(r"../best.pt")

results = model.val()
```

13.7 Testing Evidence

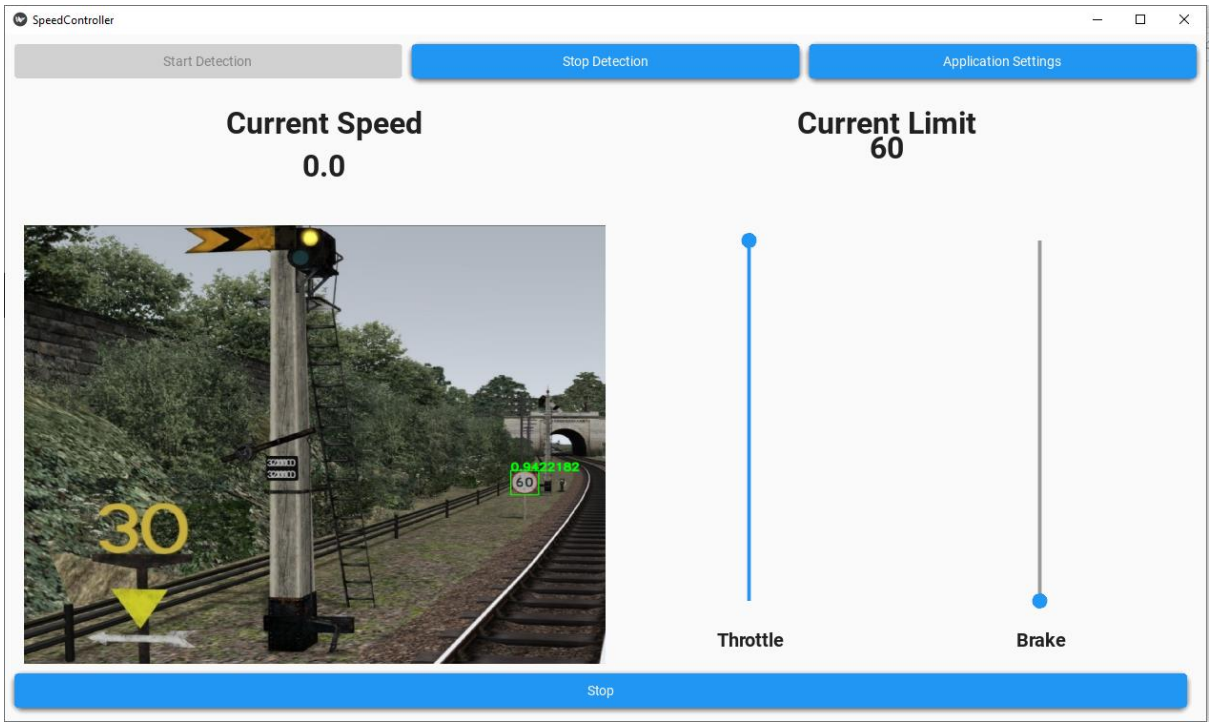

Figure 46 – REQ-001, REQ-004, REQ-008, REQ-030

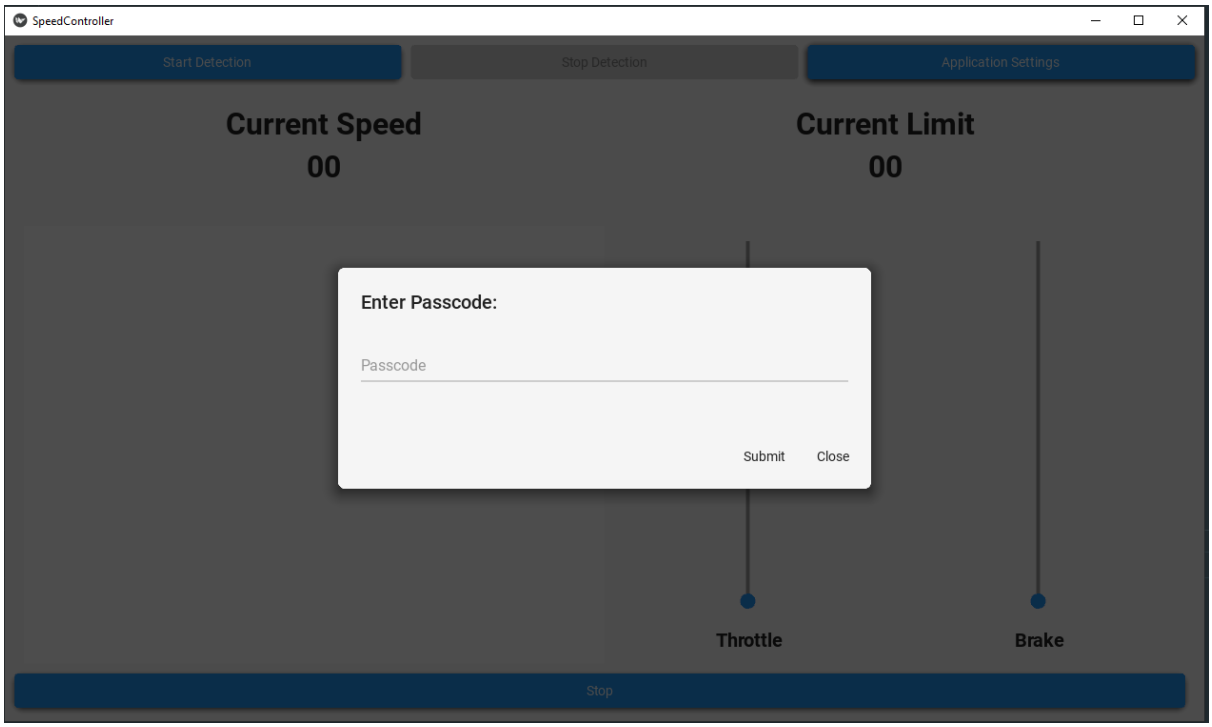

Figure 47 – REQ-040 – Password Window

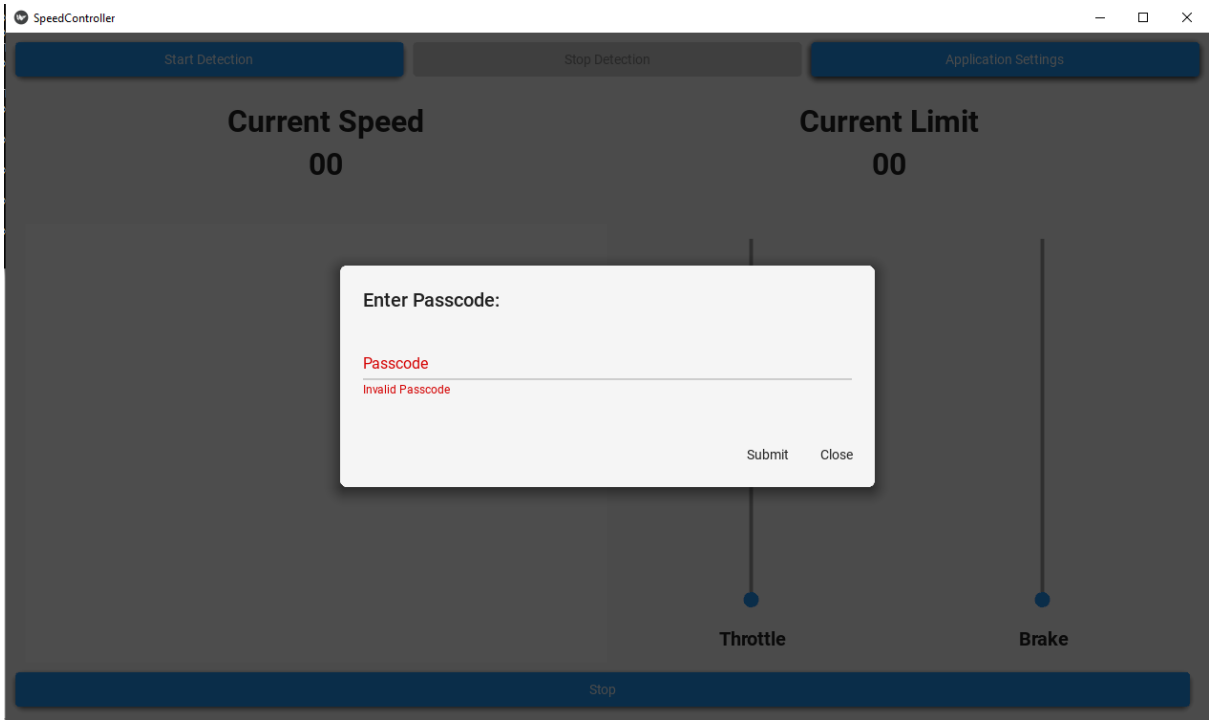

Figure 48 – REQ-041 – Invalid Passcode

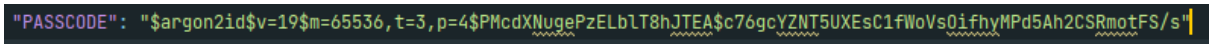

