Just a T.A.D.
Midyear Design Review Report

Christopher Barbeau, CSE, Cyril J. Caparanga, CSE, Alexander Dunyak, CSE, and Matthew T. Shin, CSE

***Abstract*—Unmanned aerial vehicles (UAV or drone) provide an effective way to get expensive or otherwise inaccessible images and video. This type of data is especially important for traffic analysis as the top-down view from a drone is ideal for gathering data about traffic. Just a T.A.D. (Traffic Analysis Drone) is a road traffic monitoring system that provides a system for capturing and analyzing video. The system uses a drone camera to capture top-down videos of traffics and performs image processing to extract traffic data such as density and interval between cars in a lane. The data is communicated to a data server, and a web interface is available to easily access the data.**

Index Terms – Image Processing, Traffic Analysis, Unmanned Aerial Vehicle

I. Introduction

Traffic is a major issue for society and is only projected to increase. The roadways are regularly filled with cars being used to transport goods and people. Traffic is just a part of life for the average person. In 2014, commuters had an average of 42 hours per year in delays, and very large urban areas (regions with over 3 million people) can have commuters with average delays of up to 82 hours per year [1]. These delays result in wasted man hours and additional costs for operating vehicles. Workers need to plan their daily commute with expectations of delays due to traffic. Truck drivers will encounter even more traffic as they spend their work hours on the road. In addition to consuming man hours, the delays result in increased consumption of fuel, which results in additional costs. The direct cost of fuels and indirect cost of man hours resulted in $160 billion in losses [1].

Their costs are only projected to increase as more cars fill the roads. This is primarily due to the growing economy, workforce, and population. These factors will result in more cars and trucks on the road, resulting in more congestion on the already full roads. The response to the growing traffic has been inadequate as infrastructure are unable to meet the demands of the traffic. By 2020, the total delay time will increase by 1.4 billions hours, totaling 8.3 billion hours in traffic [2]. And, this will result in $192 billion in losses due to congestion, which was $160 billion in 2015 [2]. If this issue is not resolved or slowed, congestion will only continue to grow, and Americans will pay the price with their time and money.

However, a solution is not so simple as building more road infrastructures. These solutions need to be based on data that ensures that there will be less traffic in the area, not just moving the traffic to a new road. Knowing where traffic occurs will help with deducing why it is occurring there and how to best tackle that space. The issue comes with how this data is collected. Current data collection methods primarily capture two types of traffic sensors: “mobile sensor data” and “point sensor data” [3]. Mobile sensor data is similar to google maps as it can capture the data of a single car via GPS, and point sensor data is based on a stationary camera recording cars in a small area [3]. These methods are inadequate as not all cars can be captured in point sensor data and mobile sensor data requires that most cars have the application available. Point sensor data is collected via stationary cameras on the road. These cameras can be easily obstructed and only provide a small sample of the larger traffic picture. Mobile sensor data is primarily done by Google Maps and GPS data from cars. For mobile sensor data to be useable for traffic engineers, almost all cars on the road would need to be recorded [3]. Mobile sensor data and point sensor data are not able to provide the necessary information needed for ideal traffic analysis.

Just A T.A.D. will be able to provide the third type of sensor data: “space sensor data.” Space sensor data is able to provide data about all cars in a large space, such as speed, density, etc and is largely considered to be the most ideal in traffic analysis [3]. T.A.D. will utilize a drone with a bottom-mounted camera to capture video. The video is then processed to provide necessary information based on the space sensor data of the video. Cars are detected and their count and space between cars in a lane (interval) can be measured. The processed data is then sent to a data server where a web browser can be used to access and view the data.

This data is useful to transportation engineers as it gives them data about the traffic situation in an area, and they can formulate a solution based on the data provided. The space sensor data provided is normally hard to obtain. It usually entailed using a helicopter to capture videos of traffic, which then needs to be further processed. Due to the cost associated with this collection method, cameras collecting point sensor data are preferred. T.A.D. will provide not only a means of

collecting space sensor data but also provide analysis of that data. In addition, by using a drone, the general cost of obtaining is comparatively cheaper.

