Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Interim Report 2017

Project Title:	**Distributed Road Traffic Speed Monitoring**
Student:	**Jeremy Chan**
CID:	**00818433**
Course:	**4T**
Project Supervisor:	**Dr. Ed Stott**
Second Marker:	

Table of Contents

I. Introduction

The number of people speeding on the roads in the UK have actually decreased over the past 2 years [1] as a result of there being more speed cameras being installed across the country [2]. From [3], it is clear that 'the higher the speed, the greater the probability of a crash'. Hence, it is in the best interest of the authorities to monitor speeding convictions using average speed check cameras.

However, most of the average speed check cameras across the UK are installed in busy trunk routes, i.e. motorways. They use automatic license plate recognition technologies to track the entry and exit times. This data is not accessible for the general public and is kept for two years [4]. This project aims to remedy that, given that many countries have supported the idea of using license plate readers and that there should not be any restrictions surrounding who and when can license plates be collected [5].

There are also very little amounts of speed cameras in rural and local roads. By using a decentralised network that can find out peers in its vicinity, anyone can download, compile, and use this project as a pseudo-speed camera in their local area, removing the need for expensive cameras and reliance on a closed database of license plates that only the police have access to. Anyone caught speeding will have their license plate posted onto a social network. Given that there is a set procedure that police use in issuing speeding tickets, the end goal is just to have the evidence on the social networks and not to actually ticket the offender.

II. Project Specification

From the project description, the goals of the project can be separated into the following:

A. *Project goals*

- Implement a number plate recognition system using existing computer vision algorithms on a low-cost, readily available hardware platform.
- Set up a peer-to-peer network to share vehicle passing times and detect violations without the need for a central server.
- Publish photo evidence of any violations

B. *Advanced project goals*

- Use the changes in the number plate geometry as the vehicle passes to detect the instantaneous speed of a vehicle. This provides a stand-alone mode that will aid adoption in areas where there isn't already an established network.
- Implement automatic peer discovery so that each device can find its neighbours and calculate the minimum legal transit time between them using a public mapping database.
- Add an encryption layer so that a hacker or rogue peer cannot use the network to track the movements of law-abiding vehicles.
- Package the system so that it can be easily installed in a home by an inexperienced user.

C. *Clarifications*

After discussions with Dr. Stott, the following clarifications were made:

1. The system should target license plates and deployment in the United Kingdom, and license plates of the UK format [6]. Custom license plates will not be in the scope of the project for simplicity.
2. The number plate recognition system should be targeted at an off the shelf package, so there should be minimal setup and calibration done. This also means anyone, with the right equipment, should be able to download and compile the system if they have existing hardware.
3. The low-cost, readily available hardware platform will be a Raspberry Pi (RPi), with a camera attached to it. Using an RPi combines the best of cost (~£40 at time of writing), power (quad core CPU [7]), flexibility (camera can be any USB webcam or RPi's official cameras), and support (development work on the RPi is extensive and there are ample tutorials/information online).
4. The peer to peer network should ideally be fully decentralised, so the system should be able to find peers without the help of a central server.
5. Publishing photo evidence will most likely be done onto a social network.
6. The public mapping database will be one accessible to most people – Google Maps API. There is a speed limit API but it is only open to premium users (https://developers.google.com/maps/documentation/roads/speed-limits).
7. A hacker or rogue peer should not be able to extract license plates from the system remotely, nor should they be able to spoof detect license plates as another user to avoid framing an innocent driver.
8. If possible, there should be a whitelist of emergency vehicles in the case of there being an emergency vehicle over the speed limit.
9. The final package should be a software package uploaded onto GitHub, with clear instructions on how to compile and run the program with examples. As this project involves testing on hardware as well, a list of recommended hardware should also be provided (after successful testing), so the project can be easily replicated in the future.

III. Related Work

Research into already existing products was done on the three main goals of the project, to judge the market feasibility of the project, and to prevent overlap with existing work where possible.

A. *License plate recognition*

1) *Hardware*

There are many license plate recognition systems available on the market, with many of them being used in speed cameras around the country [8]. Traffic cameras can generally be separated into radar based and optical based systems, with radar based cameras checking the instantaneous speed of the vehicle as it moves past, and optical based systems checking the average speed of the vehicle as it passes between two points. There has been a steady increase in average speed check cameras over the years [2], mostly along motorways but also by local councils [9]. These average speed check cameras often include multiple enticing features like 24/7 operation and 4G connectivity, as such in Jenoptik's VECTOR cameras [10].

2) *Software*

Most of the license plate recognition software available is proprietary and closed source. Some area only available for commercial use. These systems generally have features such as video stream processing, ability to detect plates from multiple points of view, and cloud services to take in a video stream and output detected license plates along with their timestamps [11]–[13]. However, the price of the commercial systems (if available for purchase) are definitely out of budget for this project, as shown in Table 1. ARES (http://platesmart.com/) did not respond to an enquiry regarding cost of usage, so is not shown.

As the project specification specifies that 'existing computer vision algorithms' should be used, the choice was made to use the consumer version of OpenALPR. The main reasons for this was that it was open source, and

very feature rich. Additionally, a choice was made to instead replicate the commercial features of OpenALPR using computer vision processing libraries instead of ignoring the extra features or paying for them as they were standard vision processing techniques, e.g. background subtraction, motion detection.