Figure 49 – REQ-043 – Password Hash

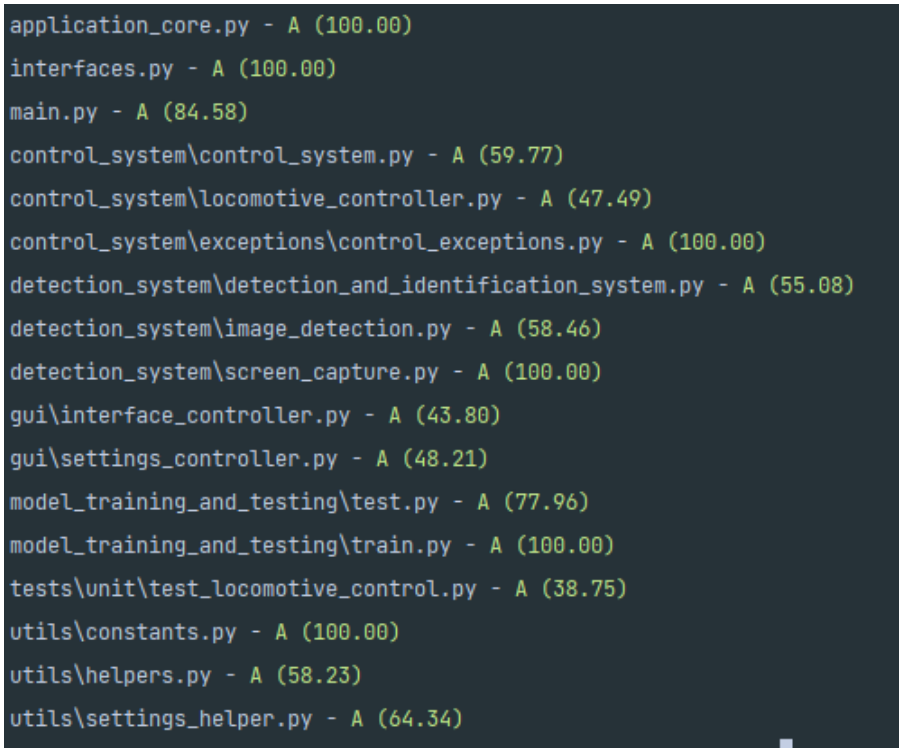

Figure 50 – REQ-059, Maintainability Index, output produced using Radon (rubik, n.d)

```
Time of Last Detection Cycle:  193.045 ms
Time of Last Detection Cycle:  195.044 ms
Time of Last Detection Cycle:  196.044 ms
Time of Last Detection Cycle:  192.042 ms
Time of Last Detection Cycle:  197.046 ms