Traffic engineers will be able to use T.A.D. to improve traffic conditions by having the necessary data needed to make decisions. The possibility of less traffic on the roads will mean a better economy as people and goods will reach their destinations sooner. This also means businesses will encounter lower losses Despite these benefits, the unintended consequence is the drone's surveillance capabilities. The drone will be able to collect a substantial amount of video data. Although, traffic cameras already collect a substantial amount of video data, a drone is able to fly anywhere and record data. It is important that the public understands that it is only for traffic data. More concerns will be raised if T.A.D. is used in residential areas as it would have homes in its field of view.

II. Design

A. Overview

T.A.D. is a video capture and analysis system that utilizes a drone to obtain top-down video of traffic to be used for processing and analysis. The general system can be applied to any drone where the module will fit. It consists of a Raspberry Pi computer with a camera and 3G communication module. The drone is operated manually and primarily acts as a medium for transporting this module.

The block diagram, displayed in fig. 1, shows the main components of the T.A.D. system. The drone block consists of a Raspberry Pi and the drone hardware. The drone itself is not necessarily part of the design but does allow for additions, if needed. The drone will be controlled manually in this proof of concept, but can also have pre-planned flight paths for autonomous flight. The Raspberry Pi consists of the 3G dongle, a camera, and image processing software. The camera is used to provide video to the image processing. The 3G out allows for communication to the database.The image processing is primarily concerned with car count and density of the video.This is done through image processing techniques that will be described in detail in a later section. Only the car count and density are sent to the database. The video itself is not sent as 3G does not have the capabilities to easily and quickly communicate video. Once the data is in the database, a web browser can be used to access it. Data is refreshed every 30 minutes, and it can be viewed and downloaded.

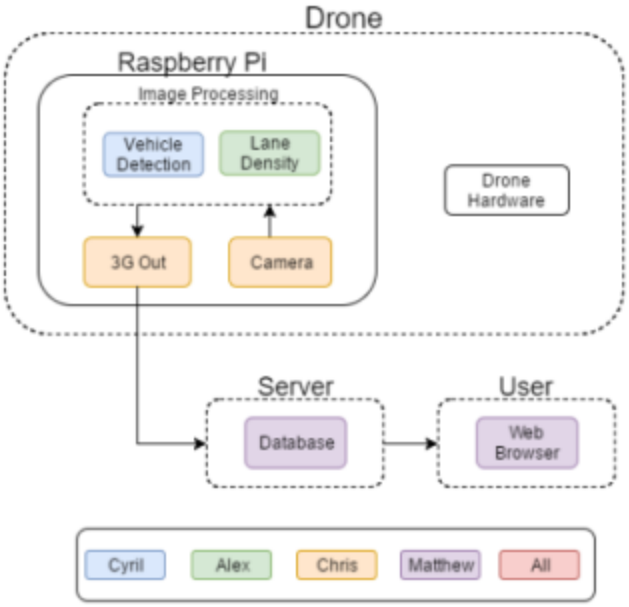

Fig. 1. General block diagram of TAD system

The design alternatives were primarily related to where the image processing would be done and how it would communicate the results to the server. Initially, the video was going to be sent to the server, where it would be processed. This provided more flexibility with the processing power needed for the algorithm. However, the final decision was to have the image processing on the Raspberry Pi as transmitting video via 3G was infeasible. Another alternative was having the drone transmit the data via WiFi and/or have the drone return when it collects all the data/runs out of battery. Additional traffic analysis data can be calculated, but would require additional time and more processing power. A GUI could also be available on the web browser, but would require additional work with integrating a GPS.

Table 1 lists the specifications set forth at the point of MDR and includes portions that will be realized upon completion and integration. The flight time and altitude specification is based on the drone used. In addition, flight time includes the drone flying to the desired observation point. The altitude allows for a wide view of the road to be observed. The final two specifications involve the data transfer and reading. These allow for the data to be transmitted via 3G and guarantees that the data will then be viewable from a web interface.

SYSTEM SPECIFICATIONS

Specification	*Value*
Flight Time	15 minutes
Altitude	500 feet
Cellular Signal Strength	>5 dB
Browser Data Refresh Rate	30 minutes

Table 1. General T.A.D. system specifications

B. Block 1: Raspberry Pi Modules

The Raspberry Pi Modules are made up of both the Arducam OV5647 Video Module (camera) [4] and the Huawei E353 3G USB Wireless Modem (3G dongle) [5]. The camera will be controlled by the Raspberry Pi and will be taking pictures at regular intervals to create the video. After taking these pictures, the Raspberry Pi will run the image processing algorithms on them. Then, the 3G dongle will be used to send the results from the algorithm to the database by using an AT&T 3G Subscription.

