Stoplight Detection: Implementation and Forgery Attacks

Olivia Degner

Stoplight Detection: Implementation and Forgery Attacks

An Undergraduate Honors College Thesis

in the

Department of Computer Science and Computer Engineering
College of Engineering
University of Arkansas
Fayetteville, AR

by

Olivia Degner

Thesis Advisor: Dr. Qinghua Li

Thesis Defense Committee Members:
Dr. Matthew Patitz
Dr. Khoa Luu

Abstract:

One of the up and coming topics in the world of technology is that of autonomous vehicles and self-driving cars. Autonomous vehicle technology has the potential to make a dynamic impact on society, drastically altering global transportation and the automotive industry. When human beings are no longer responsible for making the decisions required of controlling a vehicle, the importance of security and accuracy will become absolutely vital for these autonomous systems. If the system can be hacked and fed false information, there is the possibility of putting innocent lives at risk. In a world of growing global terrorism, this is a key and reasonable concern that must be addressed if autonomous cars are truly the future of global transportation.

The goal of this project is to create a system that will take the output of a self-driving car's camera and use it to detect if a stoplight is present in an image and collect information about the current state of the stoplight and how it changes over time. This collected data is paired with a supervised learning algorithm, such as the K Nearest Neighbor algorithm, which gives the system the ability to classify what type of color change is occurring in a stoplight. In addition to collecting raw data from several stoplights, the system was tested by creating and assessing several different types of forgery attack to see if they cause the identification algorithm to suffer a drop in accuracy. Ultimately, the purpose of this research is to create a more intelligent system in autonomous vehicles that will be able to detect forgery attacks on its camera system. This intelligent system would work to prevent situations where attacks on the system could result in collisions between vehicles and nearby pedestrians.

1. Introduction

The development of autonomous vehicles is one of the biggest areas of discussion in the technology world today. Commercialized self-driving vehicles have the potential to change transportation dramatically. In theory, intelligent transportation through autonomous vehicles could reduce the amount of accidents on the road due to human error and create an overall safer transportation infrastructure. One of the most vital components to the development of autonomous vehicles is the process by which the vehicle is able to detect the environment around it. The utilization of LIDAR sensors and cameras are two of the most common approaches to solve this problem. LIDAR sensors allow the vehicle to detect objects in its proximity, such as other vehicles or obstacles, and determine how close or far these objects are from the sensor. Cameras can provide the vehicle with a digital image of the area in front of the autonomous vehicle, providing either a monochrome or full-colored image (Mahdavi 117). Through the use of the sensors and cameras, the system can be provided with significant data about the world surrounding the vehicle such as proximity to pedestrians or other objects, whether the vehicle is close to a stoplight, what the color of that stoplight is, and the location and quantity of nearby vehicles.

While cameras are a powerful option for viewing the environment around itself, they also introduce a big security concern. If the camera is hacked by an attack, the false information that it provides to the system poses a large danger to the passengers of the autonomous vehicle, passengers in nearby vehicles, and nearby pedestrians. This forged information can lead to collisions between vehicles or pedestrians. For example, forging a green light while the light is actually red could cause the car to drive straight into the intersection and into oncoming traffic or pedestrians crossing the street and violate traffic rules.

The goal of this project is to create a system that will take the inputs from these cameras and use image processing to implement a program to detect stoplights in these images and determine what information the vehicle needs to know about the stoplight, specifically whether a change has occurred in the color of the stoplight. This information will then be run through a K Nearest Neighbor algorithm to create a program that will be able to correctly classify what type of change has occurred based on the collected data and test this process when forgery attacks are present in the KNN algorithm's testing data.

2. Related Work

Due to the widespread impact that stop lights have on a daily basis, research into stoplight detection has been an increasingly important area of research, especially as autonomous vehicles grow closer to market viability. Therefore, a wide variety of related work was referenced when beginning this research. In 2009, Raoul de Charette and Fawzi Nashashibi developed their own stoplight detection system using a modular algorithm for Traffic Lights Recognition (TLR) and an Adaptive Template Matcher (ATM). This system receives inputs from a camera mounted on a moving car. The images were converted into grayscale and a robust Stop Light Detection (SLD) was executed on the image to detect all stop lights that were visible on the image. After this, the ATM was executed on the image, which used geometric and algorithmic templates to provide each potential stop light with a level of matching confidence. De Charette and Fawzi tested their stoplight detection system using a video stream database consisting of over 20 minutes of video recorded from their prototype vehicle in Paris, France (de Charette and Nashashibi).

