Smart Traffic Camera System

Goran Goran, Arvind Govinday, Jonathan Li

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—Traffic lights historically have had minimal innovation. In the past few years though, we have seen a growth in the usage of traffic light cameras to detect inbound/outbound traffic. Our project aims to utilize modern computer vision and deep learning technologies to monitor traffic cameras and detect vehicle collisions. The goal is for our system to be able to detect vehicle collisions on the road from simple cameras mounted on traffic lights and relay detailed information including video recordings to local authorities. Additionally, we will be using this data to model traffic effects caused by collisions.**

***Index Terms*—Computer Vision, OpenCV, Pytorch, Resnet Architecture, Pytorch, Image Processing, HERE**

I. Introduction

Many traffic light networks in the U.S. are beginning to modernize. Researchers in the past few years, including Carnegie Mellon University faculty member Stephen Smith, have been working to integrate a plethora of sensors such as cameras and radars into traffic light systems to make smarter traffic light management decisions [1]. With this growth in the use of these traffic cameras, our project aims to integrate software based computer vision and deep learning systems to detect vehicle collisions and report these incidents to the relevant authorities. It is notable that with investments already being made in hardware additions (traffic light cameras) to traffic light systems, our project is entirely software based. Thus theoretically it requires minimal costs and infrastructure changes from the client's perspective.

We aim to implement collision detection using an algorithm based on the position and velocity of the vehicles in frame. We will be using an image processing pipeline between the frames of traffic videos to identify these parameters of any moving objects in frame and when a crash between moving objects has occurred. To classify with certainty that collisions are occurring between vehicles, we are using the resnet architecture to classify objects as vehicles. Additionally, we plan on recording car crashes from before the crash, during the crash, and after the crash for a period of 5 minutes using OpenCV and video buffers. In response to crashes, we plan on communicating information about the size of the crash, location, and lane closures to the web server. Using this data, along with a routing application program interface (API), we will provide drivers optimized routes to their destination as long as they input their source and destination on our webpage, With this system, the client gains immediate notification of a vehicle collision, recordings of the vehicle collision in full, and predicted traffic disruptions due to the vehicle collision. This information can be incredibly valuable to first responders and city management and because this system is entirely software, it has a minimal implementation cost.

II. Use-Case Requirements

In this section, we have included requirements necessary from the client's perspective to utilize the system we are developing and system use case requirements. They are split into two parts. The requirements are listed below:

A. Camera Requirements

We want to make the camera requirements realistic to what one might expect from an average traffic light. Therefore, the client must not need more hardware than a 1280x720p webcam that can record at a minimum 24 fps and a Linux powered device capable of running Python 3 and OpenCV. The client must have their camera at an elevated and standstill position on a traffic light facing the street. To ensure proper connectivity, the client's camera connected linux device must have a reliable internet connection capable of uploading at a rate of 8 mbps while downloading at a rate of 1mbps to support web server communication. This is important as we intend to use a web server to put together the different aspects of this project. Furthermore, the client must have a web server capable of ingesting content from camera interfaces such as Amazon's EC2 instance.

B. Crash Detection Requirements

It is important that we do not detect crashes when none have in fact occurred. This is because our smart traffic system begins to reroute traffic, alert law enforcement and has the potential to cause serious disruption if triggered for a false alarm. Therefore, we would much rather not detect some crashes but be very precise on how we handle false positives. As requirements, the system must be able to detect vehicle collisions with a false positive rate of 0.1% where failure is defined as a detected collision that is not a genuine collision. On the other hand, the system must be able to detect vehicle collisions with a false negative rate of 2% where failure is defined as the omission of vehicle collision detection in the event of a vehicle collision.

C. Web Server and Video Requirements

In the event of a detected collision, we want the succeeding events to happen timely. The system must record the minute before the collision, and the next 4 minutes upon collision

detection, and upload the recording to the web server within 6 minutes. The system must also notify the web server within 10 seconds of the detected collision. With regard to the video, it must be 640x480 resolution at 24fps specified to the web server with 30 seconds of video recording completion. In the event of a detached collision, the recorded video must not contain any skipped frames. In the event that skipped frames are detected, they shall be labeled in the data packet sent to the web server. Finally, the system must have a 24/7 uptime for 1 month intervals

D. Rerouting Requirements

In the event of a collision, we want to be able to rapidly update the paths of our drivers, so that there is as little impact as possible on their trip. We would like the location, size of crash, and relevant route data to be transmitted to our web server within 5 seconds of the actual crash taking place and being confirmed. We would then like our rendered map on our web server to update within 10 seconds of receiving this data, so that the consumer can react in a timely manner to these new changes.