The camera weighs 0.3 ounces and is capable of taking photos in resolutions ranging from 480p to 1080p depending on the configuration. At two meters, its field of view is 2.0m x 1.33m. The camera's angle of view is 54 x 41 degrees.

The dongle weighs less than 1.05 ounces and has speeds up to 150Mbps. Since we have chosen AT&T as our data provider, the dongle will provide internet wherever AT&T has service.

To achieve the video, we must interface the Raspberry Pi with ArduCam. Libraries currently exist for this, but code must still be designed in such a way that our system can be implemented.

To achieve the 3G connection, we must interface the Raspberry Pi with the database by using the dongle. To make this happen, we must install the drivers for the dongle, then make sure that the dongle is in dongle mode, instead of usb mode. The dongle stores its drivers locally and must swap modes before use. After that, the SIM card must be added. After configuring the settings and phone number, we may begin using data and testing.

To test the camera functionality, we will take multiple pictures from the camera while it's on the ground. If they are acceptable, then the camera will be mounted to the drone so that test video may be acquired mid flight. If the test video can successfully be used by the image processing algorithms, then we are successful.

To test the dongle functionality, we will use the Raspberry Pi to update the database over 3G while the Raspberry Pi is grounded. If the database is successfully appended with the desired values, then we are successful.

Past skills which will aid us in completing these tests include, but aren't limited to: Programming, Networking, and Hands on Lab courses.

C. Block 2: Image processing

This block (Fig. 1) consists of the majority of the novel engineering work. The broad goals of this block is: given the video feed from the drone's onboard camera, find the cars in any given frame on a chosen road, the spacing between cars in each lane on a chosen road, and the density of cars. The spacing is defined by the distance s_i between cars[3]:

$$s_i = x_{i-1} - x_i$$

The density k is defined by the number of vehicles N observed on a unit length of road L [3]:

$$k = N/L$$

With some tolerance for error with regard to the incomplete spacing of the first and last vehicles, we can restate L as the sum of the spacings between cars [3]:

$$L = \Sigma^{N}_{i=1} x_i$$

Now that we've defined our goals, we can proceed to our approaches and solutions to problems encountered.

We used OpenCV as a starting point, as it is a widely used computer vision library which is cross-platform capable because it uses Python. This allows us to develop code on our main computers and be reasonably sure that the code will work on the Raspberry Pi.

Early on, the group decided that a stationary, purely top-down perspective would generate the simplest data, considering distortion from perspective and optics. This top-down data would also make the calculation of spacing and density simpler, as we would need to account for trigonometric distortion of spacing and density less as well, as in Fig. 2.

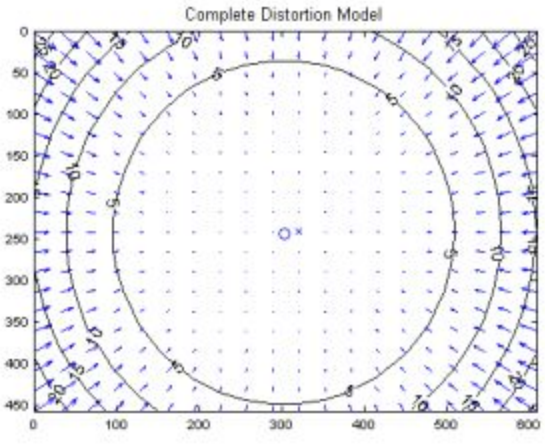

Fig 2. This demonstrates the distortion seen by a camera due to optical and perspective effects. The center of the image is relatively undistorted, but near the edges what should be dots on a grid can be seen as rays at an angle [6].

At first, our attempts were primarily naive approaches using built-in tools in OpenCV, such as edge detection[7] or built-in background subtraction [8]. Simple Canny [7] edge detection was not a viable approach as for any given image, there was too much noise to reliably detect a car versus the pavement, and it was exceptionally sensitive to car color. Background subtraction was more generally more accurate than edge detection in our stationary camera test cases, but even slight movement would make the camera background subtraction fail completely, which is a concern for .