While their system was able to function in real time, their system did not make use of machine learning. With the advancement of technology since 2009, machine learning processes and deep learning have become the focus of modern technology development. In addition, the results from De Charette and Fawzi's detection software did little to provide information about the stoplights that were detected, such as the color of the stop light in the image. The goal of this new project is to not only to detect stoplights in an image, but to gather visual information from the lights themselves to be able to provide more sophisticated results, such as the type of change occurring on the stoplights over a set of images from the camera.

3. Data Collection

The first task that needed to be completed to develop this program was the collection of data. Videos of various stoplights were collected to be used to test the detection of stoplights in an image and to determine information about the stoplights from the image. A Sony Handycam video camera was used to record this data. This camera allowed for videos to be recorded at a steady 60 frames per second, providing more accurate recordings of the transitions in the stoplights. Recordings of stop lights at intersections were conducted over a period of 4 months and each video was recorded during daylight hours. This footage was then used to test the image detection software and gather data for the system so that it can determine what type of change occurred in the stoplight during the video. The camera setup used to collect these videos is shown in Figure 1.

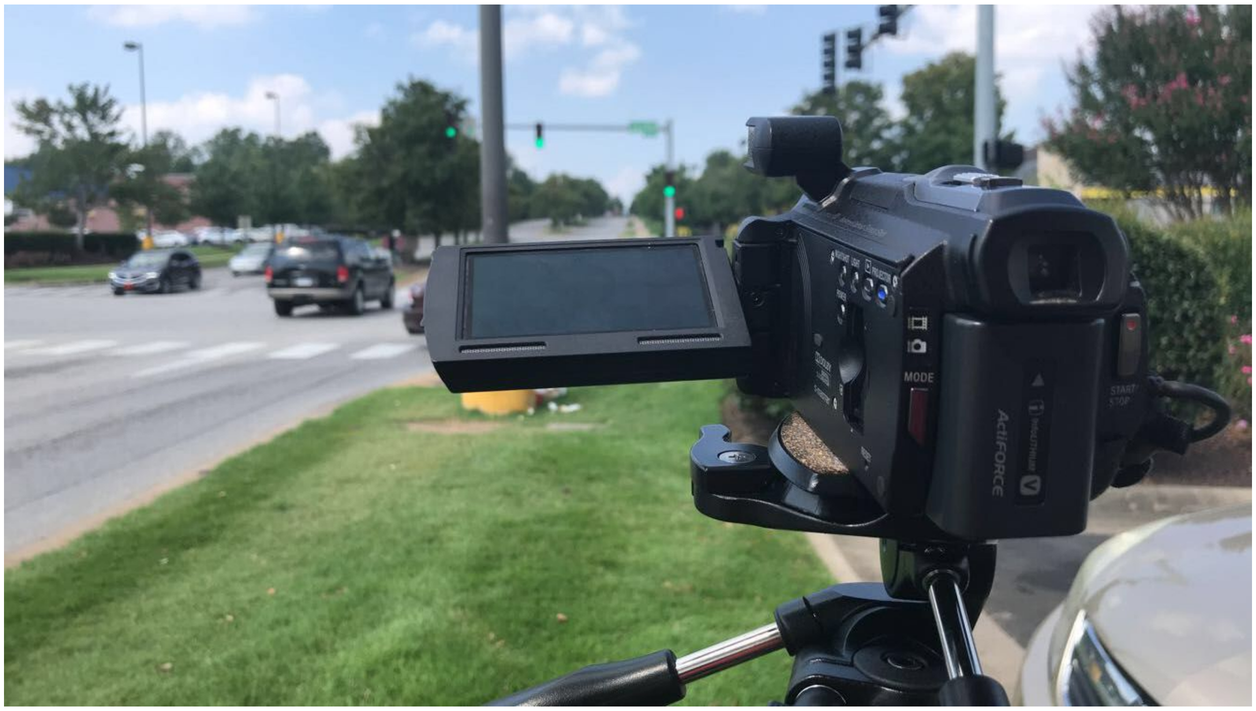

Figure 1. Camera setup used to record the stop lights.

