Detection and Speed Estimation of Vehicles Using Resource Constrained Embedded Devices

Ryan McQueen
ryan.mcqueen@alumni.ubc.ca

March 2018

Abstract

In this report, a novel approach of automating counting vehicles and speed estimation for traffic monitoring through the usage of a Raspberry Pi is explored. This device has a small amount of processing power in comparison to a modern day laptop and as a result, requires a computationally inexpensive method for detecting vehicles. Vehicle detection is performed using morphological processes such as erosion and closing to filter undesirable objects, background subtraction [1] to ignore minor environmental changes, and contour detection to identify objects of interest.

1 Introduction

The task of counting vehicles passing on a given road for a municipality is an extremely costly and time inefficient process. This counting requires individuals to be hired to watch the traffic and record on paper every time a car passing by is witnessed. This manual recording may result in errors due to the recorder being distracted or simply losing count on how many vehicles have passed in a given time frame. Additionally, there is no way to validate the counting the recorder has performed is accurate as there is no video recording of the vehicles. While the problem may be addressed by simply recording the traffic for the duration the recorder is there and analyzing it afterwards, it results in more time and money spent and does not provide a way to additionally record the speed at which the vehicles are travelling. In this report, a generalized system is introduced which resolves the issue of inaccuracies of manual vehicle counting, provides a way to measure the speed of vehicles counted, and provides a cost effective solution.

In a generalized system, the components required to perform such tasks can be divided into four main components: recording the area of interest, analyzing video frames to detect vehicles, performing calculations of the speed the vehicle is travelling, outputting a log file of the total count of vehicles with their

associated speeds, and saving both a raw video as well as the analyzed video containing the appropriate car counts. The first component involves deploying a Raspberry Pi with a Raspberry Pi camera module V2 into an area of interest for vehicle tracking. This requires placing the camera into a position such that top down view is achieved and is illustrated further within the Section 2. Secondly, undesirable objects must be removed from the video frames. This step contains numerous operations and algorithms in order to obtain only the vehicle contours. In order to reduce objects which do not follow typical vehicle shapes or are otherwise unwanted, morphological processes such as opening and closing are performed. Following from these processes, a background subtraction model must be utilized in order to remove significant noise such as shadows or subtle environment changes. Thirdly, once a contour has been detected, it can be observed as it enters a calibration region to track the amount of time it takes to reach the end of the region. By doing this, an accurate representation of the vehicle's speed can be made due to a known real world distance. Next, a log file must be generated to capture the speed at which a vehicle was travelling for later analysis and an associated image capture of the vehicle that was travelling at this speed.

There exists a wide variety of algorithms to perform object detection and tracking in the literature [2, 3]. Consideration was placed on the efficiency of the proposed system. There are algorithms which yield high correct classification techniques such as that of a neural network [2], however, they require extensive processing power which is not possible on a single Raspberry Pi board. Due to these computational power constraints, heavy emphasis was placed on methods which do not require dedicated video cards, or high amounts of processing power. The list of available algorithms drastically decreased due to the computational constraints and as a result, contour detection was found to be the highest performing solution for real-time detection of objects at no cost of accuracy if appropriate configurations had been performed to the morphological operations. Additionally, background subtraction was found to noticeably reduce performance due to the number of frames recorded per second and the amount of frames remembered for the subtraction, however, it was deemed appropriate due to the amount of environmental changes which may occur when recording.

Assumptions that were made for this work are quite limited and may impact the results shown in the experimentation section. There are four assumptions made, the first being that for all frames being read, the vehicles within a frame were moving horizontally. Secondly, the video being recorded was during the day or early evening so that there still existed some natural light. Third, there is a known region of a frame such that a calibration region could be constructed and related to real world measurements. Lastly, the camera was in a position such that if two vehicles were to pass each other that the camera would be able to observe both vehicles as they passed.

The goal of this project was to explore an automated, accurate, and cost effective method to count, track, and estimate the speed of passing vehicles. To succeed with this goal, one has to compromise on the resolution of the capture

video, the quality of the detection algorithms, and perform many configuration modifications in order to achieve promising results.