Table 1: Prices of different license plate detection systems

System	Price (one-off)	Price (per month)
DTK LDR SDK	190 – 1430 EUR	N/A
OpenALPR	1000 USD	50 USD

B. *Peer to peer network*

Peer to peer networks (P2P) have existed for a long time and there are numerous implementations available for use on the Internet. The main uses for peer to peer networks are web, messaging, and file sharing.

Bitmessage (https://bitmessage.org) is used for encrypted messaging, and is decentralised. It also uses strong authentication to prevent spoofing of the sender of a message [14]; however, it removes information about the sender and receiver. It is therefore not suitable for this project as this system needs to know information about the sender to accurately work out an average speed.

Telehash (https://github.com/telehash/telehash.github.io) is open source, fully end-to-end encrypted, enforces strict privacy rules, and cross platform. It also supports JSON message sending with unique sender and receiver ID's. Unfortunately, the development is focused around using Node.JS and not Python or C++ (the Python binding repository is still empty at time of writing). Therefore, it is not suitable as it would take too much time to implement.

C. *Photo evidence publication*

There are official API's for most social networks on uploading images to their respective social networks so there is no need to reinvent the wheel, no is there a need to use a third party service to upload images.

D. *Overall Feasibility*

Overall, the project is very feasible as there has been a lot of work in all three areas, proving that the ideas in the project are not a dead end. Moreover, there is a free and open source implementation for the license plate recognition system which is immensely useful in providing a solid head start in license plate detection, a primary goal of the project. As most advanced features are behind a commercial paywall, a novel aspect of the project is to implement new algorithms / research to better improve the detection rate of the license plate detection.
The network is also a very feasible part of the project. P2P networks have extensive research and Python has libraries which support development using web sockets and different communication protocols. However, since the license plate detection is done using existing libraries, there has to be extra emphasis placed on the networking side to make sure this is the primary novel aspect to the project. At the time of writing, there has not been any projects tying license plate detection to a decentralised network which seeks out peers in its immediate vicinity.

Lastly, there are extensive documentation on uploading images to social networks – no foreseeable problem there unless the API access is revoked.

IV. Estimation

As the project timeline spans the entire duration of the 4th year, a plan to map out actions and results with reasonable time estimates is needed. This plan will be used to track progress throughout the project. However,

given the open-ended nature of this design and build project beyond the defined project goals, there is a lot of design, build, and testing to be done. Hence, the estimates for design, build, and testing will inevitably overlap by a fair margin, thereby inflating the number of days to complete the project. The following table (Table 2) shows an estimate for each stage of project delivery, in days. A buffer is also included for unforeseen project issues that may push the timeline back. The corresponding Gantt chart is shown in section XIII.C.

Table 2: Estimation of time needed for each task in the project

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Initial research about topic before first meeting	1	2	1	2	100%
Clarify project aims and goals	2	1	2	1	100%
Prioritise project goals	2	1	2	1	100%
Define project requirements	2	1	2	1	100%
High level design of license plate recognition	5	5	8	8	50%
High level design of P2P network	15	5	15		0%
High level design of security issues, encryption	30	5			0%
Coding of license plate recognition	5	10	8		30%
Coding of P2P network	20	10			0%
Coding of privacy issues, encryption	35	10			0%
Testing of license plate recognition	20	5	15		15%
Testing of P2P network	30	5			0%
Testing of privacy issues, encryption	45	5			0%
Integration of all systems, along with possible hardware	50	10			0%
Packaging and release on GitHub	50	2			10%
Documentation of code	50	2			10%

Report writing	50	10			0%
Making of demo	55	5			0%

To track the progress of the project, a project management software was used. There available for free use. Asana (https://asana.com/) was previously used in the 3rd year project to great effect. However, as this project is not collaborative, the project pane of GitHub was used as a simple tracker (Figure 1). Four groups of tasks were made; the GitHub project tracker allows for easy moving of tasks from one group to another. Another advantage of using the GitHub project tracker is that everything regarding the project is in one place, accessible and modifiable from anywhere.

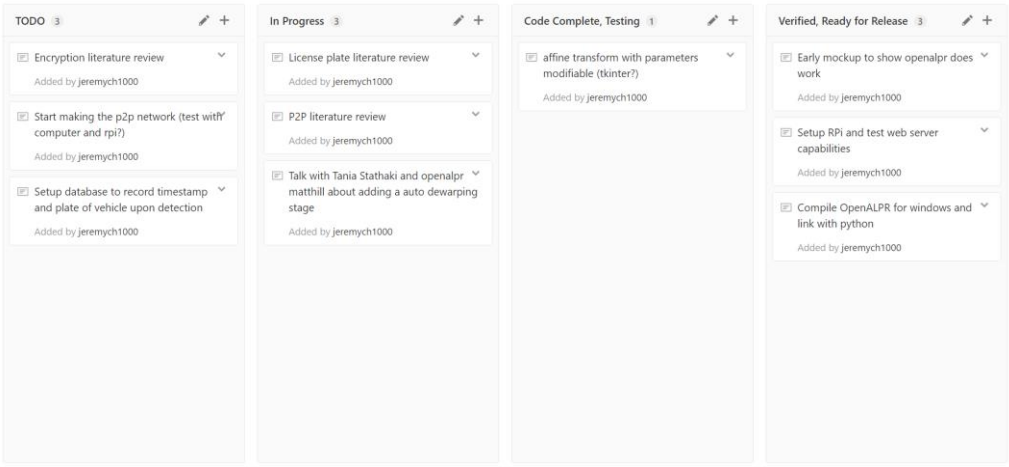


Figure 1: GitHub project tracker

V. Background Reading and Literature Review

A. *License plate recognition*

Most license plate recognition systems operate on a similar basis. Pre-processing, detection, and character recognition [15]. However, image processing and detection can used interchangeably out of order, as in the case of OpenALPR [16].