Thresholding[9] was another naive approach, but it was relatively difficult to pick a thresholding value that did not

exclude a significant number of cars in a particular test image, and it would have to be adjusted to local lighting conditions. Even then, in our available test videos, the resulting detection rate was not as high as would be preferable for the trial-and-error involved in picking threshold values.

The lack of reasonable test data was an issue early on, as it made it difficult to determine if our methods were failing due to variables that were irrelevant in our chosen data collection method. For example, we were unable to find traffic data from an overhead stationary drone, and we made due with data collected from a moving drone [10], and data collected from stationary cameras overlooking overpasses [11][12]. This data makes it difficult to account for factors such as lense viewing angle, and for the latter two videos, the angled view prevents an easy calculation of spacing, and by extension, density.

The final, chosen approach took into account the fact that background subtraction was an accurate method of determining motion for a stationary camera. By taking a frame of the video feed every 1/6th of a second, and mapping previous frames onto the most recent frame, we can approximate the effect of a stationary camera, masking all images by the area that would not be visible in the projections for simplicity's sake. Then the newly projected images are fed into a background subtraction algorithm. Since each projection does not take into account the motion of individual cars within the image, then the background subtraction "sees" the cars as moving while stationary objects in the frame do not move. We will now walk through each step of this algorithm in more detail.

Fig. 3. An individual frame of the video [10].

Fig. 3 is one frame of a video used for testing. Subsequent frames are taken at intervals of 1/6th of a second, or 10 frames, to allow for enough movement between frames. These frames are put on a queue of fixed length to use in the rest of the algorithm.

The algorithm used to match images is called SIFT, for Scale Invariant Feature Transform [13].

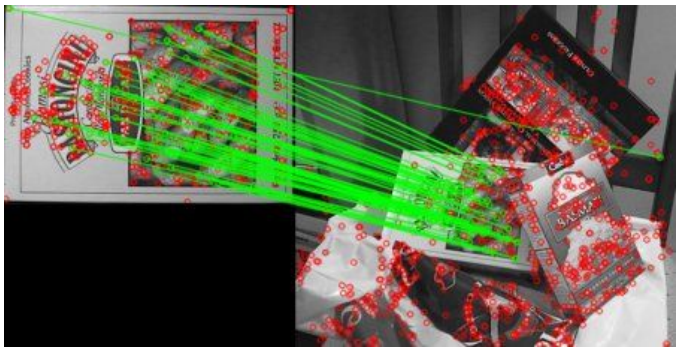

Fig. 4. An example of feature matching in a busy environment [14]

SIFT (as shown in Fig. 4) works by convolving the image with a variable scale Gaussian function in x and y, finding the differences between the various scales, then finding the extrema of these differences to get keypoint descriptors of an image that are invariant to scaling, rotation, and location. Thus, these keypoints in one image can be matched with keypoints in another. SIFT was chosen because it is integrated with OpenCV, allowing us to focus on the larger parts of the algorithm.

Now that the keypoints for each image have been found, we can find the homography matrix that maps each pixel from one image to the other [6]. A homography matrix is a 3x3 matrix that maps lines to lines, and it is defined by the following relation (where x_1 is the x coordinate in the image, x_2 is the y coordinate, and x_3 is affected by nonplanar motion):

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Furthermore, the structure of H affects what properties the transformed image has, such as representing a rotation or a scaling, but for the most part they are irrelevant to our algorithm. This is illustrated in Fig. 5.

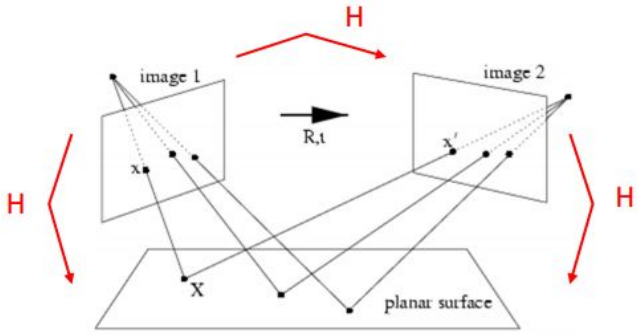

Fig. 5. A graphical explanation of how homography matrices change points in one perspective, the first image, to points in another perspective, the second image [6].