2 Existing Work

Applications of embedded devices for the purpose of traffic monitoring are quite limited within the literature [4]; indicating a large area for innovation through the usage of cost effective embedded devices such as the Raspberry Pi. The existing work provides a look into the effectiveness of simply detecting and counting a vehicle within a given video frame, however, there is no means to calculate the speed of a detected vehicle. The authors detail a procedure similar to the one explored within this paper, providing validation for the steps taken. The approach the authors followed consists of four major components. The first component begins with reading a frame from the current video source through the Raspberry Pi camera module with a resolution of 640x480 at 30 frames per second. Secondly, background subtraction is performed in order to isolate the moving objects within a frame. Morphological processes are performed in order to remove objects which fall below, or above a certain size. Lastly, contour detection is performed and a bounding rectangle is drawn around each detected contour. From this, the usefulness of calculating a vehicle's speed can be identified. Enforced policing can be directed to the region being monitored to ensure the safety of motorists along a particular roadway, prompt discussions on the implementation of speed reduction techniques such as speed bumps, or simply alert the appropriate authorities of a potential issue within their community.

3 Approach

The approach taken within this paper is demonstrated within Figure 1. This figure outlines the processing of each video frame taken by the Raspberry Pi camera module. For the current frame, background subtraction is performed in order to isolate only the moving objects within a frame. Given the moving objects, morphological processes are performed in order to enhance the potential objects within the frame. From the enhanced objects within the frame, contours are able to be detected and tracked, and a speed calculation can be made. The results of the speed calculations are outputted into a log file, the raw, and analyzed videos are output in H264 format to disk.

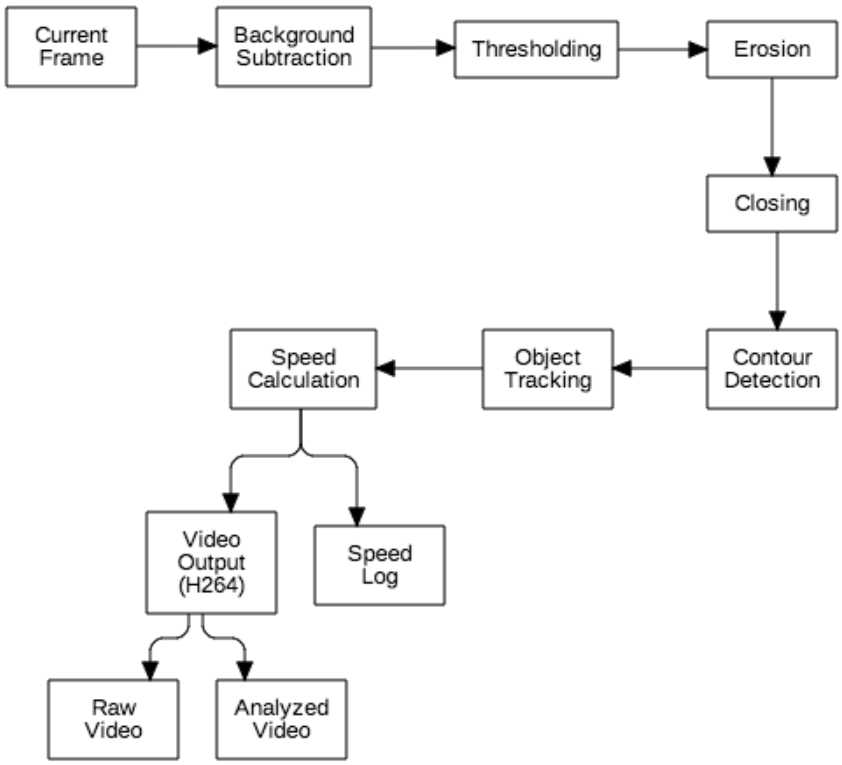

Figure 1: High Level System Overview

3.1 System Requirements

All data collection and analysis was performed using a Raspberry Pi 3 Model B+ board. The nature of this work requires a modest CPU in order to handle the extensive video processing, and as a result, boards with lower specifications than that of the board used may have difficulties handling the processing operations. An illustration of what the Raspberry Pi 3 Model B+ board (Figure 2).
The specifications of the board are as follows:

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- CSI camera port for a Raspberry Pi camera module (V2 used)
- Micro SD port for an operating system and data storage
- 5V/2.5A DC power input

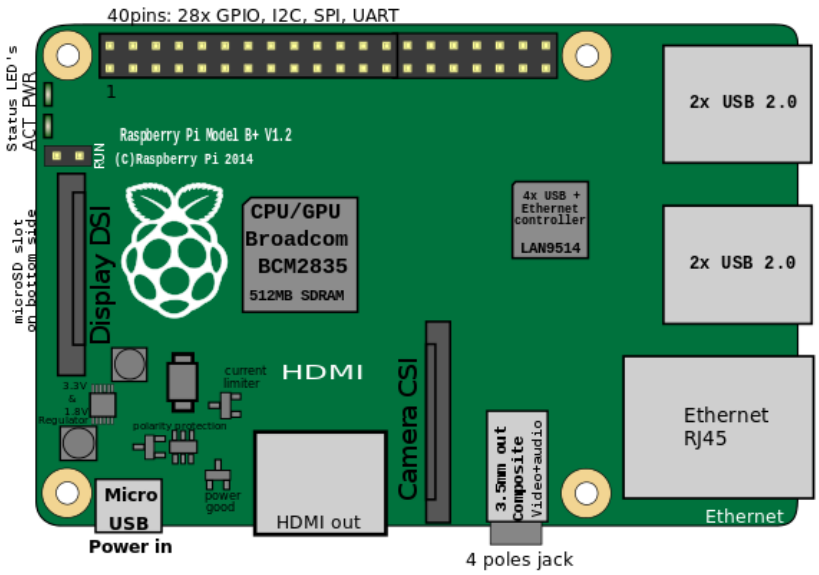

Figure 2: Raspberry Pi 3 Model B+ Board. Source: By Efa2 (Own work) [CC BY-SA 4.0] (https://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons

The specifications for the camera module are as follows:

- Sony IMX219 8-megapixel sensor

Any additional configuration settings on the Raspberry Pi such as increasing the amount of available GPU memory for processes were left at the default value. Increasing these may enhance performance of the system.

3.2 OpenCV Library (C++ 11)

This implementation is written in C++11 and utilizes the OpenCV [5] library's existing functionality for the video source, background subtraction, and contour detection. A brief summary of the functions used are listed below:

- cv::Mat is to store each frame from a video
- cv::BackgroundSubtractorMOG2 for background subtraction
- cv::VideoCapture for reading from the Pi cam module
- cv::VideoOutput to store each analyzed frame
- cv::findContours for contour detection

3.3 Live Video

In order to begin counting and tracking vehicles, a means to obtaining live video must be introduced. A Raspberry Pi camera module V2 was used for this project due to its higher megapixel count and higher quality lens as opposed to the V1 version. A USB web cam may also be used for this process, however, it may introduce some additional latency due to the speed at which a serial port can read data. In order for the reduction of false-positives caused by shadows, or obstructed vehicles due to one passing by another, an angled view is required. This view is demonstrated within Figure 3.

Figure 3: Angled Camera Viewpoint

Following the placement of the camera module, a pre-determined area must be selected from an image at the defined height. This will ensure there are consistencies with the speed measurements as the position will be fixed. This component is the most important as it will set the accuracy for the system in the following components. Failure to define the correct dimensions and the appropriate real world measurements will result in inaccurate tracking and potentially result in failure of contour detection. An example of how the calibration may look is shown within Figure 4

Figure 4: Selected Calibration Region

3.4 Contour Detection

For each frame read by the camera module, analysis was required to remove any potential noise in the image such as pedestrians walking by, animals within the frame, or cyclists. This component involved numerous algorithms in order to achieve the desired result of obtaining only vehicles for analysis. The first step applied to each frame is to subtract the background image. When the background has been isolated, minor environmental changes or shadows are effectively eliminated and thus reduces the complexity of vehicle detection. An example of the extracted background from the frame is shown within Figure 5. It can be seen there resides some residual fragmentation from the subtracted object, however it was found to cause no effect on the detection of contours.