In the pre-processing stage, a combination of techniques can be used to make the image more recognisable to the following pipeline stages. Azad et al. [17] utilises the HSV colour space instead of the RGB colour space to better determine the location of the plate as the plate is assumed to be of a certain colour. Some implementations implement both the RGB and HSV colour space to get saturation and intensity maps [18]. Duan et Al. [19] first converts the image to greyscale, then normalises the histogram to perform histogram equalisation to get a picture with better contrast. This is especially important during the night when there are a lot of dark areas from shadows. An example is shown below in Figure 2.

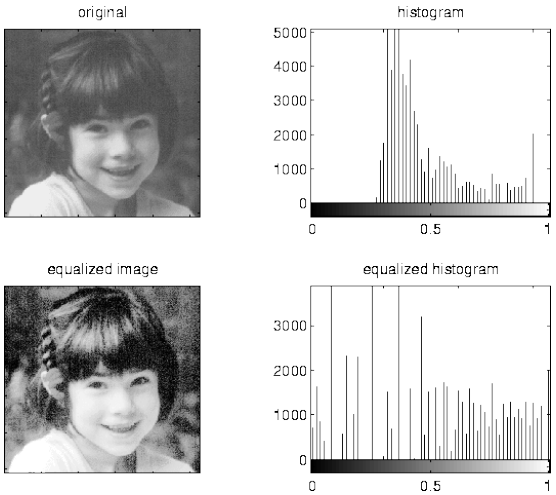


Figure 2: Histogram Equalisation (http://electronicsinourhands.blogspot.co.uk/2012/10/histogram-equalization-in-image.html)

In the detection phase, several algorithms are prominent. Edge detection is used extensively in Badr et al. [20], where the Sobel edge detection method is used in conjunction with thresholding techniques to determine the plate's location in areas with high vertical lines. However, these methods often fail if the assumption that the license plate is captured from a fixed face-on angle is false as the plate can no longer be guaranteed to have perfectly vertical lines. Moreover, using edge detection as the first step often brings false positives, where buildings and road signs can also exhibit perfectly vertical lines – however they are not an area of interest for the license plate detection system.

Hence, other implementations such as OpenALPR's and Liu et al. [21] use an algorithm called Local Binary Patterns (LBP), generally used in face recognition, to get areas of the image where it thinks there is a license plate. However, since LBP uses a sliding window approach, the performance of the detection is quite CPU dependent [16] (can be accelerated using a CUDA but the RPi does not support this). This method has been proven by Nguyen and Nguyen [22], who successfully used LBP with extra classifiers and algorithms in a real time license plate detector.

Edge detection is not unused, however. The most often used combination is a mixture of edge detection algorithms (Sobel, Laplacian, Canny), the Hough transform, and the contour algorithm. The Hough transform is used in implementations like [19] and OpenALPR [16] to find the actual edges of the plate as detection only gives a rough area for the location of the plate, and may end up with an area that does not have a plate in, or slightly bigger than the plate.

Other implementations such as Kwaśnicka and Wawrzyniak [23], and Chang et al. [18] use a technique called connected component analysis before applying the Hough transform. This technique is applied to a binary image of the license plate, and rejects any connected components whose aspect ratio is outside a prescribed range (a license plate is rectangular). The reliability of each implementation has yet to be tested against each other, however.

Lastly, the plate area needs to be warped to be a rectangle for the Optical Character Recognition (OCR) algorithms to work their best. A perspective transformation is often used to warp the image – the final image looks like it has been taken from another perspective (Figure 3). Moreover, straight lines remain straight [24] after the transformation so the next stage, character recognition, is not compromised. OCR libraries such as tesseract (https://github.com/tesseract-ocr) are then used to get the alphanumeric characters from the plate.

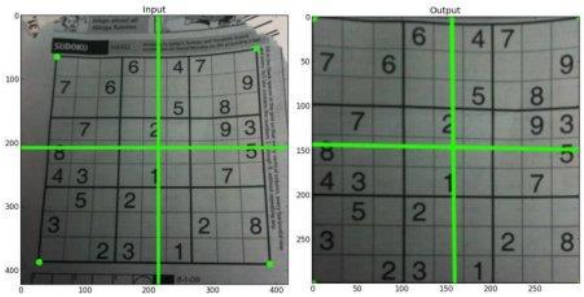


Figure 3: Using a perspective transformation [24]

B. *Peer to peer network*

A P2P network is a network where many machines connect in an ad-hoc manner (they do not join and leave the network based on a schedule), and serve as a server for others. This means there is no need for files and messages or whatever the network is serving to pass through a central server, providing no one single point of failure.

P2P networks account for a significant portion of the web's traffic [25], but its widespread nature means it comes with a few downsides as well. Making sure all copies of a file is free from corruption is one big challenge, as well as security. As all machines are servers of their own, a malicious hacker may attempt to distribute a virus or malware over the network to the many users using the network.

In this project, the main area of P2P networks that will be examined are decentralised networks. Centralised networks still have a central server that does routing, and peer finding, among other tasks [26], whereas decentralised networks do not.