There are many different manufacturers of stop lights, and their products may have different designs. In a single city, it is highly likely that more than one style of stoplight will be present. While these changes in stop light design may be small, they could cause problems in the image processing system if they are not accounted for. In order to combat this, videos were recorded at seven different stop lights of varying manufacturers around the city of Fayetteville, Arkansas. At each stoplight, three types of changes were observed and recorded with the Sony Handycam. These included a change from a green light to a yellow light, a change from a yellow light to a red light, and a change from a red light to a green light. Each change cycle was recorded 10 times. The chosen stop lights are indicated by the red dots on the map in Figure 2 below:

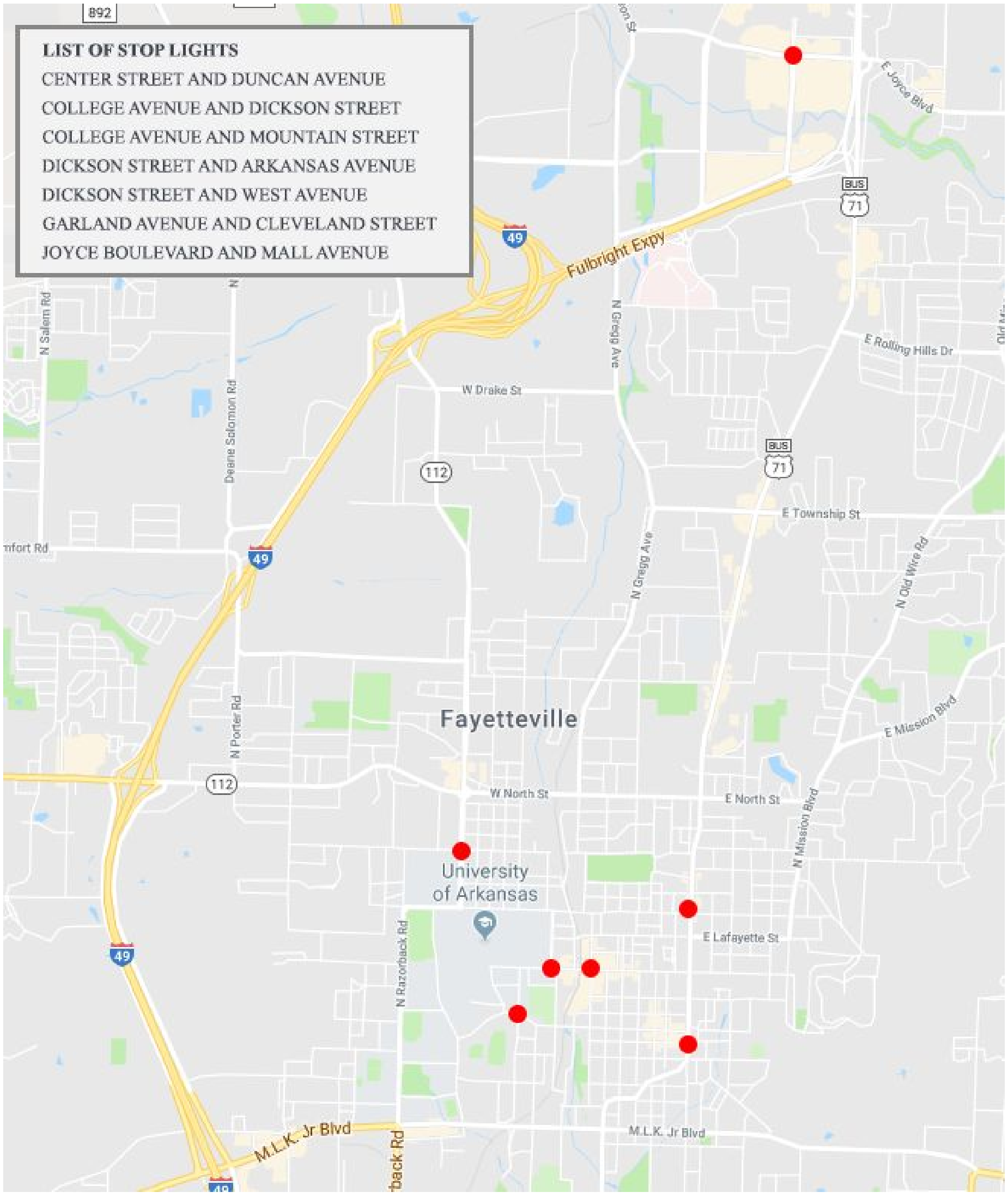

Figure 2. Map of stop lights used to collect data (Map of Fayetteville, AR).