Again, OpenCV provided pre-written code to estimate the homography matrix and apply it from one image to another.

To compensate for the fact that the projected image does not cover the entirety of the newest image (as the drone is moving), a blank mask is also warped by the homography matrix and is combined with a bitwise and operator of previous masks. This mask will be applied over each frame to be fed into the background subtractor. This will ensure that the background subtractor does not see extraneous detail.

The background subtractor is based on the paper "Improved Adaptive Gaussian Mixture Model for Background Subtraction" (Zivkovic, 2004) [15]. This algorithm uses a per-pixel estimation of whether the pixel is part of the background (stationary) or part of the foreground (moving) by a recursive methodology. It is relatively responsive to changes in illumination, and can also estimate whether a change in a portion of the background is actually just a shadow from a foreground object. OpenCV also provided access to and implementation of this algorithm. This background subtractor provides the estimated background as a binary mask, where foreground is white and background is black. After background subtraction, we perform opening transformation on an image [16] to remove noise using a 3x3 kernel. An opening is image erosion followed by dilation. Erosion is a transformation where the kernel "slides" over the image (as in image convolution) and if the area under the kernel centered at a particular pixel does not only contain white pixels, then that pixel is turned black in the resulting image. A dilation does the opposite, if any pixels are white under the kernel centered at a particular pixel, then the pixel is turned white in the resulting image [16].

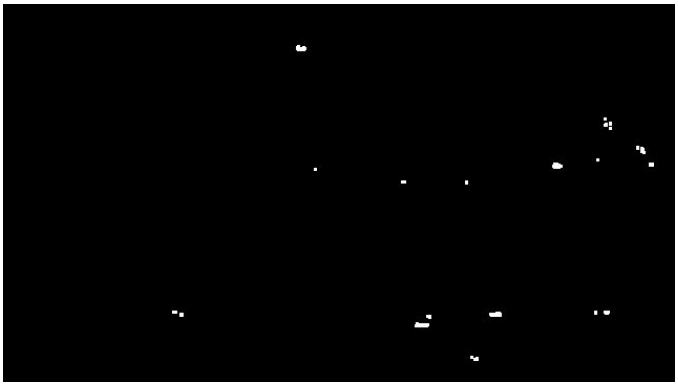

Fig. 6. The output of our background subtraction.

This output (shown in Fig. 6) is given to OpenCV's contour detector [that thing]. This gives us a point description of detected contours, and we can use that to find motion, and presumably cars. This allows us to find Fig 7.

Fig. 7. Detected motion. As you can see, there is some noise and false positives, but on the whole it detects motion accurately. (The far left of the screen falls under the mask mentioned above.)

This algorithm fails for stationary cars, but it is reasonably responsive to poor camera conditions in most of our test cases. In this particular case it detected 28 out of 30 cars in the lower highway, or 93%. We can improve the false positive rate if we assume that we know a few things about the drone's position, the position of the road, and the orientations of the road and drone, illustrated in Fig. 8. This information is all available from the DroneKit API that will be used to control the drone [17].

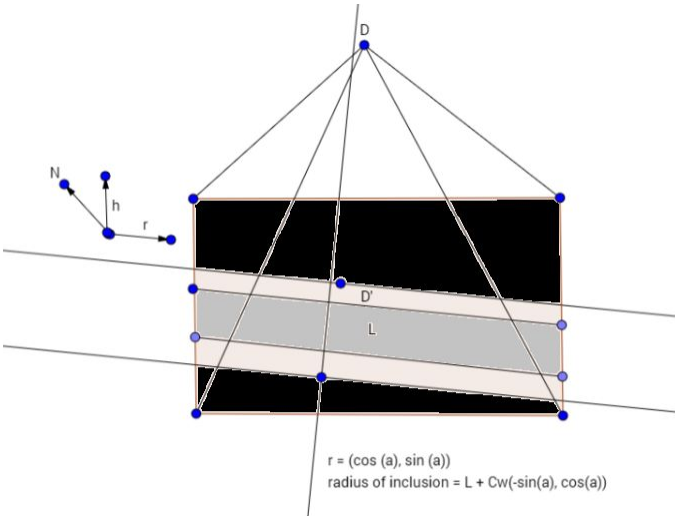

Fig. 8. Where *a* is the compass direction of the road, and *h* is the compass direction of our drone, L is a chosen GPS coordinate in the road, D is the GPS coordinate of our drone, we can find a perpendicular vector to *r*, and then use this vector to create a mask for our images. This mask will prevent the motion detection algorithm from capturing as many false positives.