Time of Last Detection Cycle:  189.043 ms
Time of Last Detection Cycle:  198.044 ms
Time of Last Detection Cycle:  196.045 ms
Time of Last Detection Cycle:  200.045 ms
```

Figure 51 – REQ-006, Application Debug Output

```
174 blocks (classes, functions, methods) analyzed.
Average complexity: A (1.367816091954023)
```

Figure 52 – REQ-007, Cyclomatic Complexity

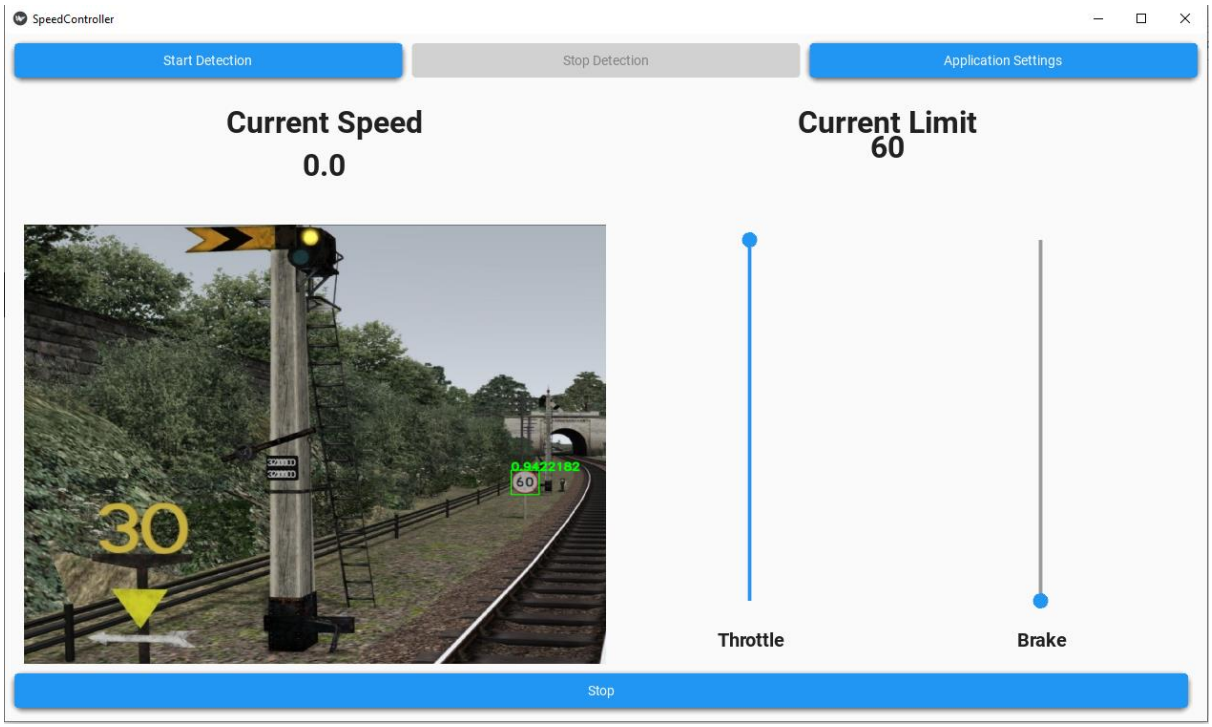