III. Architecture and/or Principle of Operation

Our architecture involves multiple components that interconnect with one another. We have included Fig. 1 to demonstrate the connections between the multiple components.. Fig. 1 which is located at the end of this design report demonstrates the high level architecture that we are using. It includes modules such as the opencv recording module, object detection module, resnet module, traffic rerouting module, etc. Details and specifications for each block are provided in the implementation section of the design report.

IV. Design Requirements

A. Web Server and Video Requirements

Similar to the use case requirements, we want our design to ensure efficient video logging. Therefore, the recording module shall not skip frame processing given an ingestion rate of 24 frames per second. This implies that frame processing and saving must occur in 41.67 milliseconds. Note that other modules run concurrently so this design requirement is strictly in the context of the recording module. To ensure that we are only keeping the relevant frames we need and not risking running out of our camera hardware's storage space, the recording module shall save frames in a FIFO style queue and maintain the last 1 minute / 1440 frames processed.upon receiving a CRASH_DETECTED signal, the recording module shall copy the 1440 frame FIFO queue. The recording module shall receive signals in a timely fashion (must not process > 1 frame when a signal is raised and before handling the signal). As mentioned earlier, the video processing module shall not take > 41.67 milliseconds to process individual frames to prevent "runaway" processing in the context of inbound video frames.

B. Image Processing & Classification Requirements

As with the web server requirements, we want to ensure efficient image processing and classification. In addition to our use case client side requirements, we want to make sure that 99% of all moving vehicles in any given frame in the video data stream are correctly detected, classified, and matched with a correct-to-form boundary. We expect this high classification because our project specifically only deals with vehicle on vehicle crash, so we want to eliminate other moving objects that are not vehicles from our system. With regard to the speed of the classification, the resnet architecture shall take no greater than 250 ms per vehicle in classification of vehicles. In terms of what makes a correct - to - form boundary, we want to make sure 90% of the pixels enclosed within the contour boundary are aligned with the vehicle object and not the vehicle's surroundings.

V. Design Trade Studies

Some specific quantitative requirements of our subsystems have trade offs that are worth exploring. This section discusses such metrics, and how the trade offs can be studied and experimented with to come up with the best design for our use cases.

A. Crash Detection Speed and Orientation Metrics

A very important use case requirement is detecting vehicle collisions with a false positive rate of 0.1% and detecting vehicle collisions with a false negative rate of 2%. When two vehicles are detected to be very close together, we must make sure that they have indeed crashed and not simply happen to be very close together for a benign reason. Two parameters that we can fine tune to make sure we are meeting our requirement is the speed and orientation required of the vehicles suspected of crashing. With regard to speed, there are two parameters to tune- how fast must the cars be going before the crash and how much did they slow down after the crash? Intuitively, the cars should be going at a decent speed before being suspected of crashing, and then should have slowed down more or less to a complete stop after stopping. With regard to the orientation- looking at the direction of travel in the frames prior to being suspected of crashing, are the motions predicted to intersect- and specifically we must tune how much leeway between the suspected crash location and the intersection point is truly representative of a crash instead of a near miss. These three parameters can be fine tuned, and will result in trade offs with regard to the efficacy of our crash detection system. However, this is an optimization problem. A basic approach is to make an initial reasonable guess for all three metrics, and then keep 2 constant and vary 1 the first

until an optimum is reached, and similarly repeat for the other two parameters.

B. Tradeoffs regarding vehicle classification networks

In our efforts to classify collided objects as vehicles, we have decided to use efficient deep learning architectures for the classification problem. Originally we considered Mobilenet v2, Resnet 50, and Resnet 34. All networks had their own pros and cons as listed below

1) Mobilenet v2

Mobilenet v2 is a relatively lightweight network. In our experimentation we found that the classification accuracy was relatively low and required many epochs to train. In order to get a well trained network we required > 50 epochs at ~10-20 minutes per epoch which seemed unreasonable for our context.