We can also use similar methods to find the spacing between detected cars by lane, shown in Fig. 9.

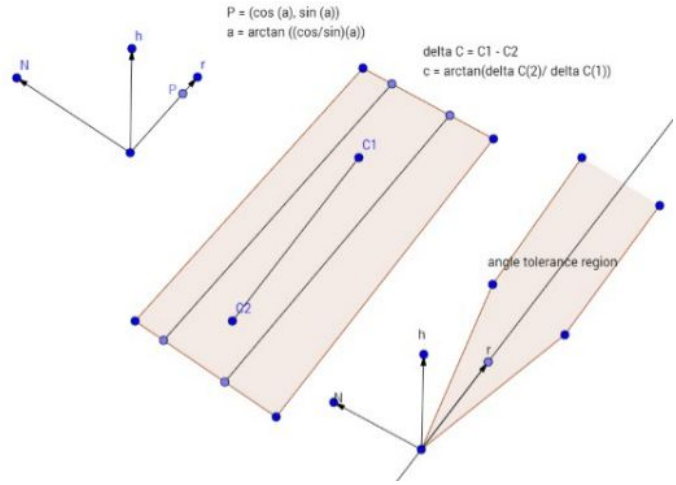

Fig. 9. Given the point descriptions of detected cars, we can find their centroids [opencv contour detection]. And given the orientation of the road, we can compare the angle of the road to the angle of the slope generated by two centroids.

Fig. 10. An implementation of the above lane detection algorithm.

However the latter two systems have not been implemented in code yet, as the drone is not fully functional. One of our main priorities over the break is implementing these systems in code.

D. Block 3: Data Server and Web Browser

This block represents the implementation of the data server and web browser. From a high level point of view the processed data from the Raspberry Pi would be transmitted to the data server via 3G. The data, primarily car density and interval (spacing between cars), would then be sent to the database by the 3G dongle. The web browser would then query the database and provide the end user an easy to use and more aesthetically pleasing UI. That being said the goal of this block is to implement this system from the backend (database) to the web browser (frontend) utilizing the MEAN stack of web development. The MEAN stack represents the technologies utilized in this popular branch of web development which are mongoDB, ExpressJS, AngularJS, and NodeJS [18].

Starting from the bottom-up, the database is hosted on mongolab, a cloud based database host [19]. The primary benefits to hosting the database on the cloud rather than on the Raspberry Pi is to take as much processing load off the Raspberry Pi as possible due to the intense image processing already taking place on it. This would be more beneficial as opposed to running the image processing on a server as transmitting images over 3G would most likely cause bandwidth issues. The hosting service is also free up to 500 MB at any given time which should be more than enough as only numbers are being stored. In addition, mongolab also provides a low level visualization of the database content for manual entries which aids developers as well as connection information to the database either by the shell or mongoDB URI for smooth integration into the code involving data transmission over 3G. The database itself utilizes mongoDB, a NoSQL database system that stores its contents as JSON documents which allows for varying structure [20]. This allows for dynamic schemas meaning the parameters and variables setup in the initial database implementation can be changed at anytime. This flexibility favors all phases of this project from development and testing to final staging as a database bottleneck is not present.

In order to connect the backend (database) with the frontend, middleware and server side technology is required. ExpressJS and NodeJS are both backend technologies that will enable this connection and framing of the web application with the database. NodeJS is a lightweight backend runtime environment used to build the raw components of the web application in terms of server side activity such as connections to the application [21]. The aforementioned connections also include the connection to the database which utilizes its own driver for mongoDB Driver API [22]. ExpressJS works in hand with NodeJS by creating the framework for the web application. More specifically, routes are created in which any type of HTTP request to/from the web application will need to be redirected by ExpressJS in order for the web application to service said requests as seen in this example in the ExpressJS 4.X API [23].