4. Data Processing

4.1 Tensorflow and Image Detection:

Being able to detect objects in an image is a powerful characteristic of image processing. In the case of autonomous vehicles, it is absolutely vital to be able to locate and correctly interpret a stoplight in an image. When it comes to detecting certain objects in an image, many traditional approaches in machine learning are being replaced by more efficient and intelligent deep learning methods (Haridas and S.). One of the powerful deep learning tools that has become popular in recent years is Tensorflow's Object Detection API. Using Tensorflow, a model can easily be trained to detect a certain type of object, such as a person, a car, or even a stoplight.

With the rise of the open source culture in software development, it is becoming increasingly common for software developers to release their pre-trained models for other software developers to use in their own programs. This collaborative environment fosters productive development worldwide. In this collaborative spirit, Junsheng Fu's "traffic-light-detector" repository on GitHub was selected as the base for this program's design (Fu). In his code, Fu used Tensorflow's Object Detection API and a frozen detection graph to detect where a stoplight was in the images input into his program (Fu). The result of his program was an output command of "stop" if a light was yellow or red and "go" if the stoplight was green or if no stoplight was detected in the image. This was done by using OpenCV to apply a mask to the image to only consider if there was red in the stoplight and whether it exceeded a certain threshold. If this threshold was breached, then the program would output the "stop" command.

4.2 Data Preprocessing

In order to gather data about the color values of the light, each image of a stoplight that was detected by Fu’s trained Tensorflow model was converted from the traditional RGB color space into a HSV color space. A pixel in a HSV color space contains three main components. The first component, H, contains the hue or dominant wavelength of the image. This value most holistically represents the color value of the image. The second component, S, holds the saturation value of that pixel. The saturation values represent the “purity” of the color or what shade the color is. The third and final value, V, represent the value or intensity of the image. This tells the program how vibrant the color is (Gupta). One of the benefits of using an HSV color space over a RGB color space is that all of the color’s hue value is stored into one value. This makes filtering out colors in an image much easier, as only the H value is vital to performing this filtering process.

Using Fu’s code as a base, functionality was added to the program that created two additional masks for filtering colors in the images. One mask was made for detecting green pixels and one mask was made for detecting yellow pixels. These masks were combined into one total mask that would ignore the dark areas of the stoplight and only show and consider the sections of the stoplight where the lights were present. The benefit of using masks in this way was that any the parts of the stoplight that did not contain the lights could easily be removed from the pixel calculations that would occur in the next part of the program. Figure 3 shows an example of a green stoplight in the HSV color space before the color mask was applied and after the mask was applied to the image.

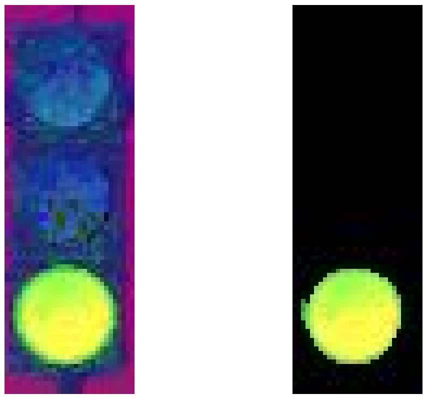
Figure 3. Stop light image with no mask applied (left) vs. with the color mask applied (right).

Once a mask has been applied, every pixel in the image that was not included in the mask will be set to black. In other words, its HSV values have been set to [0, 0, 0].

After the color mask was applied to the stoplight image, the image was converted back to a RGB color space and a function iterated over every pixel to calculate the average red value, the average green value, and the average blue value for the entire image, excluding the black pixels. This was done by taking each non-black pixel and adding it to the count of the total pixels and adding its red, green, and blue values to a running total for each, respectively. The calculated average RGB value for each image was then added to an array to be used to calculate the change in RGB values between the images.