2) Resnet 50

Resnet 50 was the next deep learning network we considered. The architecture was relatively simple to set up and after around 15-20 epochs at ~15 minute per epoch, the network seemed to classify objects as vehicles with a solid degree of accuracy. The only problem with Resnet 50 is that it seems to miss our design requirements of classification taking no greater than 250 ms. This is due to the exponentially large number of convolutional layers needed as the Resnet size increases.

3) Resnet 34

Resnet 34 was the final deep learning network we considered. The architecture was relatively simple to set up as it is essentially the same Resnet 50 but with many fewer layers and smaller layers that carry less data. Around 15-20 epochs at ~10 minute per epoch, the network seemed to classify objects as vehicles with a solid degree of accuracy. In fact the accuracy was very similar to resnet 50. We believe that this is due to the low resolution and "simple design" of vehicles. As we are classifying the change that an object is a vehicle, the network can be very simple and still function well.

C. Open Street Maps vs Google Maps vs HERE

In our efforts to find the best way to implement dynamic rerouting for our drivers, we were presented with a multitude of options. Originally OSM, Google Maps, and HERE all seemed like potential options for implementing our rerouting, as all three were prominent map API's that have been around for quite a while.

1) OSM

We at first attempted to make use of OSM to implement our rerouting plans, but although it seemed to bear much fruit in the beginning, we quickly hit a wall. Although with OSM we have the flexibility of running actual spatial network analysis, and are able to express geographical locations and roads as a network of weights and edges, it was not quite what we needed for our project. The difficulty in rendering maps for a variety of locations, and honing in on specific intersections to lower the weight on became a big enough issue that we moved on to exploring other avenues.

2) Google Maps

We then moved onto attempting to make use of the Google Maps API, which seemed to be a very fleshed out and viable choice. In reality we quickly discovered that there currently is not any method in place for developers to develop route avoidance. Although we can have the API create routes based on a variety of factors, there is no way for us to have it avoid specific intersections and/or streets.

3) HERE

Finally we landed on the HERE Routing API. HERE technologies is a company dealing with mapping, location, and related automotive services. They are the ones behind most of the built in mapping systems placed in cars. The HERE Routing API allows us to block out entire areas based on input coordinates, and optimally route drivers around the crash location. Although we have less flexibility when compared to OSM in how we are routing the drivers, the ability to have route avoidance is worth the trade off. Also the built in routing algorithm for the HERE Routing API is already close to ideal, and should not deviate much from what we also expect the optimal pathing to be.

VI. System Implementation

A. Image Processing & Object Detection

The first aspect of our project is the image processing pipeline for object detection. Our data set is a video feed of the Sherbrooke / Amherst intersection in Montreal. We first apply frame differencing, and then image thresholding at 30 (out of a maximum value of 255 in a grayscale image). This isolates only the moving objects between frames, as the stationary areas will not have large values after frame differencing. To make the moving areas more prominent, we apply image dilation, which is a convolution operation with a kernel 3x3 matrix of all ones. Calling this kernel matrix k, for two successive frames F_1 and F_2:

$$(1.)\ (\mathbf{1}_{(F2 - F1) > 30}) * k$$

Here, **1** is the indicator function- which results in a one when the thresholding condition is met at a given pixel, and 0 otherwise. At this point, we have blacked out all stationary areas and have made the moving areas prominent. To get the locations of the moving areas, we need to plot the contours. Using OpenCV's findContours function, we can get the overall shape and boundaries of the moving areas. In other words, we will have visible boundaries of the cars, pedestrians and other moving objects in frame.

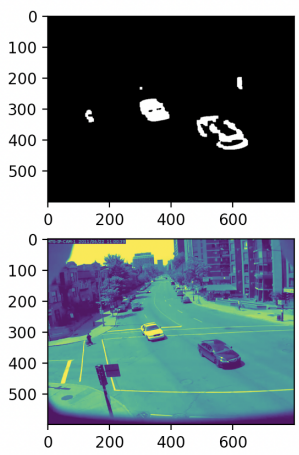


Fig 1: Expression (1.) implemented on consecutive frames

B. *Deep Learning Vehicle Classifier*

Since we are primarily concerned with vehicle on vehicle crash detection, we will be using a deep neural network to classify the boundaries of the moving objects as vehicles. This will be using the Resnet Architecture. Combined, the image processing and deep learning pipeline together must make sure to meet our requirements of detecting vehicles with a 99% success rate. This means that 99% of vehicles that are seen in frame are detected, and their contours are matched properly to the vehicle outlines. Due to hardware constraints, we plan on using a Resnet 34 architecture that is configured as shown below.