Peer finding in a decentralised network is a challenge. As there is no central server, decentralised networks use techniques such as flooding the network with discover requests [26] and randomly pinging IP's around the world to see if they are part of the network [27]. A well-known decentralised network, Gnutella, uses another technique where it finds new peers by expanding its search radius in its broadcasting phase. When a peer accepts this connection, it rebroadcasts the new peer's address to its own peers, but decreases a counter called Time-to-Live (TTL) to ensure the message eventually dies out (when TTL=0) [28].

However, the broadcasting strategy in decentralised networks means as the search radius increases, there is an increasing amount of traffic in the network, and this may cause congestion in the network [28].

Other types of peer-finding techniques can be found in Mastroianni et al. [29].

C. *Photo evidence publication*

Image upload API's exist for many social networks, and the following table shows the documentation link for three popular image sharing sites.

Table 3: Photo upload API's

Network	Link
Facebook	https://developers.facebook.com/docs/php/howto/uploadphoto
Twitter	https://dev.twitter.com/rest/reference/post/media/upload
Imgur	https://api.imgur.com/endpoints/image

VI. Design

Figure 4 shows a high level view of the flow of the system, with the main tasks shown.

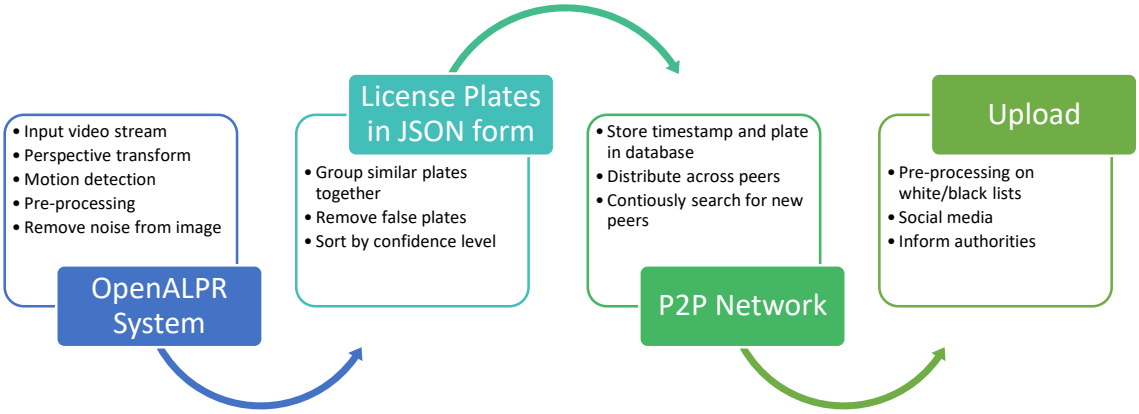


Figure 4: High level view of system

A. *License plate recognition*

As the project specification mentions using existing libraries, OpenALPR was chosen for its open source release and its usability in having Python links to the C code. Moreover, the developer was active on GitHub in responding in issues and queries, further cementing OpenALPR as the top choice for the project.

The license plate recognition system consists of the following stages: pre-processing, OpenALPR license plate detection, and post-processing.

1) *Pre-Processing*

From the literature review, beneficial pre-processing steps include, but definitely not limited to, contrast normalisation, and the perspective transform. As using HSV thresholding and extracting license plate colours such as white and orange interfere with OpenALPR's LBP algorithm (https://github.com/openalpr/openalpr/issues/442), HSV thresholding was not used. However, HSV thresholding remains a candidate for more pre-processing in the future as it works quite well in extracting wanted colours, as demonstrated in Figure 5.

Figure 5: HSV Thresholding (left – original, right – after (cropped as the rest of the image is black))

OpenCV, the preferred computer vision library for this project, has a perspective transform. This transform takes in a set of coordinates of the input image, followed by where those coordinates map to in the output image. Ideally in the final product, the pre-processing system intelligently picks out the car using motion detection or library image recognition techniques, and comes up with these coordinates automatically, as opposed to having to define them by hand. OpenALPR does not support automatic perspective transform – it has a utility to do pre-processing perspective transforms but again this is manual. This utility will be replicated in Python as well.

Last but not least is motion detection. Several simple motion detection techniques are used, like taking the absolute different between sum of squares of pixels for consecutive frames to see areas of motion, along with using dilation techniques to remove holes in the difference map from the sum of squares. This technique works quite well, but assumes that the first frame is the background, which is not a very sustainable assumption in real world usage. There are more complicated motion detection techniques planned to be implemented.

A final note is that motion detection will serve as the trigger for the infrared flash during night time operation.

2) OpenALPR

The pre-processed image is then fed through OpenALPR, with suitable parameters defined such as country, region, max amount of results to return, etc. OpenALPR is also configured to return a JSON output, so Python can access the diferent members of the output array easily, for example, timestamp, confidence level, the actual plate, and the coordinates of such plates.

As OpenALPR is a software package, there is not much to be modified in this section.

3) Post-Processing

Post-processing includes replicating a commercially available feature of OpenALPR. Specifically, as OpenALPR processes video frames, a Python script will keep track of how many times a specific plate has occurred, and also take into account the confidence level to return a one plate for a period of video instead of multiple repeats of the same plate.

B. Peer to peer network

Since the amount of users for this project's P2P network is unknown, nor is it meant to be in the millions, bootstrapping the network using brute-force broadcast, or random walking IP's is unlikely to result in finding a peer. Therefore, the proposed method of bootstrapping the network is to use a list of known hosts, along with a, or several, bootstrapping servers which hold knowledge of the IP addresses of current peers. Using several servers will provide multiple points of failure instead of one as that would be detrimental to a P2P network.