With the backend setup, the final implementation of the MEAN stack is AngularJS. AngularJS can be thought of as an extension of HTML in which it allows for dynamic views as HTML in its core was made for static views in terms of web pages. The web application to this date allows for CRUD (Create, Read, Update, Delete) operations with respect to the database contents. AngularJS allows for non intrusive implementation of this technology with its dynamic front end framework [24].

Further research into these technologies will be required in order to implement the planned automatic update feature and any other useful feature to the web application. Currently, an “Update” button exists on the website which needs to be pressed in order to update the database contents to the most up

to date view (not to be confused with the refreshing of the entire web page). The functionality of the auto-update feature would be tested by manually entering data into the database via mongolab and viewing the website to see the new entry being displayed properly. Basic programming knowledge from ECE 242 Data Structures & Algorithms and exposure to databases from ECE 373 Software Intensive Engineering applies to work done in this block.

III. Project Management

MDR Goals

Goal	*Completion*
Car Counting	100%
Interval Spacing	100%
Camera and 3G	25%
Database and Web UI	100%

Table 2. Proposed MDR deliverables and level of completion

The MDR goals are displayed in Table 2, above. These goals were largely completed: car counting, interval spacing, and database and web UI. The only goal that had issues was the camera and 3G. This was largely due to complications with establishing the 3G but will be completed in time for CDR along with other functionality. These subsystems were tested individually and will be integrated.

Our team meets weekly with our advisors, Professor Hossein Pishro-Nik and Professor Daiheng Ni (CEE).. We also meet separately as team to talk about individual work and how we will integrate our pieces for the final design. We also meet when ordering parts and had extensive discussions when initially selecting the drone. The image processing was primarily handles by Alex and Cyril. Cyril handled the high-level image processing for detecting and counting cars. Alex handled image processing for detecting intervals between

Fig. 11: Gantt chart of work done and planned work for the future

cars and developing algorithms to offset camera movement. Chris was responsible for the Raspberry Pi and its modules. This included setting up the Raspberry Pi's OS, the camera, and the 3G attachment. Matt was responsible for the data server and creation of the web page that displays the data from the drone. In addition to each member's individual components, all team members were responsible for completion of the project and assisted each other when necessary.

Figure 11 displays a Gantt chart of work that has been completed as well as work that will be done in the future for CDR and FDR. The reports, presentation, and demo will be handled by all team members.

IV. Conclusion

Our team accomplished most of our goals for MDR, but encountered issues with the 3G module. We did not expect this component to have as many issues as it did. In addition, the sim card was delayed and further postponed implementation of this overall block. The other component of this block was the camera, which was not implemented due to the 3G issues, as mentioned. Otherwise, the image processing and database/browser goals were completed. Despite the initial set-back of a second PDR of a new project, we believe significant progress has been made, but there is still a lot of work that needs to be done by this point.

For CDR, we plan on finishing basic integration of systems. Primarily this will be done to ensure that test images can be taken with the drone, and those images can be analyzed correctly by the image processing algorithms. The drone's flight capabilities will be tested and Matt will practice flying the drone to ensure his piloting skills are sufficient. The image processing software will be put into the Raspberry Pi, and the camera will need to be able to provide its video feed to the image processing software. For the CDR presentation, we hope to be able to demo the image processing software working with our own test images.

The difficulties ahead are primarily with integrating the image processing software with the Raspberry Pi and being able to get our own test data. The primary concern is that the software requires more processing power than the Raspberry Pi is able to provide. This will result in significant delays in the processing of the images, so the image processing will not be real-time with the video. In addition, being able to fly the drone near roadways can be dangerous. Discussion with our advisors, especially Professor Ni, will need to be done to ensure it is safe and legal to fly the drone in certain area.

Integration of systems in order to get test images is our main goal for CDR. Completion of this goal will provide a clearer picture of our overall progress. It will help display changes needed in the image processing software and possible issues with the Raspberry Pi and its components. All team members will work to ensure the system integration and testing is completed and done in a timely manner.