layer name	output size	18-layer	34-layer	50-layer	101-layer
conv1	112×112	7×7, 64, stride 2			
conv2_x	56×56	3×3 max pool, stride 2			
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax			
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9

Fig 2: Resnet layer designs for Renet 18 through Resnet 101. [3]

C. *Crash Detection Formula and Actions*

Now that we have located the boundaries and location of each vehicle, we will formulate a method for crash detection. The crash detection algorithm begins when the boundaries of two moving vehicles overlap. Once this is detected, we want to know if this is a crash, or simply two vehicles that happen to be very close to each other for another reason. We will track the direction of movement as well as the change in speed of the vehicles[4]. Our frame images are dimensioned at 600 x 800 pixels. Therefore, our notions of velocity- both magnitude and direction- will simply be the movement of the "center" of the boundary of a vehicle from one frame to the next. The "center" of the vehicle is the average of all the plotted contours that make up the vehicle boundary. We will track the movement of the overlapping vehicle from 150 frames before they began to overlap. We will see if their directions of motion are in an orientation and speed that will lead to a crash, as well as if the vehicles came to a stop once they did overlap, by looking at 150 frames after. The video is 30 fps so this is the equivalent of looking 5 seconds before overlap and continuing to monitor for another 5 seconds. The logic looks as follows:

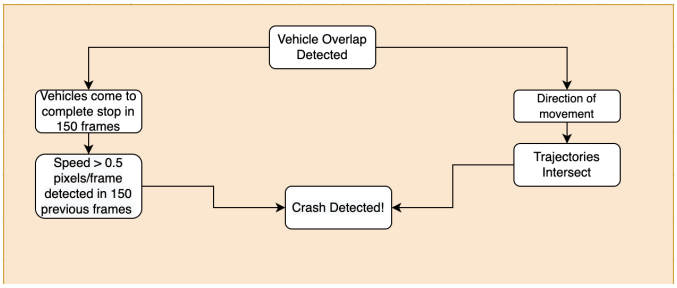


Fig 3. Crash Detection Logic Flow

Combined, the image processing, deep learning and crash detection algorithms together must make sure to meet our requirements of detecting vehicle collisions with a false positive rate of 0.1% and detecting vehicle collisions with a false negative rate of 2%

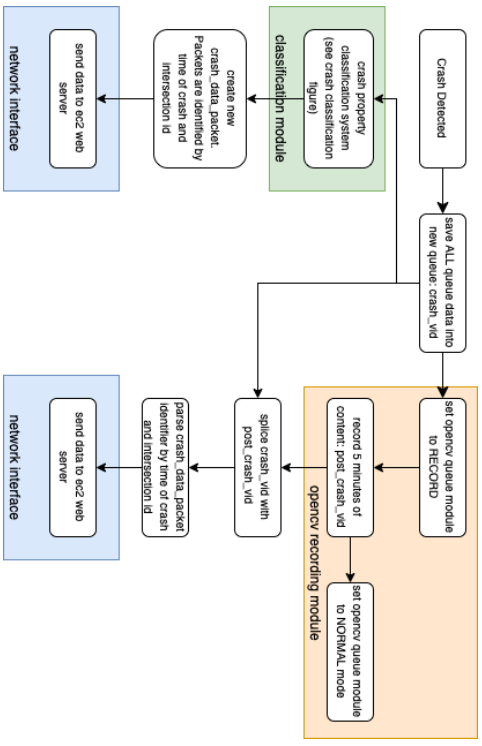


Fig 4: The post crash detection actions are shown here. Multiple modules respond to the detected crash.

D. Web Server

We are planning to use Apache linked with Amazon EC2 for our web server. This platform is highly scalable while also very inexpensive. In addition to Amazon EC2, we plan on using Django to implement the web interface that clients will utilize. Django also has a built in SQL-Lite based database that we will use to store crash information.