The actual sending of the plates through the network is still under consideration, but an initial direction would be to send them in JSON form, and store them on a local database on each device (RPi). This could be done using MySQL databases, which would provide a nice link to a web interface using Python and Django if so necessary.

C. Photo evidence publication

Several scripts will be written to handle image upload. Pictures can be tagged with a location, and the plate data can be sent over HTTP or HTTPS using multipart/form-data.

Since there will be many pictures of the same car as the input is video, the final photo evidence will have to choose a best image to upload to social networks. The logical approach is to choose the image with the highest confidence level.

D. *Hardware*

The hardware package will target a Raspberry Pi running Raspian, a fork of Debian. The software package will contain directions on how to compile the system, as well as any dependencies (OpenALPR, OpenCV, Tesseract, Python, etc). A short wiki can also be written to put on GitHub pages or otherwise.

Additional hardware will include a camera (without IR filter), and a IR flash to ensure night time operation.

Regarding the position of the camera in the finished system, there needs to be careful consideration as some cars do not have license plates on the front of the car (motorcycles). Therefore, it is best to point the camera in the direction of traffic so the license plates on the rear of the car (orange in colour by law [6]) get detected. Another design concern is that during night-time operation, the headlights blind out the camera, leaving the license plate overexposed. Testing will need to be done on this.

VII. BUILD

Development was done on a PC running Windows 10. The IDE is PyCharm from JetBrains. Python 3 is installed using Miniconda. OpenCV and OpenALPR are compiled from their GitHub repositories. OpenALPR's dependencies, leptonica, and tessearact were compiled from their GitHub repositories as well.

Some web server work was done on the RPi using a LAMP stack. Similar packages to the PC are installed there to provide two testing environments.

GitHub is used to sync code across devices. SSH is used to access the Raspberry Pi.

A. *License plate recognition*

A successful prototype mock-up system consisting of pre-processing, OpenALPR detection, and license plate grouping was created. This is the first two stages of the high level flow (Figure 4). TkInter, a Python GUI framework, is used to create simple GUI to change parameters of the pre-processing (Figure 6**Error! Reference source not found.**). The first four lines of sliders set the starting coordinates of the perspective transform. HSV sliders do real time colour extraction based on the hue, saturation, and value (brightness) of the image. As tasks in TkInter use a 'call function after a set period' and not a sleep function, another slider was put it to configure the rate of image processing – in video, this corresponds to how many frames are processed per second.

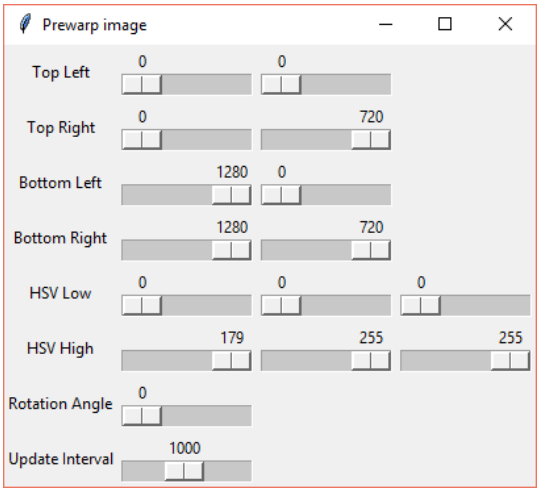


Figure 6: TkInter simple testing interface

This prototype works well and successfully outputs license plate data based on input video.

B. *Peer to peer network*

The actual construction of the peer to peer network has not been started, but all the supporting infrastructure has been set up. This includes web servers and databases. However, these configurations may change based on what framework the system uses – to be finalised later. Python has many networking libraries and web frameworks to ease the effort of programming a network from scratch. As demonstrated in section V.B, building the P2P network is feasible. There should not be any issues that will foreseeably cause the entire development process to fail.

C. *Photo evidence publication*

This section has not been started yet. As shown in section V.C, uploading an image to a web service is feasible, very well supported, and very well documented. The implementation should be straightforward.

With regards to picking the best image, a simple sort on the JSON results from OpenALPR will give the image with the highest confidence level.

D. *Hardware*

Several bootstrapping scripts have been created to help setup a brand new RPi by downloading all the needed libraries and dependencies. This is assuming the end-user has already installed Raspian on their RPi.

VIII. TESTING

As the only section of the system with a working testable prototype is the license plate recognition section, this section will focus on that.