With only the moving objects left to examine within a frame, some filtering operations must be applied to further eliminate any potential objects which are not desired to be matched. The operations used to perform such filtering are known as morphological operators. An erosion operation was initially applied to remove any small noise within the image caused by thresholding values of pixels to match that of a binary representation where 0 signifies a black pixel and 1 signifies white pixel. Following from this, the closing operation was performed on a frame. The closing operation performs dilation such that any objects which do not fit within some pre-defined shape and size are filled in and set as white pixels, followed by erosion. For the purposes of this project, closing was performed to maintain some of the form of the vehicle which erosion may have removed. This can be useful in the scenario in which a bird has flown into the frame, or a pedestrian is walking by as they will be removed from the frame by the erosion operation. An example of the effects of erosion and closing is shown within Figure 6

Figure 5: Background frame after segmentation from foreground frame

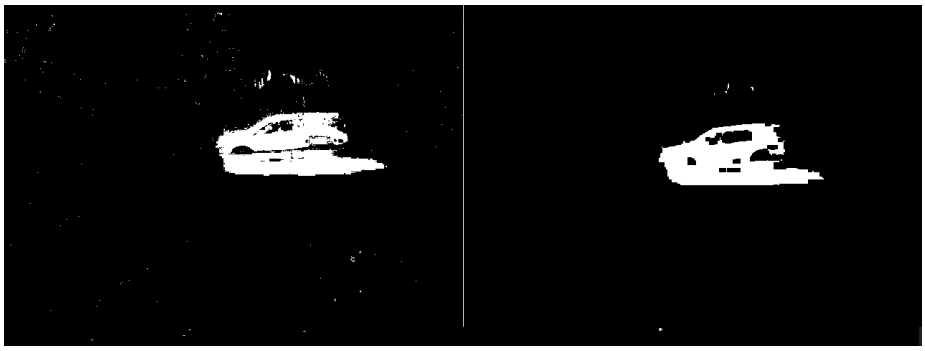

Figure 6: Outcome of Morphological Operators on a Frame (Left: Raw frame after thresholding, Right: Morphological operators applied)

With the contours now clearly identifiable, a contour detection algorithm implemented within OpenCV [5] was applied. This resulted in the convex hulls of each contour as well as the area of each contour. From this, a bounding box could be constructed around each contour.

3.5 Speed Calculation

From the bounding box of each contour, a means to tracking when a vehicle would enter and exit a region became possible. An example of what this looks like can be seen within Figure 7. In order to track the speed of a vehicle, the calibration region discussed within the Live Video section will be referenced. Upon the vehicle's successful detection and determined direction of travel, the right or left side of the bounding rectangle will be used to identify whether or not it has entered or exited the calibration region. Upon entering the calibration region, a flag for monitoring the vehicle's distance is enabled. With this flag enabled, the number of frames it has travelled since entering, and the distance in pixels is stored. Upon successful exit, the time can then be calculated based on the total amount of frames covered and a distance can be determined by converting the pixels to a measurement in meters. The for distance is rather trivial and shown below:

$$Distance = \frac{|P_{end} - P_{start}|}{c}$$

where P represents a pixel, and c represents a constant conversion value.

Following this, the timing method cannot use a simple wall clock timing method as the time is dependent on the frame rate of the video. The equation to determine the vehicle's time spent within the calibration region in seconds is defined by the following:

$$Time = \frac{total\ \ frames\ travelled}{frame\ rate}$$

Figure 7: A detected vehicle with a bounding box drawn around it

With both of these values a simple velocity calculation is performed and the resulting meters per second is converted to kilometers per hour. This equation is shown below:

$$v = \frac{d}{t} * 3.6$$

3.6 Data Logging

After a vehicle has been detected and a speed has been associated with it, the data must be logged for future analysis and validation for testing purposes. The speeds are saved in a comma separated value format to allow for easy importation and manipulation into languages such as R. With the speeds, an associated identification number is written to allow for a relation to be applied to each speed. In order to identify potential reoccurring speed violators, images are saved of each detected vehicle upon passing the calibration region. An example of a saved vehicle image can be seen within Figure 8.

Figure 8: Snapshot of a detected vehicle upon passing the calibration region

Raw and analyzed videos from the system are required to be saved to identify any potential missed edge cases, or signs of a false detection and are saved in a h264 format. This allows for efficient video compression while maintaining high video quality.