E. Rerouting

Now that we have received both the location and size of the crash, we are now able to use that information to aid the driver in avoiding the area. To calculate directions between two coordinates, without traveling through a specific area or other more specific restrictions, we make use of the HERE Routing API. Based on the size of the crash we define custom penalty parameters to determine which road attributes to use in our route calculations, allowing us to give drivers the fastest route to their location.

We will use HERE's mapping application to render the geographical representation of the route together with the map data, so that the route is displayed on a map. Drivers will be able to access this map, as well as input their current location and destination, on our web server. This means that drivers will be easily able to always receive current and up to date rerouting information straight from any internet connected device that they prefer.

Fig 5: Example of route avoiding an area

F. Video Logging

The video logging interface utilizes a FIFO style queue. When frames are processed, they are added to a fixed size FIFO queue of size 1440 frames (1 minute of footage at 24 frames per second). We have implemented the functionality of freezing snapshots of the queue in the event of a detected collision, the queue is copied and saved for video exporting. This can be shown in Fig 4. which covers post crash detection actions.

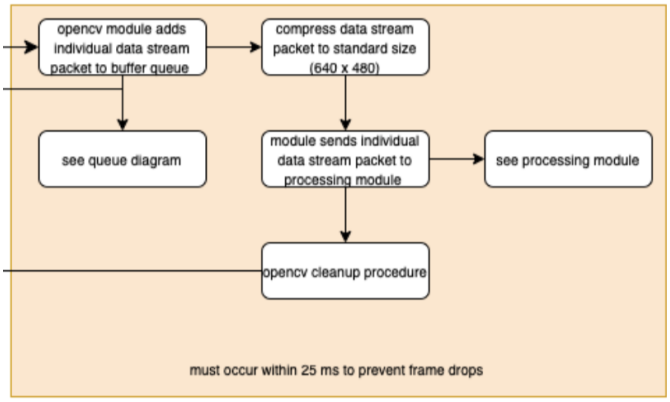


Fig 6: Frame processing and logging shown

VII. Test, Verification and Validation

To verify that we are meeting our use-case and design requirements, we will conduct the following tests.

A. Tests for Crash Detection

We want to test that we are detecting vehicle collisions with a false positive rate of 0.1% and detecting vehicle collisions with a false negative rate of 2%. While the overall pipeline is being trained and refined on the Montreal intersection video stream, it is not efficient to use a video stream with very intermittent crashes to specifically test for crash detection. Instead we can use traffic light car crash clips from YouTube to ensure that our algorithm is working as intended and up to specifications. In order to attain the video from YouTube, we can make use of screen recordings.

B. Tests for Post-crash Timeliness

We have two conditions for timeliness once a crash has been detected: the web server is notified within 10 seconds and the five minute recording is subsequently uploaded - we will allow slack and round to the next minute so 6 minutes allocated for uploading. To test this, we will simply begin a timer the moment a crash is detected and monitor the web server. If our queue based implementation of video logging is working as efficiently as intended, the time requirements should be met and the results should show up as intended on the web server.

C. Tests for Rerouting Efficacy

We would like our route data to update within 5 seconds of the crash, and then have the geographical representation of the map that we are rendering on our web server to update within 15 seconds of the entire crash. We would also like to make sure that the rerouting module is not forcing our drivers on long convoluted routes. We will make sure these conditions are all satisfied by testing our design on simulated crashes, and timing how long is required for our map to fully update and

render our route after being given specific locations. We will also look at the newly created route and see if it makes sense for a driver to take those specific turns in order to avoid the crash area.

VIII. Project Management

A. Schedule

Our schedule is based around first creating individual components, and then figuring out ways to connect them together. The web server acts as the glue that helps us integrate all of our different pieces, and have them work together hand in hand. Our crash detection, rerouting module, queue based recording system, and webserver module are all distinct entities that can be developed in parallel all the way until they need to be connected to the web server. The web server connection can be mimicked in testing by providing data to our different modules and seeing how they operate under different conditions, The emergency response notification system on the other hand heavily relies on the existence of the web server, and must wait until that has been fully set up before we can see any progress in that regard. The schedule we have leaves time at the end for both integration, testing, and reworking based on feedback we receive from our demo before the final due date.

Our Gantt chart can be seen in appendix A.