Acknowledgment

We would like to thank Professor Pishro-Nik for meeting with us week-to-week and helping us stay on track. In addition, Professor Pishro-Nik generously purchased our drone for us, which would have been outside of our budget. We would also like to thank Professor Ni for helping us establish the project idea and providing information about traffic analysis concepts. We would like to thank Professor Hollot and Professor Koren for their insight and feedback in establishing our project. We would also like to thank Mr. Fran Caron for his assistance in the SDP lab and in ordering our parts for our project.

References

[1] E. Dooley. (2015, Aug. 26). *Here's How Much Time Americans Waste in Traffic* [Online]. Available: http://www.abcnews.go.com.

[2] D. Schrank, et al, "2015 Urban Mobility Scorecard," Texas A&M Transportation Inst. and INRIX, College Station, TX, Aug., 2015.

[3] D. Ni, *Traffic Flow Theory A Unified Perspective*, Amherst, MA, 2015, pp. 14-46.

[4] New Low Cost OV5647 Mini Camera Module for Raspberry Pi Now Available (2017, Feb 6). Arducam. [Online]. Available: http://www.arducam.com/lowcost-raspberry-pi-mini-camera-module/

[5] E353 Specifications (2017, Feb 6). Huawei. [Online]. Available: http://consumer.huawei.com/bd/mobile-broadband/dongles/tech-specs/e353-bd.htm

[6] C. Gava, G. Bleser, *2D Projective Transformations*. [Online]. Available: http://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws11-12/3DCV_WS11-12_lec04.pdf

[7] Canny Edge Detection (2016, Dec 21). OpenCV. [Online]. Available: http://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html

[8] Background Subtraction (2016, Dec 21). OpenCV. [Online]. Available: http://docs.opencv.org/trunk/db/d5c/tutorial_py_bg_subtraction.html

[9] Image Thresholding (2016, Dec 21). OpenCV. [Online]. Available: http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html

[10] Simon Laprida. Aerial drone scene of jamed [sic] highway. [Online]. Available: https://www.videoblocks.com/video/aerial-drone-scene-of-jamed-highway-top-view-of-traffic-in-the-road-city-rush-hour-camera-moves-gently-showing-the-city-jam-ynwv5gg/

[11] Supercircuits. (2014, Aug 20). Alibi ALI-IPU3030RV IP Camera Highway Surveillance. [YouTube video]. Available: https://www.youtube.com/watch?v=PJ5xXXcfuTc

[12] mjrzeman. (2011, Sep 14). HTC Sensation FULL HD 1080p Video Sample (highway traffic). [YouTube video]. Available: https://www.youtube.com/watch?v=wWLAc6mdJrs

[13] D. Lowe. (2004, Jan 5). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*. [Online]. Available: http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[14] Feature Matching. (2016, Dec 21). OpenCV. [Online]. Available: http://docs.opencv.org/trunk/dc/dc3/tutorial_py_matcher.html

[15] Z. Zivkovic. (2004). Improved adaptive Gaussian mixture model for background subtraction. *Proceedings of the International Conference on Pattern Recognition*

[16] Morphological Transformations. (2016, Dec 21). OpenCV. [Online]. Available: http://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html

[17] DroneKit-Python API Reference. (n.d.). 3DR. [Online]. Available: http://python.dronekit.io/automodule.html

[18] The MEAN Stack: MongoDB, ExpressJS, AngularJS, and Node.js (2017, Feb 5). MongoDB. [Online]. Available: https://www.mongodb.com/blog/post/the-mean-stack-mongodb-expressjs-angularjs-and

[19] Quick-Start Guide to mLab (2016, Dec 22). mLab. [Online]. Available: http://docs.mlab.com/

[20] NoSQL Databases Explained (2016, Dec 22). MongoDB. [Online]. Available: https://www.mongodb.com/nosql-explained

[21] Node.js v6.9.2 Documentation (2016, Dec 22). Node.js. [Online]. Available: https://nodejs.org/dist/latest-v6.x/docs/api/

[22] Node.js MongoDB Driver API (2016, Dec 22). mongodb. [Online]. Available: http://mongodb.github.io/node-mongodb-native/2.0/api/

[23] ExpressJS 4.X API (2016, Dec 22). ExpressJS. [Online]. Available: http://expressjs.com/en/4x/api.html

[24] AngularJS API Docs (2016, Dec 22). AngularJS. [Online]. Available: https://docs.angularjs.org/api