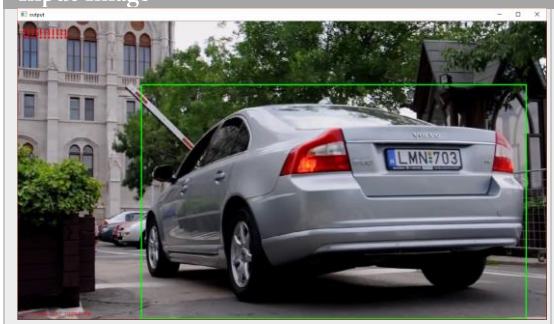
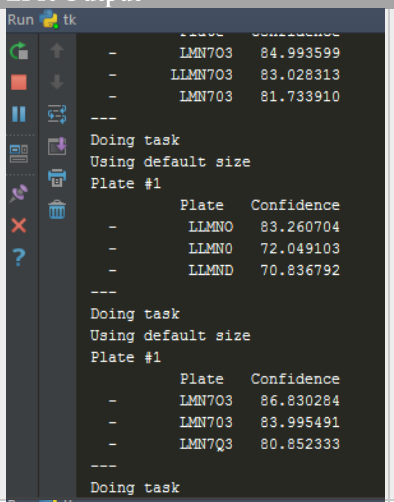
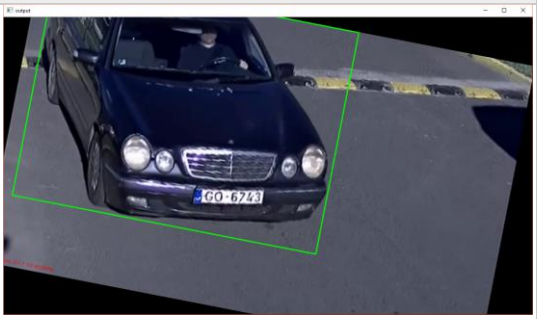
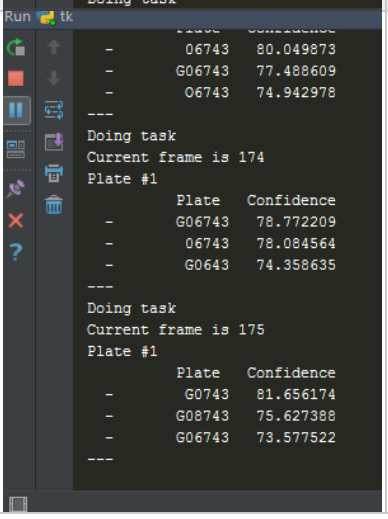
A. *License plate recognition*

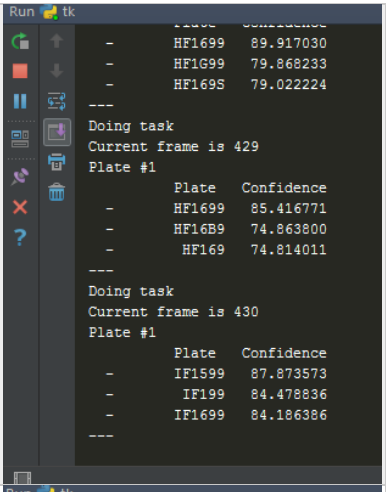
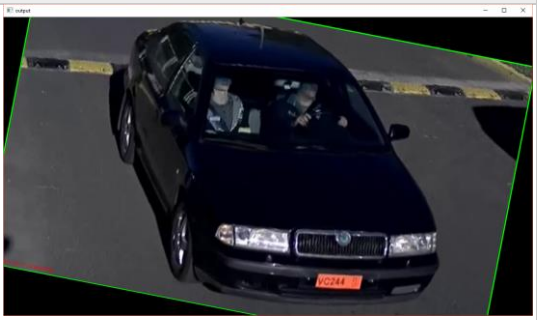
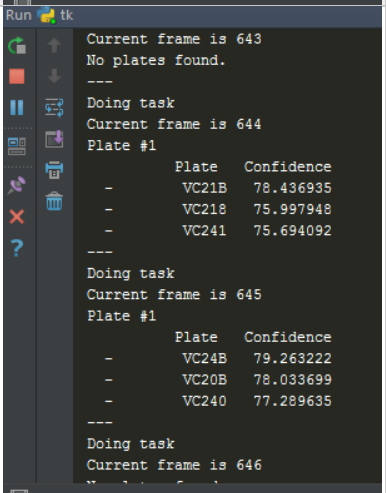
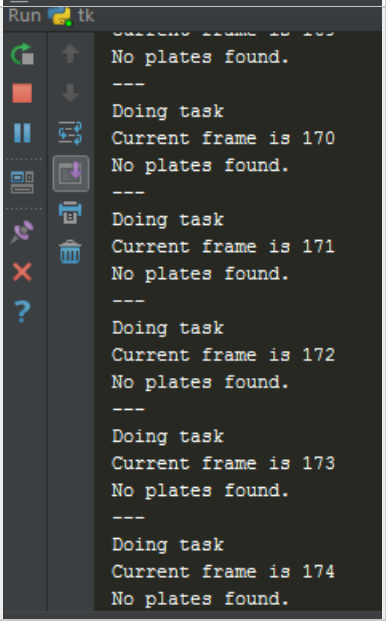
Several sample videos were downloaded off the web as input to the LPR system. For each video, paramters were changed (rotation angle and perspective transform coordinates) in the aforementioned TkInter interface to try and produce an input image that was best suited for OpenALPR (horizontal, rectangular license plate without

any warping). No HSV colour thresholding was used. Table 4 shows a few examples of the prototype correctly detecting plates, and also includes some comments where applicable.

More testing videos will definitely need to be gathered, in a range of lighting conditions and locations. A possible solution is to set up a video stream from a webcam connected to the RPi pointing down a main road. This should provide day/night testing. Otherwise, several sample videos can be taken with a smartphone. YouTube was a disappointing dead end in trying to find sample videos as most car footage is taken from dashcams, and does not give a constant background – no good for the current form of motion detection, which assumes a static non-moving background.

Table 4: LPR Testing

Input Image	LPR Output	Comments
		Quite accurate, but sometimes fails to detect the last '3'.
		Fails to detect the plate when it is in the edges of the input image. Sometimes misses out a character or two.

		Pretty accurate, but sometimes fails to detect the full 'H', and only detects one side as 'I'.
		Not very reliable in find the license plate, but to be expected as it is small and blurry, and orange.
		No plates detected at all as it does not follow the EU convention of license plate letter/number combinations. May need some more warping.