Figure 53 – REQ-017

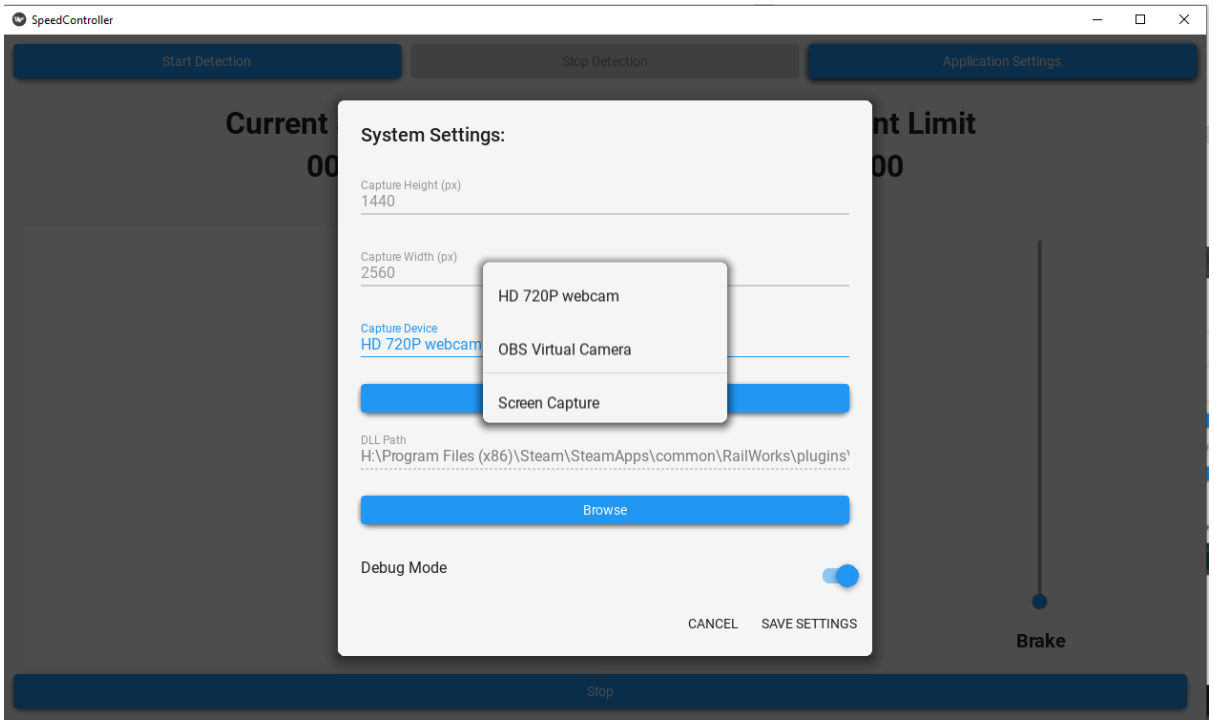

Figure 54 – REQ-020

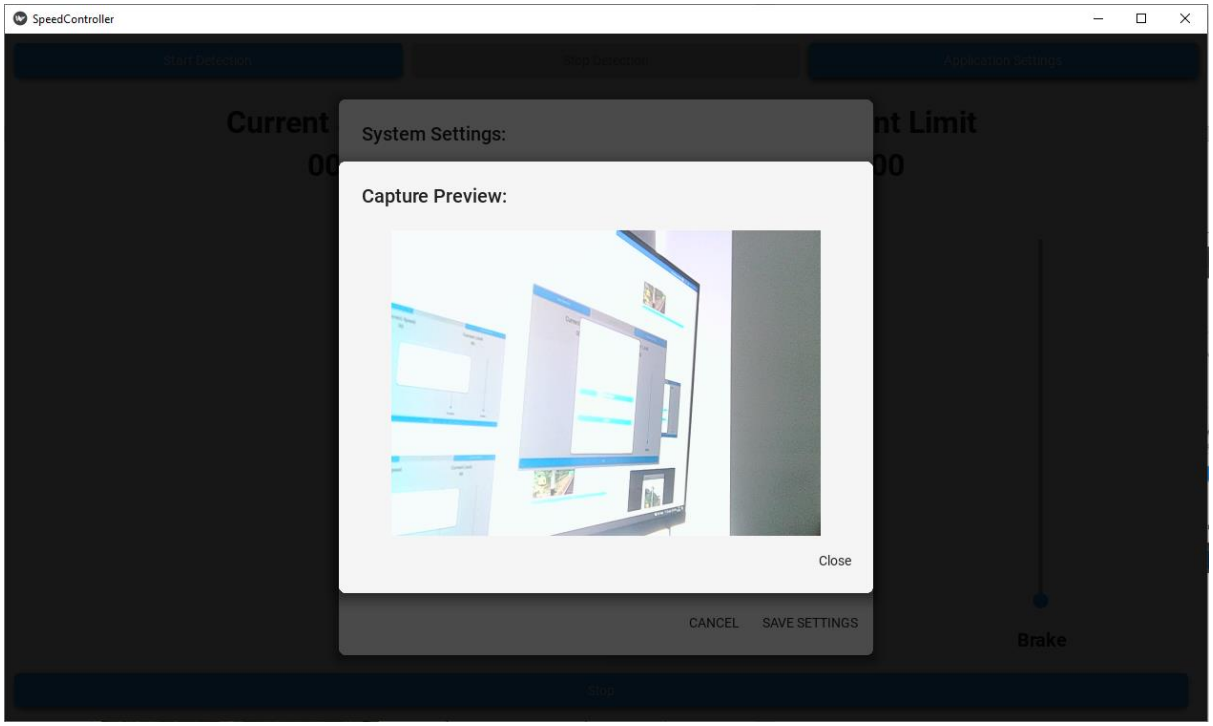

Figure 55 – REQ-020

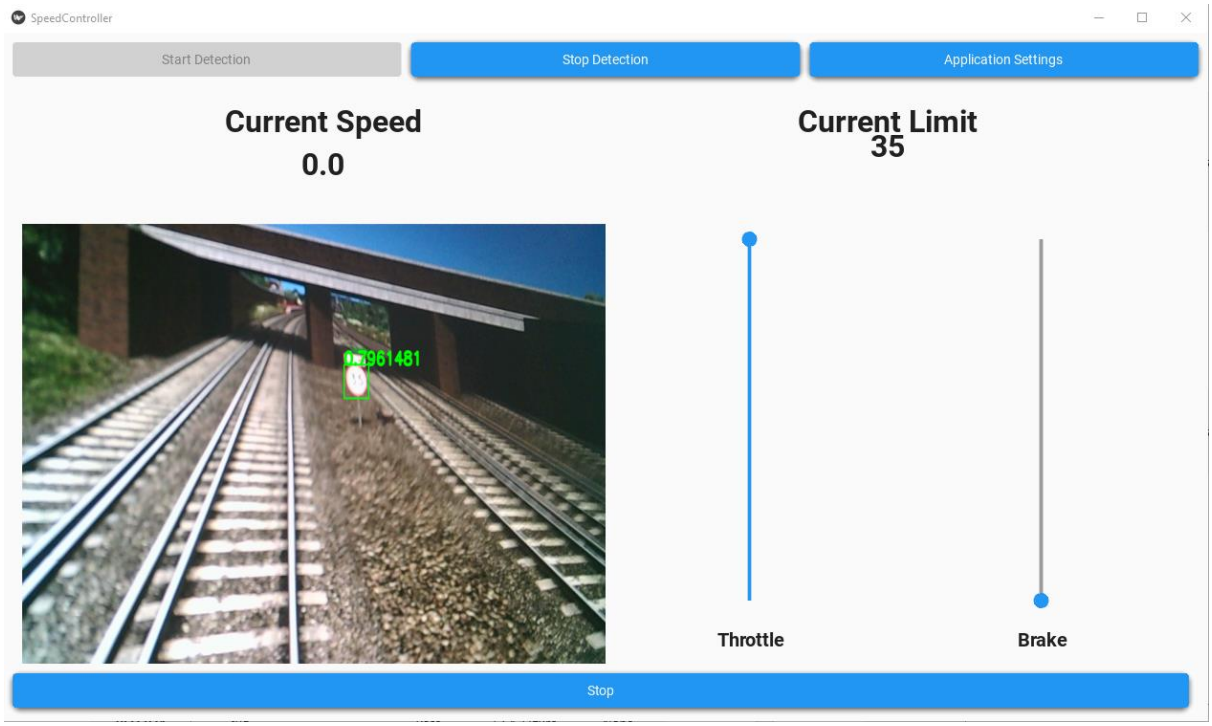

Figure 56 – REQ-029

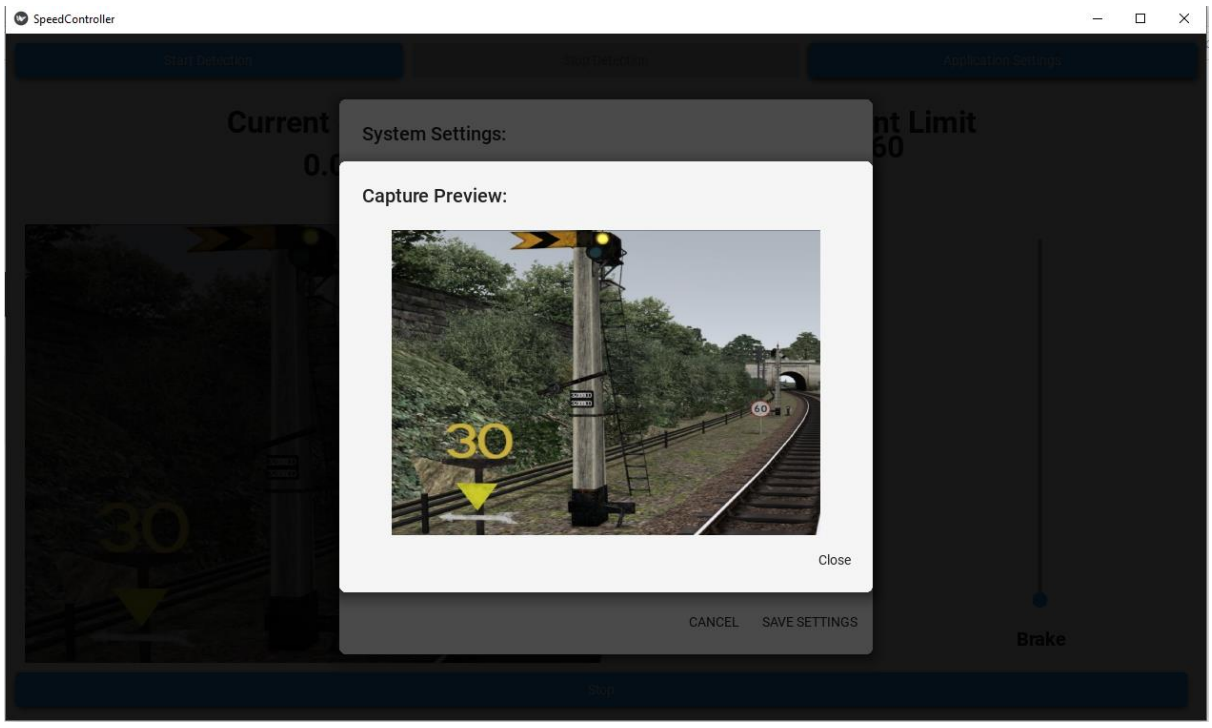

Figure 57 – REQ-021

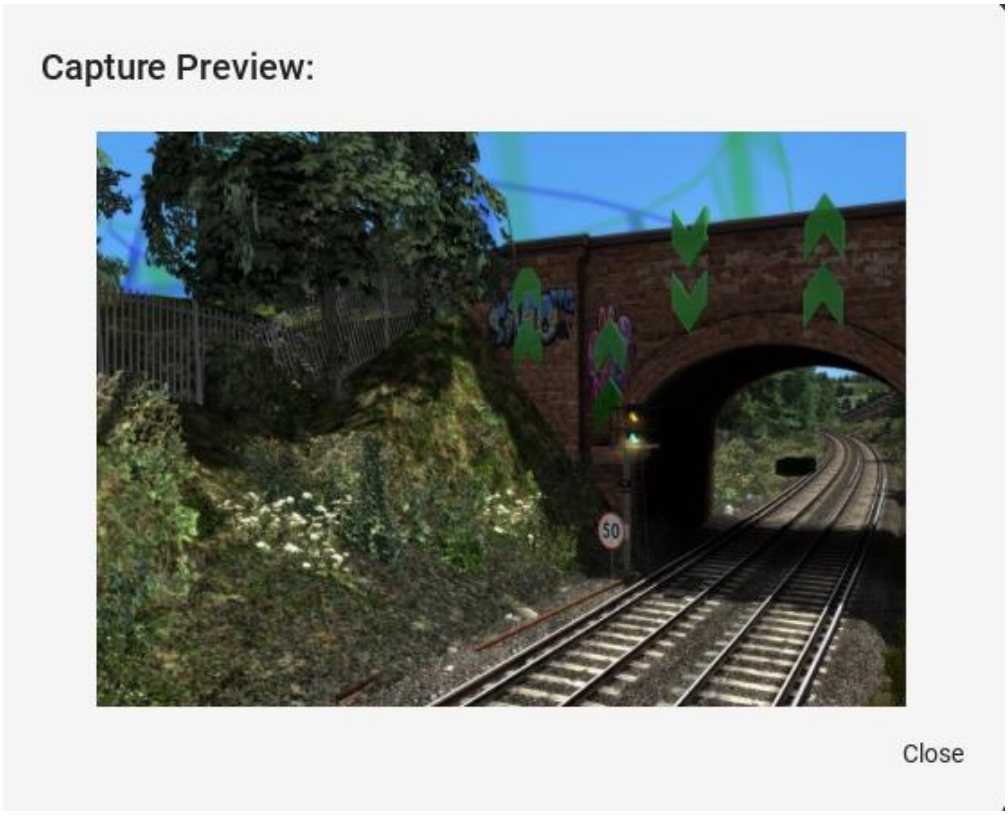

Figure 58 – REQ-025, Captured at 1080px Height.