4 Case Study

Numerous tests and hours of video data were collected in order to gauge the efficiency and accuracy of the proposed system. These tests included expected cases of failure such as the scenarios when a large amount of shadowing is trailing a vehicle, unexpected cases of failure such as two pedestrians walking together and being detected as a vehicle, and finally, scenarios in which the system should accurately detect and track a vehicle. The system was deployed in a residential environment to allow for a large exposure of vehicles, animals, and pedestrians to be observed. Additionally, a testing run was performed on a pre-existing video of cars travelling on a freeway to gauge the initial effectiveness of the system. The results of the vehicle counts and speed estimations can be found within Table 1 & 2. Figures shown within this segment range from early morning to late in the afternoon with varying brightness levels caused by the movement of clouds. In addition, subsets of the recorded frame have been used to focus the reader's attention onto the important information presented within each image, indicating the varying view points among each image.

In Figure 9, it is shown that a pedestrian walking their dog was not detected as a vehicle, thus verifying the system's ability to filter out smaller objects within a frame.

Figure 9: Undetected pedestrian walking their dog

In Figure 10, it is shown the vehicle was detected, however, a large chunk of the vehicle has been ignored due to apparent shape destruction caused by the filtering process.

Figure 10: Poorly detected vehicle

In Figure 11, an interesting result was found. It is shown that the vehicle was initially detected, but upon passing through the calibration region, the tracking stopped, and only began again once the vehicle had passed through the region. It was later found this was caused by an issue with reading multiple frames at once.

Figure 11: Demonstrates a tracked vehicle being dropped upon passing through the calibration region

Environment	Expected Count	Actual Count	Correctness in %
Residential	5	5	100%
Freeway	52	52	100%

Table 1: Vehicle Count Results

Environment	Expected Speed (mp/h)	Actual Speed (mp/h)	Correctness in %
Residential	30	27.595	91.98%
Freeway	40	35	87.5%

Table 2: Vehicle Speed Results

5 Discussion

A fast and highly accurate automated vehicle detection and speed calculating system has been presented which expands on the existing work of [4]. Future work for this system would include two main components. Removing the requirement for real world measurements of the desired road area would greatly improve the usability of this system. The need to measure the region of interest and select the pixel range the calibration region falls under as it is extremely time consuming. Additionally, finding a way to classify the vehicles which have been detected would further improve the data logging as a count of certain vehicles classes can be constructed from this. For example, if a large amount of commercial trucks are found to be travelling on a residential road, a solution can be constructed by a municipality to resolve this issue.

6 Conclusion

From the results this paper, it has been demonstrated in Section 4 that the proposed problems are able to be satisfied. The problems entailed are producing a highly accurate, cost effective, and automated solution. In the literature, [4], it is noted that vehicle detection is possible on a resource-constrained device such as a Raspberry Pi, and from the results of this paper, the functionality of speed calculation and detailed data logging has been included. Additional functionality to implement includes the classification of vehicle types i.e, car, truck, and commercial truck. In Section 4 of this paper, vehicles have been shown to have a 100% detection accuracy followed by speed calculations having an accuracy of greater than 80%. In summary, a cost efficient and automated solution is plausible for replacing the manual, expensive task of counting vehicles.

References

[1] P. Kaewtrakulpong and R. Bowden. "An improved adaptive background mixture model for real-time tracking with shadow detection". In: *Proceedings of 2nd European Workshop on Advanced Video Based Surveillance Systems*. Vol. 5308. 2001.

[2] Kunihiko Fukushima and Sei Miyake. "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position". In: *Pattern Recognition* 15.6 (Jan. 1982), pp. 455–469. ISSN: 00313203. DOI: 10.1016/0031-3203(82)90024-3. URL: http://dx.doi.org/10.1016/0031-3203(82)90024-3.

[3] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[4] M. Kochláň et al. "WSN for traffic monitoring using Raspberry Pi board". In: *2014 Federated Conference on Computer Science and Information Systems*. Sept. 2014, pp. 1023–1026. DOI: 10.15439/2014F310.

[5] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).