The final step in this program was to calculate the changes in the average RGB values between adjacent images. In addition to calculating the changes between the images, the program also checked to see if it had found the first set of RGB differences in which either the red, green, or blue difference value was greater than 15.0 and saved its index in the difference array. This marked the beginning of a color change in the light. For example, the first time the value exceeded this 15.0 threshold was an indication for the first frame of a stoplight's change sequence from green to yellow.

Once all of the change values were calculated, the program called a function that takes the index of the first change and records 10 sets of RGB values for the frame changes, beginning with the 2nd change before the marked index and ending with the 7th change that occurred after the marked index. These 10 RGB change values were then written to a .csv file that would be used as an input to the K Nearest Neighbor Algorithm that is used to classify what type of change is occurring based on these RGB change values. When implemented, the user will have to go into the CSV file and add the classification of the stoplight change (what type of change occurred) if the data will be used in the training data set for the K Nearest Neighbor algorithm.

4.3 Data Analysis

Frames were taken from the videos at each stoplight and run through the program dictated above. After observing all of the frames and analyzing the data produced when run through the data preprocessing program, several patterns were observed. These patterns primarily pertained to the number of frames it took to observe a complete change from one color to another in a stoplight. The images for the patterns below were all taken from the same stoplight at College Avenue and Dickson Street in Fayetteville, AR.

The first observed pattern was in the change from a red light to a green light. This change occurred fairly quickly, usually taking 3 to 4 frames for the image to go from a fully lit red light to a fully lit green light. In most cases, the red light did not go from completely on to completely off in 2 frames, but would instead the red light would dim over a few frames before turning off completely, as seen in Figure 4 below.

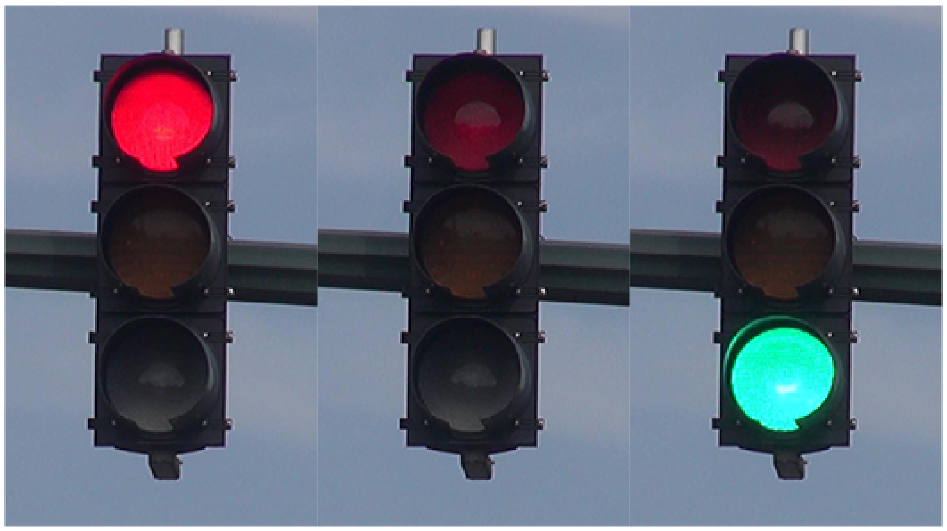

Figure 4. Frames from a red to green change on a stoplight at the intersection of College Ave. and Dickson St.

The second pattern was observed when a stoplight changed from a yellow light to a red light. This change occurred in a similar fashion to the red to green change. The change occurred over 3 to 5 frames on average and the yellow light would take several frames before it no longer produced any light. In some cases, the red light would not turn on completely in one frame. Instead, the red light would require 2 frames before it no longer showed a visible change in the image.