IX. Evaluation Criteria

Below are several evaluation criteria that will determine the performance of the system. Since the prototypes are still in their early stages of testing, this list will definitely change.

A. *License plate recognition*

- Ability to detect a license plate after pre-processing
- Reliability in coming up with the same result from subsequent video frames
- Accuracy in detecting the correct license plate (not skipping any characters, numbers, not getting false readings such as O being 0, 6 being 5, etc.)
- Flexibility in detecting license plates of differing character lengths
- CPU time used in detecting a plate (real-time operation is required)
- Maximum resolution supported/required to ensure real-time operation

B. *Peer to peer network*

- Ability to bootstrap itself from a fresh installation
- Ability to find peers in its near vicinity
- Ability to verify authenticity of peers
- Ability to correctly send information to peers and verify that all peers have similar information
- Bandwidth usage

C. *Photo evidence publication*

- Latency from detection to publication
- Ability to resume upload of evidence after a sudden shutdown or loss of connectivity
- Ability to select best image for upload

D. *Hardware*

- Size of binary package
- Ease of compilation
- Hardware support (should also work with generic USB webcams)

X. Risks and Fallbacks

As this project is more implementation and product based than research and simulation based, there is a chance that it may turn into just stitching existing libraries together. Therefore, after the background literature review, more emphasis was placed on the network side, as there has not been any products linking license plate recognition with a decentralised network yet. The spatial awareness of the network is also a novel aspect of the project.

For the sections below, there are several possible scenarios: complete failure, non-completion, and slow completion. Solutions or comments are outlined for each.

A. *License plate recognition*

1) *Complete failure*

This is not possible as there is already a working prototype showing successful operation of OpenALPR on sample images and video.

2) *Non-completion*

Several pre-processing steps or the automatic perspective transform may not get completed on schedule. If so, there should be a manual way of setting pre-processing parameters. There is, as shown by the TkInter interface.

3) Slow completion

In the case that the system is running behind schedule, more priority should be given to the networking side and/or testing.

B. *Peer to peer network*

1) Complete failure

In the case of complete failure, a centralised network can be used, where different systems report license plates to a central server. This server will have a database storing the different timestamps of detected license plates, and will routinely (either on a schedule or upon a new insertion in the database) search for speed violations. The estimated time of setting up a centralised network is 2 – 3 days as most of the PHP server/database code can be recycled from an old EE3 project.

2) Non-completion

In the case that some parts of the network should not be completed, an existing decentralised P2P messaging network can be used. Bitmessage was mentioned during related work research, and would suffice in sending messages across peers.

3) Slow completion

A known peer list can also replace the peer finding aspect of the project if time is running out. This will guarantee at least some connections between peers, however, defeats the point of a P2P network.

C. *Photo evidence publication*

1) Complete failure

Very unlikely given the abundance of photo uploading services on the web. A private image upload script is also tested and working and can be used in the case that all image uploads to popular hosts fail.

2) Non-completion

Unlikely given the clear instructions to upload images to the web, however if the final system uses non pre-defined API keys there needs to be a fallback upload to an anonymous image host, like Imgur.

3) Slow completion

Priority will be given to license plate detection and the networking side as image upload is relatively simple. Moreover, from the EE3 project, there is PHP code that handles image upload, server interfacing, and automatic generation of a PDF containing relevant details, as well as automatic emails to any email address.

D. *Hardware*

1) Complete failure

A software package will serve as the fallback. As long as the software can be compiled and run it will serve as a fallback for the scenario where all hardware fails. Webcam footage can be simulated by input videos.

2) Non-completion

Again, pre-recorded videos can replace the live webcam feed. In the case that the IR flash cannot be tied to motion detection, it can be set to always be on.

3) Slow completion

Priority will be given to license plate detection and the networking side as the hardware is more to show off the implementation.

XI. Deployment and Maintainance

Deployment will entail a GitHub release of a software package (https://github.com/jeremych1000/ee4-FYP/), with a readme and steps or scripts to help recompile the project.

Maintenance will be done where necessary to ensure the bundled dependencies, if any, stay up to date. However, as the repository is private there is no planned maintain ace work after the project concludes. This will change depending on the success of the project and interest from 3rd parties, if any.

XII. References

[1] 'Number of speeding convictions from average speed cameras - a Freedom of Information request to Driver and Vehicle Licensing Agency', *WhatDoTheyKnow*, 28-Jul-2015. [Online]. Available: https://www.whatdotheyknow.com/request/number_of_speeding_convictions_f. [Accessed: 22-Jan-2017].

[2] 'Number of average speed cameras have doubled in three years', *This is Money*, 31-May-2016. [Online]. Available: http://www.thisismoney.co.uk/money/cars/article-3617584/Number-average-speed-cameras-doubled-three-years.html. [Accessed: 23-Jan-2017].

[3] 'FS_Speed.pdf'. [Online]. Available: http://m.swov.nl/rapport/Factsheets/UK/FS_Speed.pdf. [Accessed: 29-Jan-2017].

[4] 'Automatic Number Plate Recognition - Police.uk'. [Online]. Available: https://www.police.uk/information-and-advice/automatic-number-plate-recognition/. [Accessed: 29-Jan-2017].

[5] '2014_04_29-Californians-Overwhelmingly-Support-Use-of-License-Plate-Readers-and-Their-Ability-to-Solve-Crimes.pdf'. [Online]. Available: https://vigilantsolutions.com/wp-content/uploads/2014/04/2014_04_29-Californians-Overwhelmingly-Support-Use-of-License-Plate-Readers-and-Their-Ability-to-Solve-Crimes.pdf. [Accessed: 23-Jan-2017].