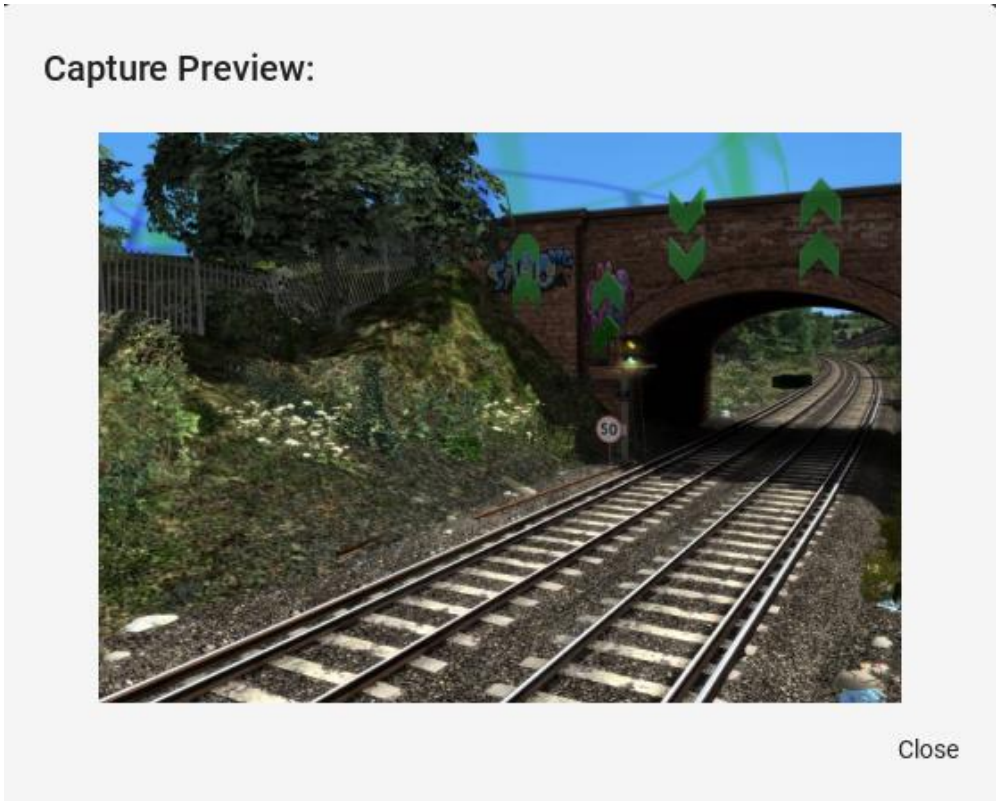

Figure 59 – REQ–025, Captured at 1440px Height.

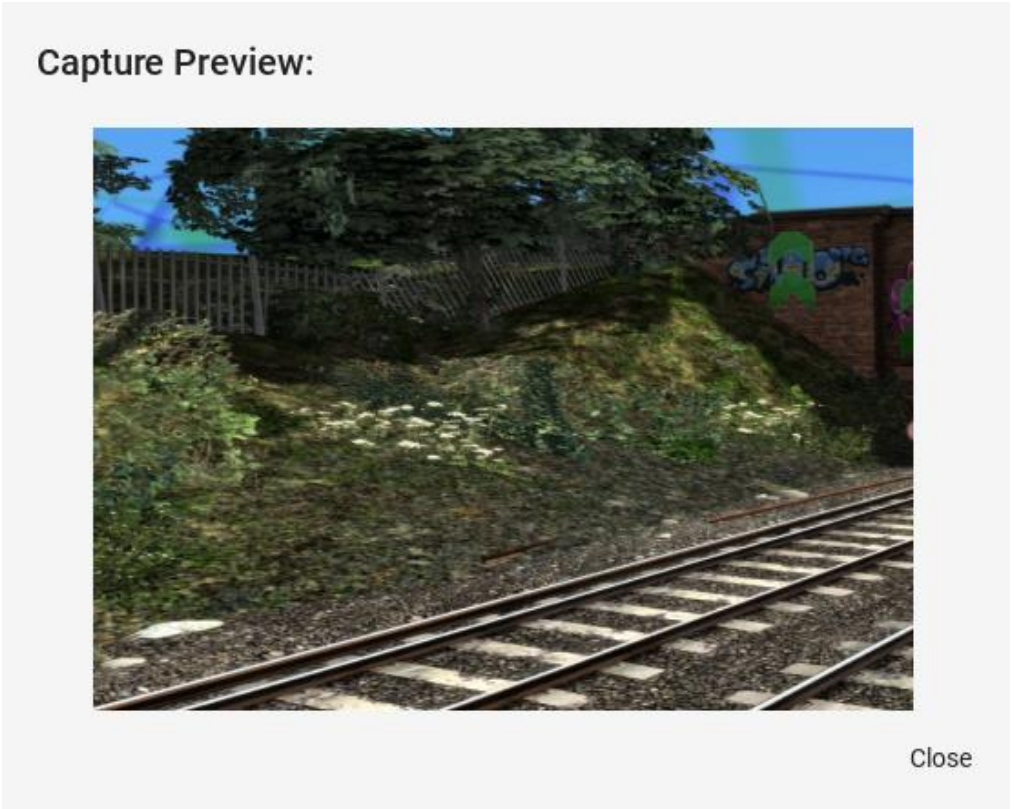

Figure 60 – REQ-026, Captured at 1200px Width.

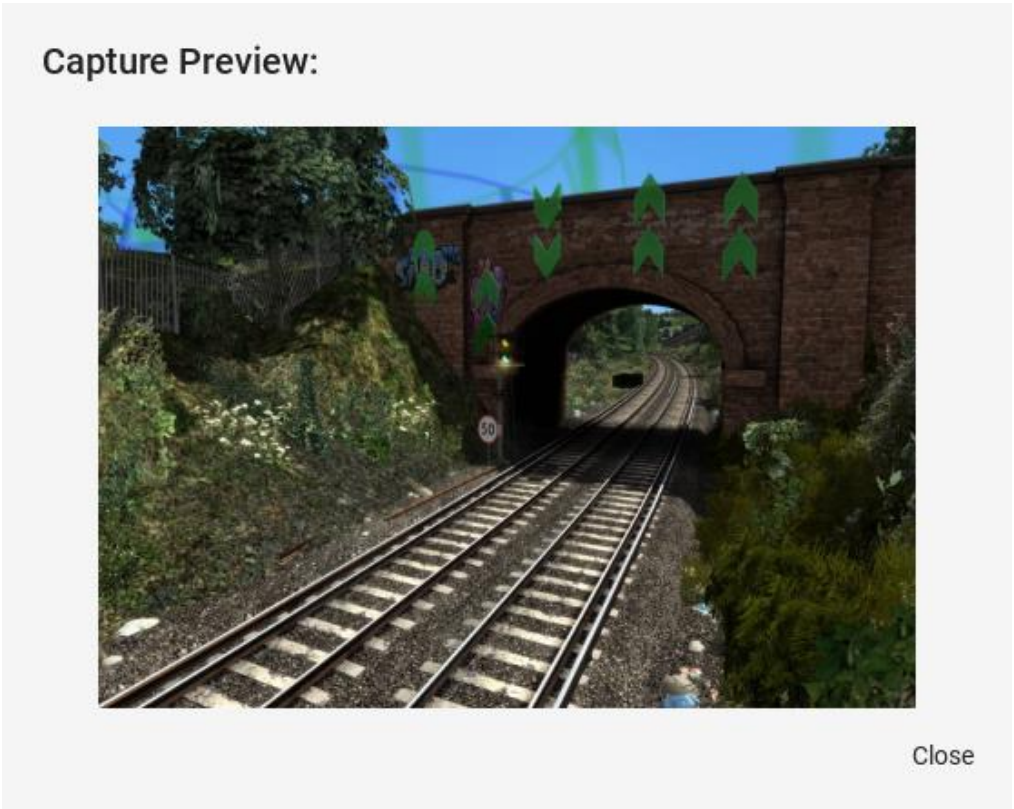

Figure 61 – REQ-026, Captured at 2560px Width.