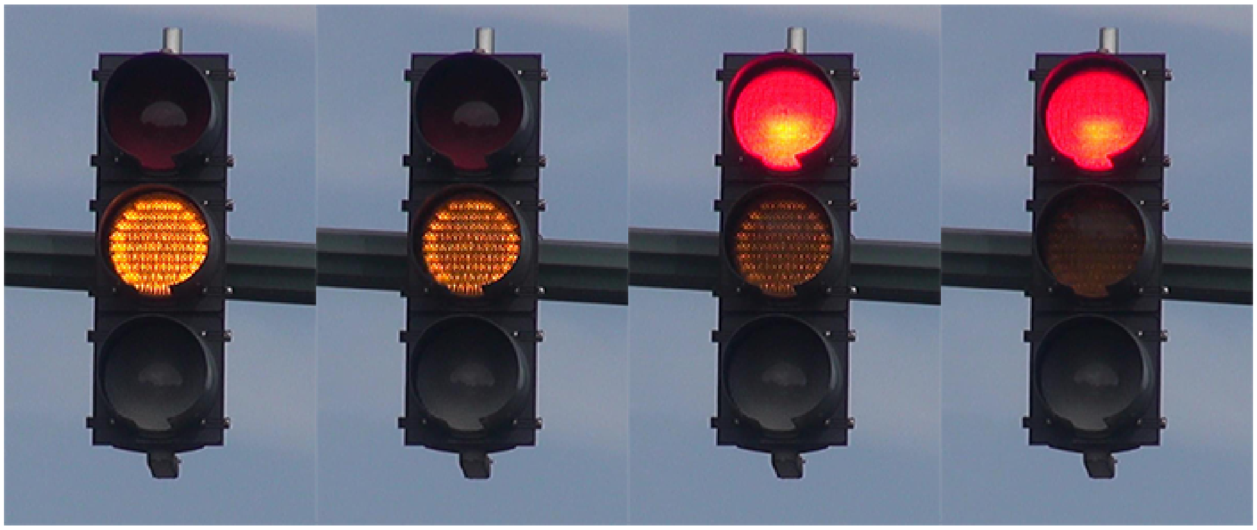

Figure 5. Frames from a yellow to red change on a stoplight at the intersection of College Ave. and Dickson St.

The third pattern that was observed was in the change from a green stoplight to a yellow stoplight. This was, by far, the most unique change between the three types. Being the longest observed change, the transition from green to yellow would usually take around 6-9 frames. In this change, there would be a few frames that were "dark", where neither the green light nor the yellow light were exuding any light output. There would also be at least one frame where the green light would be dimmed, but not completely off. This was also the case for when the yellow light turned on. The light would take an extra frame or two before it turned on completely, as seen in Figure 6.