B. Team Member Responsibilities

While all group members are responsible for collaborating well on all aspects of the project, the work has been split up to allow members to focus on their areas of interest or expertise. Arvind Govinday shall primarily focus on the image processing pipeline, and getting the crash detection working. Jonathan Li will primarily focus on the video logging and web server aspects of the projects. Jonathan also has a lot of deep learning experience, and so will focus on the vehicle classifier as well. Goran Goran will primarily focus on rerouting.

C. Bill of Materials and Budget

Bill of Materials and Budget are included at the end of the design project report.

D. Risk Mitigation Plans

The biggest risk to our project is that the image processing and classification aspects do not work. The rest of the modules very much rely on these aspects working, so it would put us significantly behind if this part did not work as intended. However, the problem of vehicle detection has been widely studied, and there exist numerous open source pre-trained classifiers for us to use on images. If we are unable to get our own vehicle detection and classification system working, we will simply use one of these open source classifiers and work on the rest of the project. With regard to web server and rerouting, there is less risk as these modules are more isolated and simpler to accomplish.

IX. Related Work

There are currently no other projects that we are aware of that integrates all of these features into a traffic light module.

A lot of research has been done in the field of both crash detection, and traffic flow rerouting, but nowhere in our searches were we able to find anyone that was attempting to integrate these two components into the same system in quite the same way. We place most of the responsibility of following through with the rerouting changes on the drivers themselves, having them access a website and follow specific directions..

The closest project to our design that we have been able to find is Surtrac [2]. Surtrac is an AI-based adaptive traffic management system designed to better control traffic flow. It makes use of sensors to identify approaching vehicles, calculate their speed and trajectory, and adjust a traffic signal's timing schedule as needed. Surtrac was coincidentally spun out from a Carnegie Mellon Institute project.

Although both our project and Surtrac are smart traffic lights, there are a few distinct differences. Our main focus is detecting large disruptions to traffic flow caused by crashes and best aiding drivers in avoiding it, as well as providing documented footage on the crash itself. Along with notifying the proper authorities of the location and approximate size of the crash, in efforts to reduce emergency response time. Our traffic lights are all stand alone systems, different from the decentralized network of smart traffic lights that form Surtrac, and have no control over the timing of traffic light signal changes in other areas.

X. Summary

Our goal for our smart traffic signal system has always been two fold, to both mitigate loss in productivity caused by car crashes, and to lower emergency response time to crashes. Therefore it is important that we are able to detect and record the crashes accurately as they occur, understand how that affects individual drivers' routes, and report these occurrences to law enforcement with a high degree of confidence in their occurence.

Challenges in achieving these requirements lie in the fact that we must be able to do all of these tasks at both a high speed and with little to no error. If we are not able to detect these crashes fast enough and cut down on emergency response time, there may as well not be a smart traffic light in that location in the first place. We also do not want to accidentally report crashes that do not actually exist, resulting in traffic slow downs as we reroute cars away from a perfectly viable street. Reporting crashes that do not exist would also

result in a significant waste in emergency response resources.

Glossary of Acronyms

API - Application Program Interface
OSM - Open Street Maps
Resnet - Residual Learning Network

References

[1] Austin, P. L. (2019, January 21). *Want to fix traffic? try smarter signal lights*. Time. Retrieved March 3, 2022, from https://time.com/5502192/smart-traffic-lights-ai/

[2] Technologies, Rapid Flow. "Surtrac: Intelligent Traffic Signal Control System." *Rapid Flow*, https://www.rapidflowtech.com/surtrac.