[6] 'inf104-vehicle-registration-numbers-and-number-plates.pdf'. [Online]. Available: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/533255/inf104-vehicle-registration-numbers-and-number-plates.pdf. [Accessed: 29-Jan-2017].

[7] 'Raspberry Pi 3 Model B', *Raspberry Pi*. .

[8] 'The UK's Speed Camera Types | Fixed and Mobile speed cameras explained'. [Online]. Available: https://www.speedcamerasuk.com/speed-camera-types.htm. [Accessed: 23-Jan-2017].

[9] Y. D. Silva, 'Average speed enforcement (ASE) camera'. [Online]. Available: https://www.birmingham.gov.uk/info/20163/road_safety/364/average_speed_enforcement_ase_camera. [Accessed: 23-Jan-2017].

[10] 'specs3_vector_v1.1_final.pdf'. [Online]. Available: http://www.jenoptik.co.uk/sites/vysionics.vmdrupal04.lablateral.com/files/specs3_vector_v1.1_final.pdf. [Accessed: 23-Jan-2017].

[11] 'ARES | Fixed ALPR', *PlateSmart*, 18-Jun-2015. .

[12] 'LPR/ANPR License Plate Recognition SDK'. [Online]. Available: http://www.dtksoft.com/dtkanpr.php. [Accessed: 23-Jan-2017].

[13] 'OpenALPR Features'. [Online]. Available: http://www.openalpr.com/features.html. [Accessed: 23-Jan-2017].

[14] J. Warren, 'Bitmessage: A peer-to-peer message authentication and delivery system', *White Pap. 27 Novemb. 2012 Httpsbitmessage Orgbitmessage Pdf*, 2012.

[15] V. Sharma, P. Mathpal, and A. Kaushik, 'Automatic license plate recognition using optical character recognition and template matching on yellow color license plate', *Int. J. Innov. Res. Sci. Eng. Technol.*, vol. 3, no. 5, 2014.

[16] 'Accuracy Improvements — openalpr 2.2.0 documentation'. [Online]. Available: http://doc.openalpr.com/accuracy_improvements.html#openalpr-design. [Accessed: 24-Jan-2017].

[17] R. Azad, F. Davami, and B. Azad, 'A novel and robust method for automatic license plate recognition system based on pattern recognition', *Adv. Comput. Sci. Int. J.*, vol. 2, no. 3, pp. 64–70, 2013.
[18] S.-L. Chang, L.-S. Chen, Y.-C. Chung, and S.-W. Chen, 'Automatic License Plate Recognition', *IEEE Trans. Intell. Transp. Syst.*, vol. 5, no. 1, pp. 42–53, Mar. 2004.
[19] T. D. Duan, T. H. Du, T. V. Phuoc, and N. V. Hoang, 'Building an automatic vehicle license plate recognition system', in *Proc. Int. Conf. Comput. Sci. RIVF*, 2005, pp. 59–63.
[20] A. Badr, M. M. Abdelwahab, A. M. Thabet, and A. M. Abdelsadek, 'Automatic number plate recognition system', *Ann. Univ. Craiova-Math. Comput. Sci. Ser.*, vol. 38, no. 1, pp. 62–71, 2011.
[21] L. Liu, H. Zhang, A. Feng, X. Wan, and J. Guo, 'Simplified Local Binary Pattern Descriptor for Character Recognition of Vehicle License Plate', in *Imaging and Visualization 2010 Seventh International Conference on Computer Graphics*, 2010, pp. 157–161.
[22] T.-T. Nguyen and T. T. Nguyen, 'A real time license plate detection system based on boosting learning algorithm', in *Image and Signal Processing (CISP), 2012 5th International Congress on*, 2012, pp. 819–823.
[23] H. Kwaśnicka and B. Wawrzyniak, 'License plate localization and recognition in camera pictures', in *3rd Symposium on Methods of Artificial Intelligence*, 2002, pp. 243–246.
[24] 'OpenCV: Geometric Transformations of Images'. [Online]. Available: http://docs.opencv.org/3.1.0/da/d6e/tutorial_py_geometric_transformations.html. [Accessed: 25-Jan-2017].
[25] The Government of the Hong Kong Special Administrative Region, 'PEER-TO-PEER NETWORK'. [Online]. Available: http://www.infosec.gov.hk/english/technical/files/peer.pdf. [Accessed: 25-Jan-2017].
[26] H. Park, R. I. Ratzin, and M. van der Schaar, 'Peer-to-peer networksprotocols, cooperation and competition', *Streaming Media Archit. Tech. Appl. Recent Adv.*, pp. 262–294, 2010.
[27] C. Grothoff and C. GauthierDickey, 'Bootstrapping Peer-to-Peer Networks'. [Online]. Available: http://grothoff.org/christian/dasp2p.pdf. [Accessed: 21-Jan-2017].
[28] Q. H. Vu, M. Lupu, and B. C. Ooi, *Peer-to-Peer Computing: Principles and Applications*. Springer Science & Business Media, 2009.
[29] C. Mastroianni, D. Talia, and O. Verta, 'Designing an information system for Grids: Comparing hierarchical, decentralized P2P and super-peer models', *Parallel Comput.*, vol. 34, no. 10, pp. 593–611, Oct. 2008.

XIII. APPENDIX

A. *Table of Figures*

B. *Table of Tables*

C. *Gantt Chart*