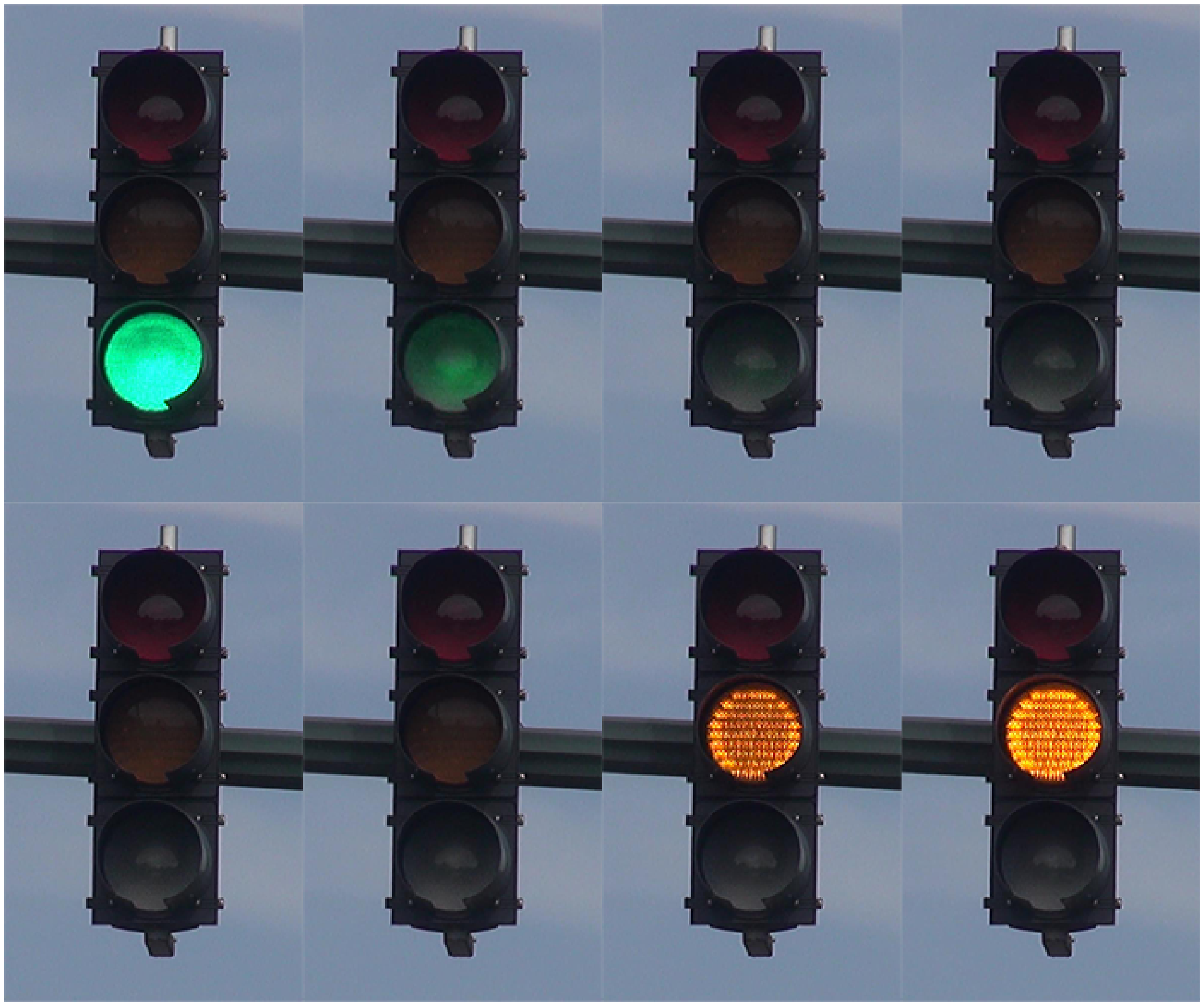

Figure 6. Frames from a green to yellow change on a stop light at the intersection of College Ave. and Dickson St.

5. Fabricating an Attack

After the data for each stoplight had been retrieved through the data preprocessing program, the next step was to create several type of forgery attacks that would also be run through the data preprocessing program and have their data preprocessed. When testing the K Nearest Neighbor algorithm program, these forged attacks are needed in order to prove that the algorithm can still accurately predict what type of change is present even when there are attacks present in the training data. Three different types of forged attacks were created.

The first type of attack that was created, Type 1, was done by taking the last fully lit frame before the color change started and altering its RGB value. The frames for this attack were created through the same program as the data preprocessing. By adding the "attack" parameter when running the main.py file, the program would still detect the stoplight in the image. However, this time it would perform image manipulation to change the pixels in the image to reflect the attack. For the green to yellow changes, the green value of the light's pixels were increased by 100. This change can be seen in the image shown in Figure 7.

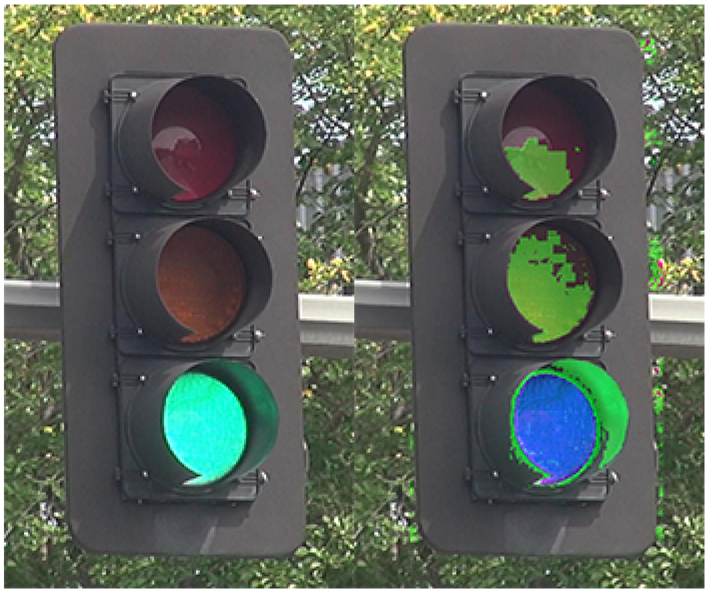

Figure 7. A Type 1 fabricated attack on a green to yellow change.