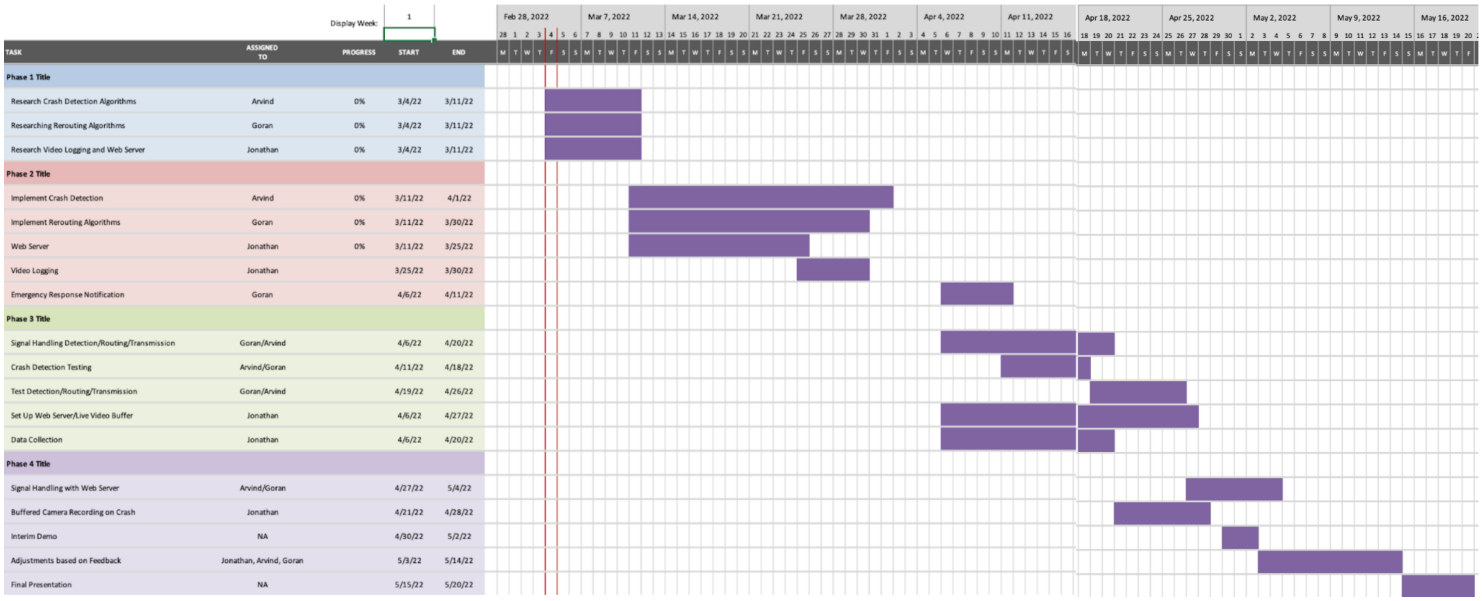
[3] he, K., Zhang, X., Ren, S., & Sun, J. (n.d.). *Deep residual learning for image recognition - arxiv.org*. Retrieved March 3, 2022, from https://arxiv.org/pdf/1512.03385.pdf

[4] Xiaohui Huang, Pan He, Anand Rangarajan, and Sanjay Ranka. 2010. Intelligent Intersection: Two-Stream Convolutional Networks for Real-time Near Accident Detection in Traffic Video. ACM Comput. Entertain. 9, 4, Article 39 (March 2010), 23 pages. https://doi.org/0000001.0000001