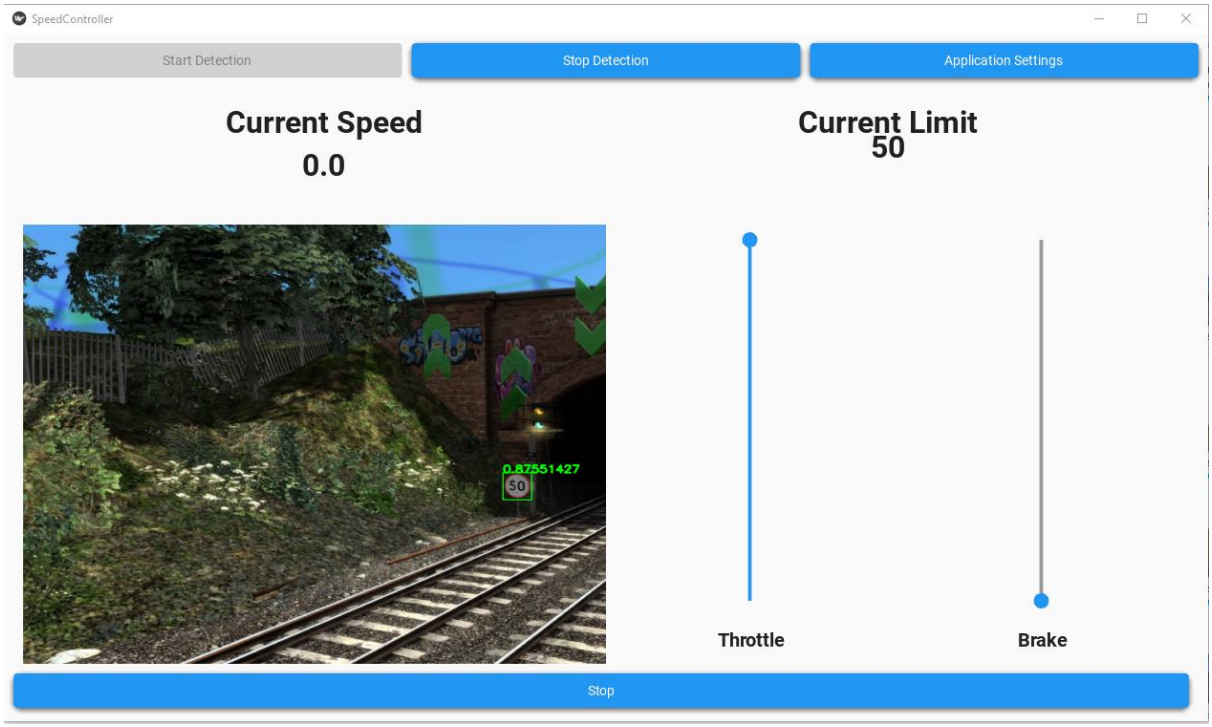

Figure 62 – REQ-033

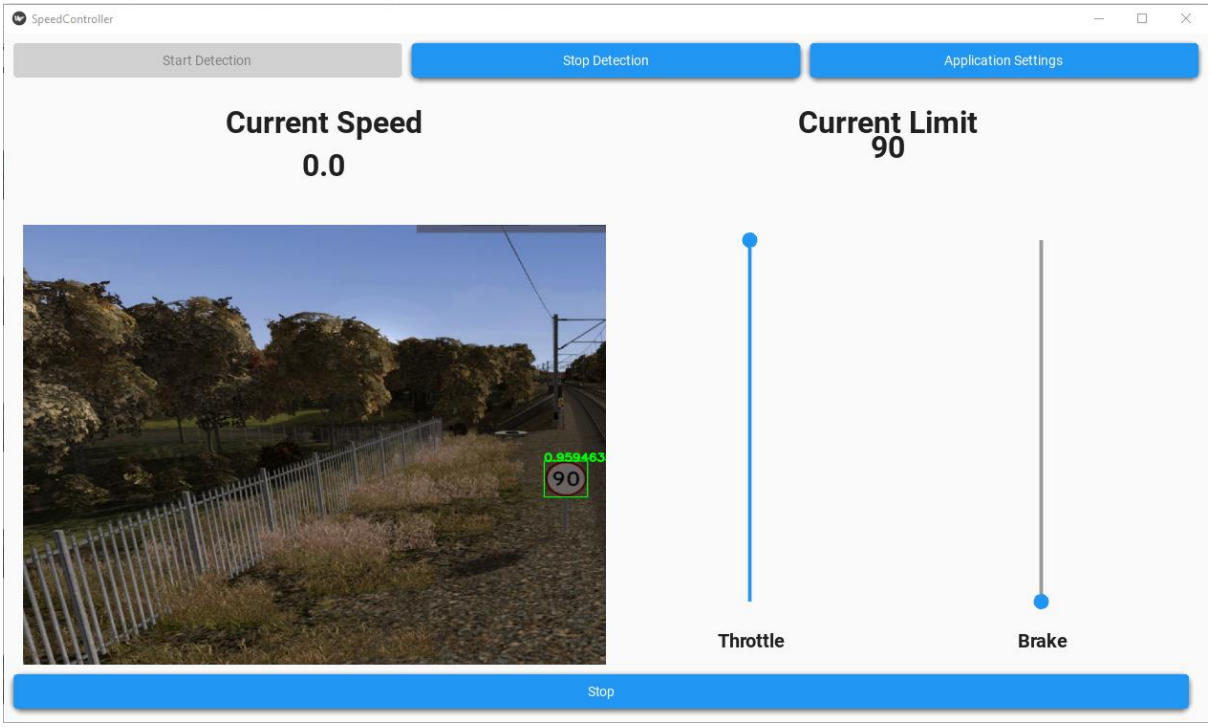

Figure 63 – REQ-033

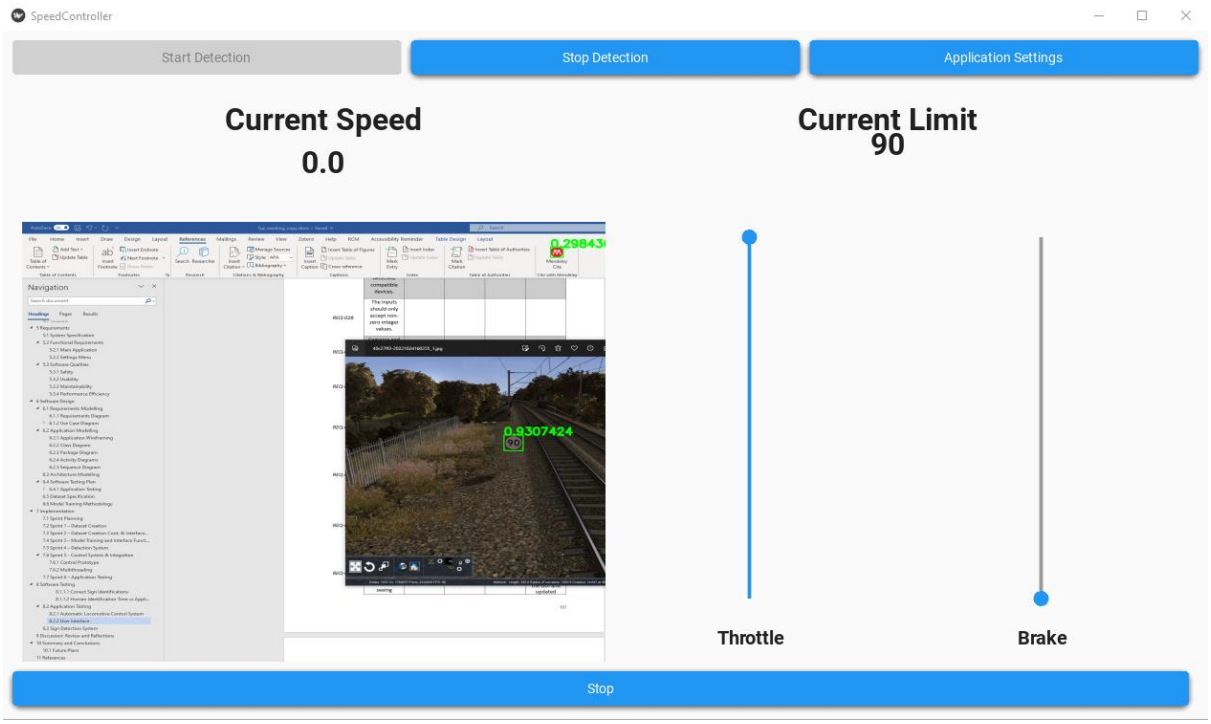

Figure 64 – REQ-036

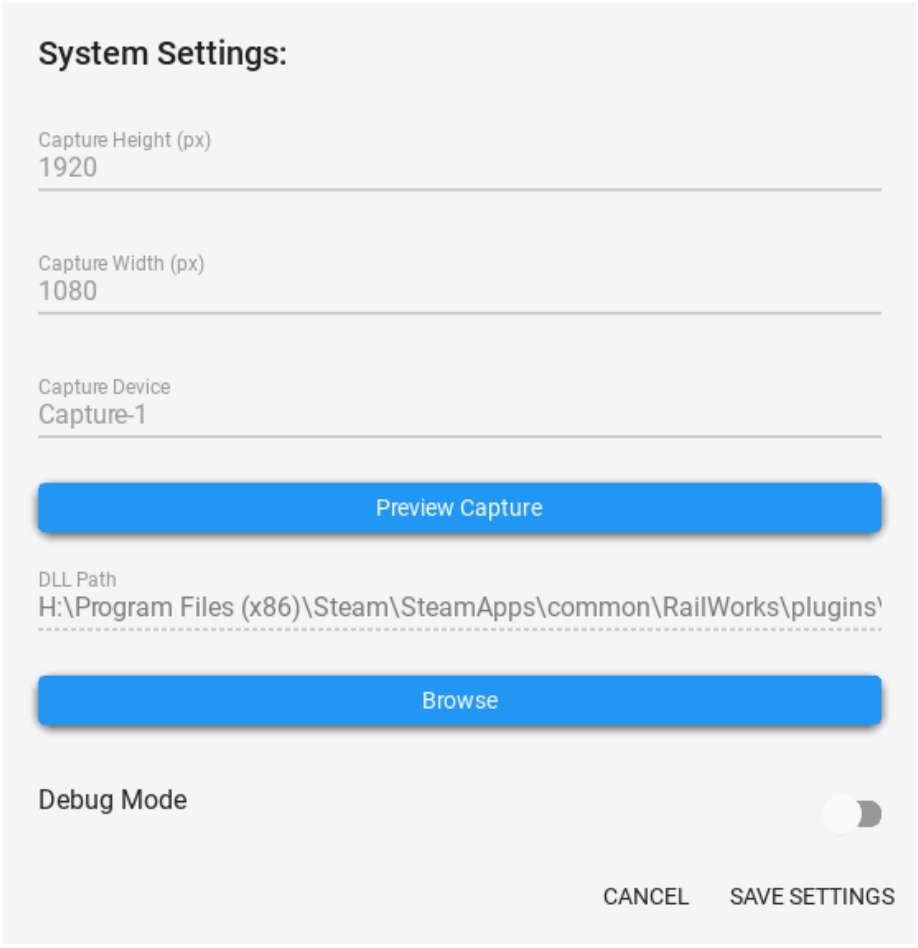

Figure 65 – REQ-044

```
{
  "CAPTURE_HEIGHT": "1920",
  "CAPTURE_WIDTH": "1080",
  "DLL_PATH": "H:\\Program Files (x86)\\Steam\\SteamApps\\common\\RailWorks\\plugins\\RailDriver64.dll",
  "CAPTURE_DEVICE": "Capture-1",
  "IS_DEBUG": false
}
```


Figure 66 – REQ-044

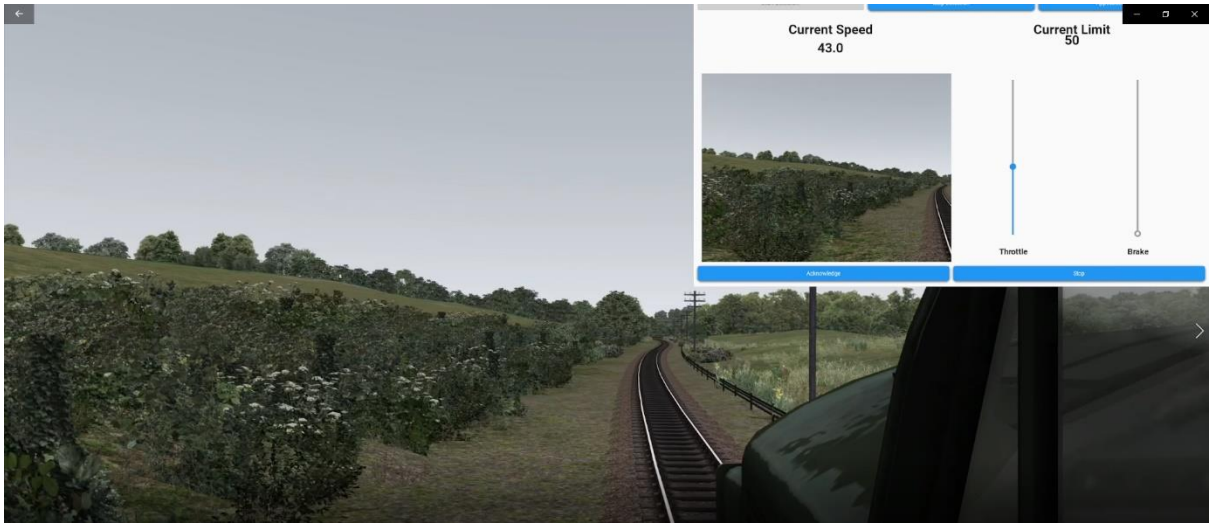

Figure 67 – REQ-034

WEATHER TYPE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	TOTAL
CLEAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	20/22
FOG	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X	X	X	X		18/22
RAIN	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X		20/22
SNOW	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	20/22

Table 12 – REQ-022, Detections in different weather conditions

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	150	181	0.948	0.945	0.976	0.823

Figure 68 – REQ-010 – Validating the model.