Appendix for Oversized Figures

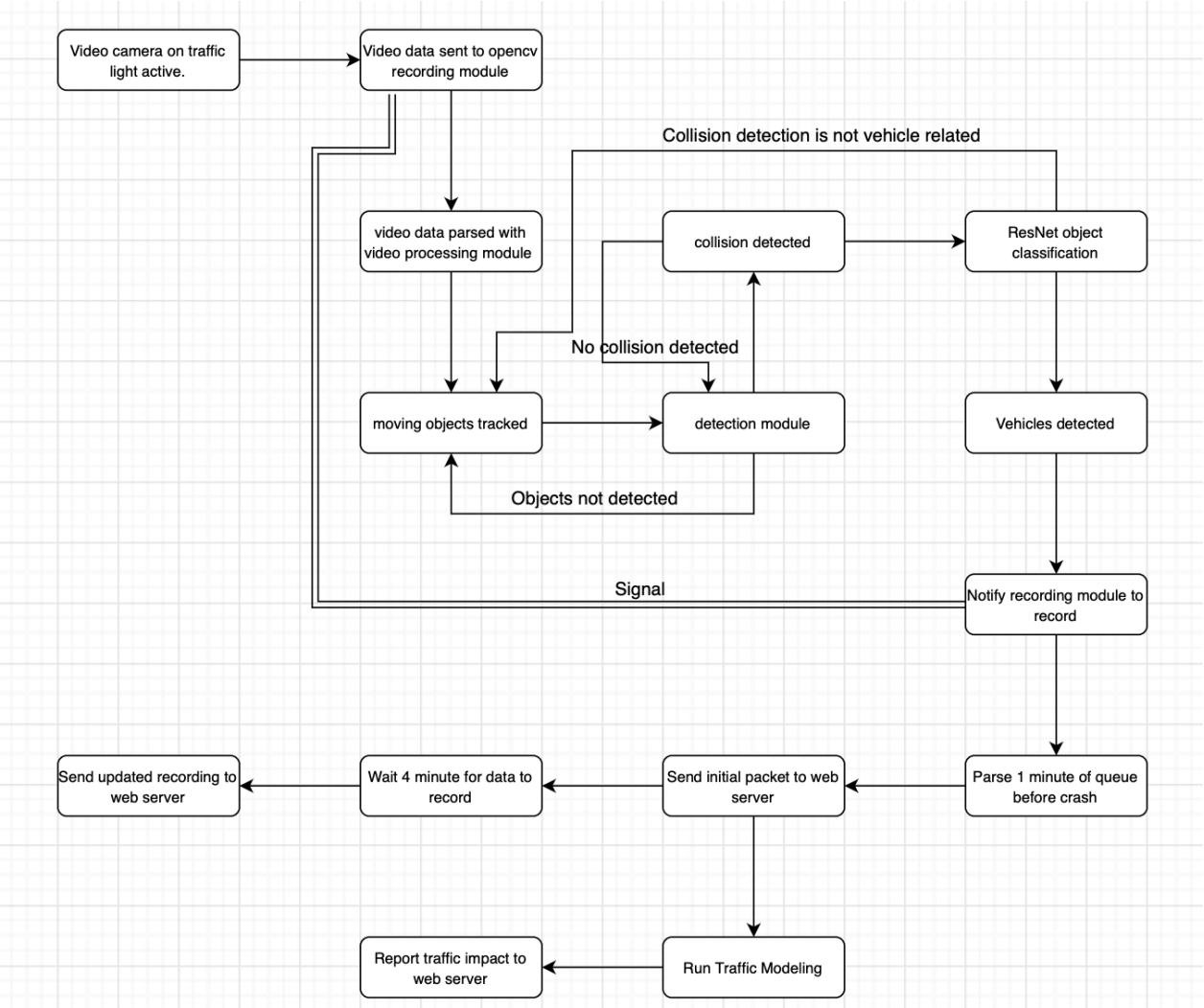
Description	Model #	Manufacturer	Quantity	Cost @	Total
Colab Pro	NA	Google	2 Months	$9.99	19.98
AWS	NA	Amazon	NA	$50	$50
					$69.98

Appendix B: Bill of Materials and Budget



TASK	ASSIGNED TO	PROGRESS	START	END
Phase 1 Title				
Research Crash Detection Algorithms	Arvind	0%	3/4/22	3/11/22
Researching Rerouting Algorithms	Goran	0%	3/4/22	3/11/22
Research Video Logging and Web Server	Jonathan	0%	3/4/22	3/11/22
Phase 2 Title				
Implement Crash Detection	Arvind	0%	3/11/22	4/1/22
Implement Rerouting Algorithms	Goran	0%	3/11/22	3/30/22
Web Server	Jonathan	0%	3/11/22	3/25/22
Video Logging	Jonathan		3/25/22	3/30/22
Emergency Response Notification	Goran		4/6/22	4/11/22
Phase 3 Title				
Signal Handling Detection/Routing/Transmission	Goran/Arvind		4/6/22	4/20/22
Crash Detection Testing	Arvind/Goran		4/11/22	4/18/22
Test Detection/Routing/Transmission	Goran/Arvind		4/19/22	4/26/22
Set Up Web Server/Live Video Buffer	Jonathan		4/6/22	4/27/22
Data Collection	Jonathan		4/6/22	4/20/22
Phase 4 Title				
Signal Handling with Web Server	Arvind/Goran		4/27/22	5/4/22
Buffered Camera Recording on Crash	Jonathan		4/21/22	4/28/22
Interim Demo	NA		4/30/22	5/2/22
Adjustments based on Feedback	Jonathan, Arvind, Goran		5/3/22	5/14/22
Final Presentation	NA		5/15/22	5/20/22

Appendix A: Gantt Chart



Appendix C: Block Diagram of System