13.8 Speed Sign Reference

SIGN NUMBER	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
SPEED LIMIT (MPH)	20	30	60	30	40	50	40	60	30	25	50	90	40	10	15	25	60	20	40	30	15

Figure 69 – Speed Sign 1

Figure 70 – Speed Sign 2

Figure 71 – Speed Sign 3

Figure 72 – Speed Sign 4

Figure 73 – Speed Sign 5

Figure 74 – Speed Sign 6

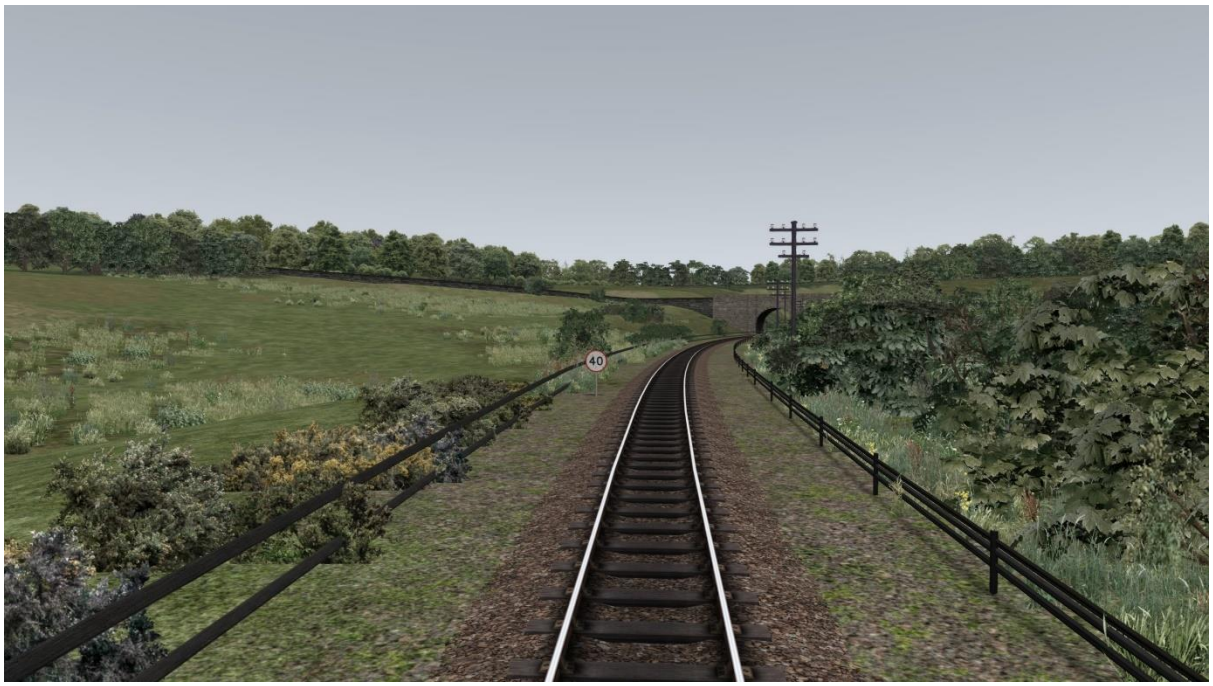

Figure 75 – Speed Sign 7

Figure 76 – Speed Sign 8

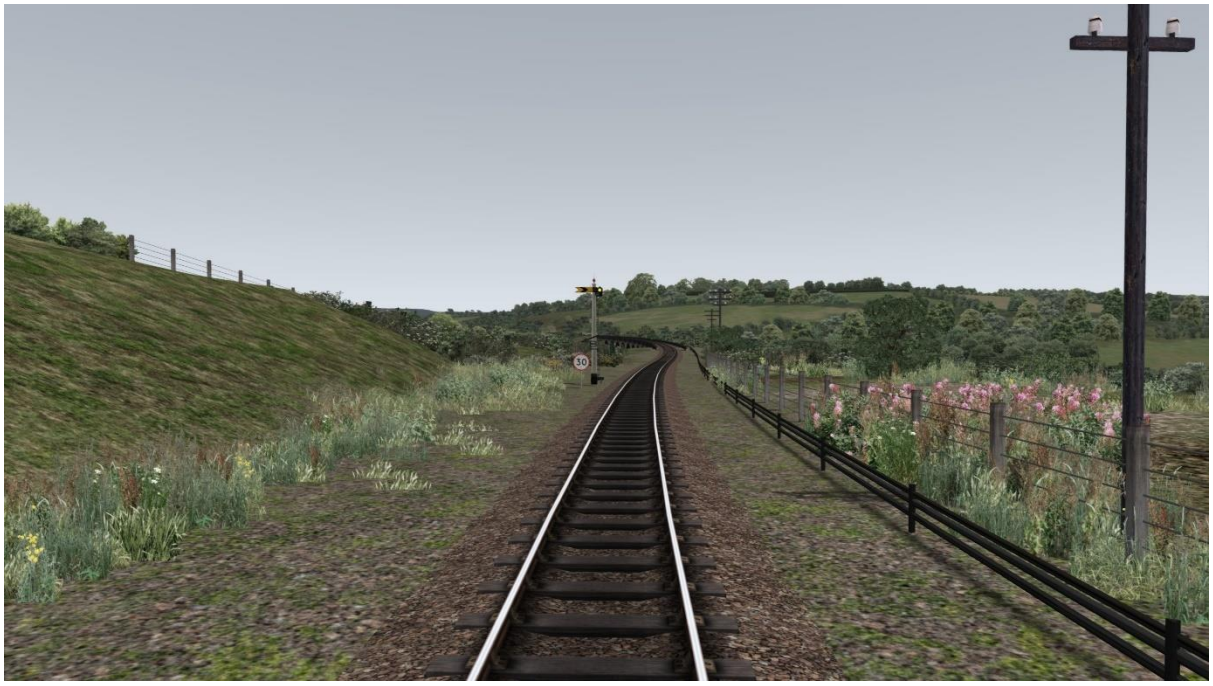

Figure 77 – Speed Sign 9

Figure 78 – Speed Sign 10

Figure 79 – Speed Sign 11

Figure 80 – Speed Sign 12

Figure 81 – Speed Sign 13

Figure 82 – Speed Sign 14

Figure 83 – Speed Sign 15

Figure 84 – Speed Sign 16

Figure 85 – Speed Sign 17

Figure 86 – Speed Sign 18

Figure 87 – Speed Sign 19

Figure 88 – Speed Sign 20

Figure 89 – Speed Sign 21

13.9 End-to-End Test Results

Key: D – Signs missed due to derailments, N – Sign not identified, X – Sign identified.

13.9.1.1 Correct Sign Identifications – Dawn/Dusk

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	TOTAL	ACCURACY
CLEAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	21/21	100%
FOG	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	21/21	100%
RAIN	X	X	X	X	X	X	X	X	X	X	X	X	X	N	N	X	X	X	X	X	X	19/21	90.50%
SNOW	X	X	X	N	N	X	N	N	X	X	X	X	N	X	D	D	D	D	D	D	D	9/21	42.86%

13.9.1.2 Correct Sign Identifications - Day

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	TOTAL	ACCURACY
CLEAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	20/21	95.24%
FOG	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X	X	X	X		18/21	85.71%
RAIN	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X		20/21	95.24%
SNOW	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	20/21	95.24%

13.9.1.3 Correct Sign Identifications – Night

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	TOTAL	ACCURACY
CLEAR	X	X	X	X	X	X	X	X	X	X	X	X	N	N	D	D	D	D	D	D	d	12/21	57.14%
FOG	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	21/21	100%
RAIN	X	X	X	X	X	X	X	X	X	X	X	X	N	X	X	X	X	X	X	X	X	21/22	100%
SNOW	X	X	X	N	X	X	N	X	N	X	X	X	N	X	D	D	D	D	D	D	D	10/22	47.62%