To create the Type 1 attack for the red to green change, the image with the last fully lit red light before the change began had their pixels' R color value increased by 100. An example of this addition is shown below in Figure 8.

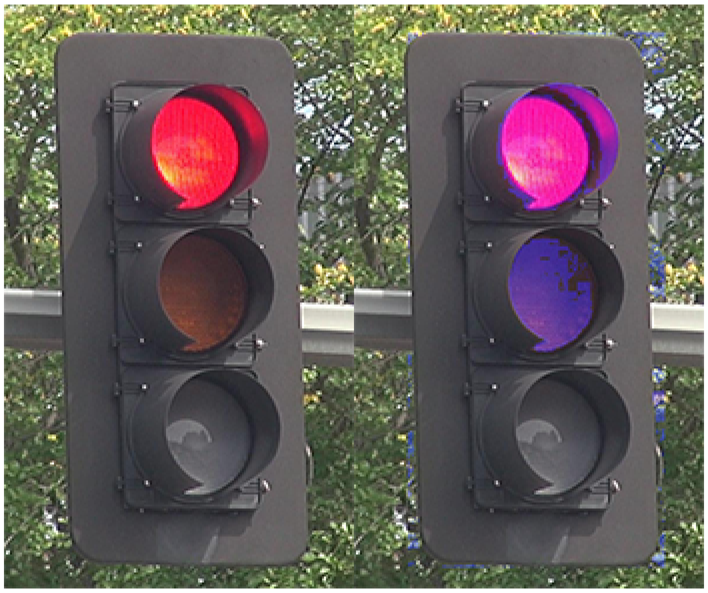

Figure 8. A Type 1 fabricated attack on a red to green change.

The Type 1 attack for a yellow to red change was created in the same manner as the red to green change. The program used the RGY_mask to detect the colored pixels in the stoplight and added 100 to the R color value in each pixel.

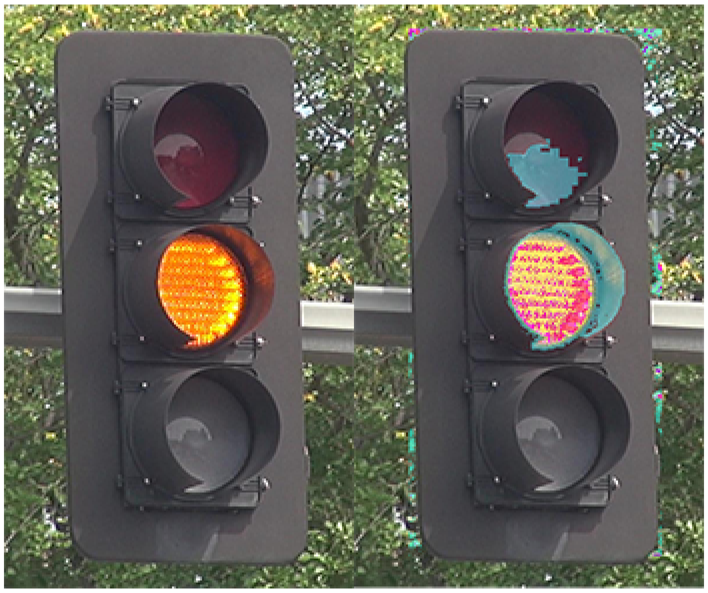

Figure 9. A Type 1 fabricated attack on a yellow to red change.

All of the images shown for the Type 1 attacks were created using videos from the intersection at Dickson Street and West Avenue in Fayetteville, AR.

The second type of attack, Type 2, was created by removing all of the frames that the stoplight changed over. The goal of this attack was to simulate a situation where instead of taking several frames for a stoplight to change from one color to another, the change would appear to be instantaneous. This means for each of the three type of changes, there would be no partially lit frames or any of the "dark" frames that were mentioned earlier. Examples of these Type 2 attacks are shown with the following figures. Figures 10, 12, and 14 shown the full transition of a stoplight change without any frames removed. Figures 11, 13, and 15 show how the change appears when the Type 2 attack is created by removing these transition frames. The images used in these figures were taken from the videos at the stoplight at the intersection Garland Avenue and Cleveland Street in Fayetteville, AR.

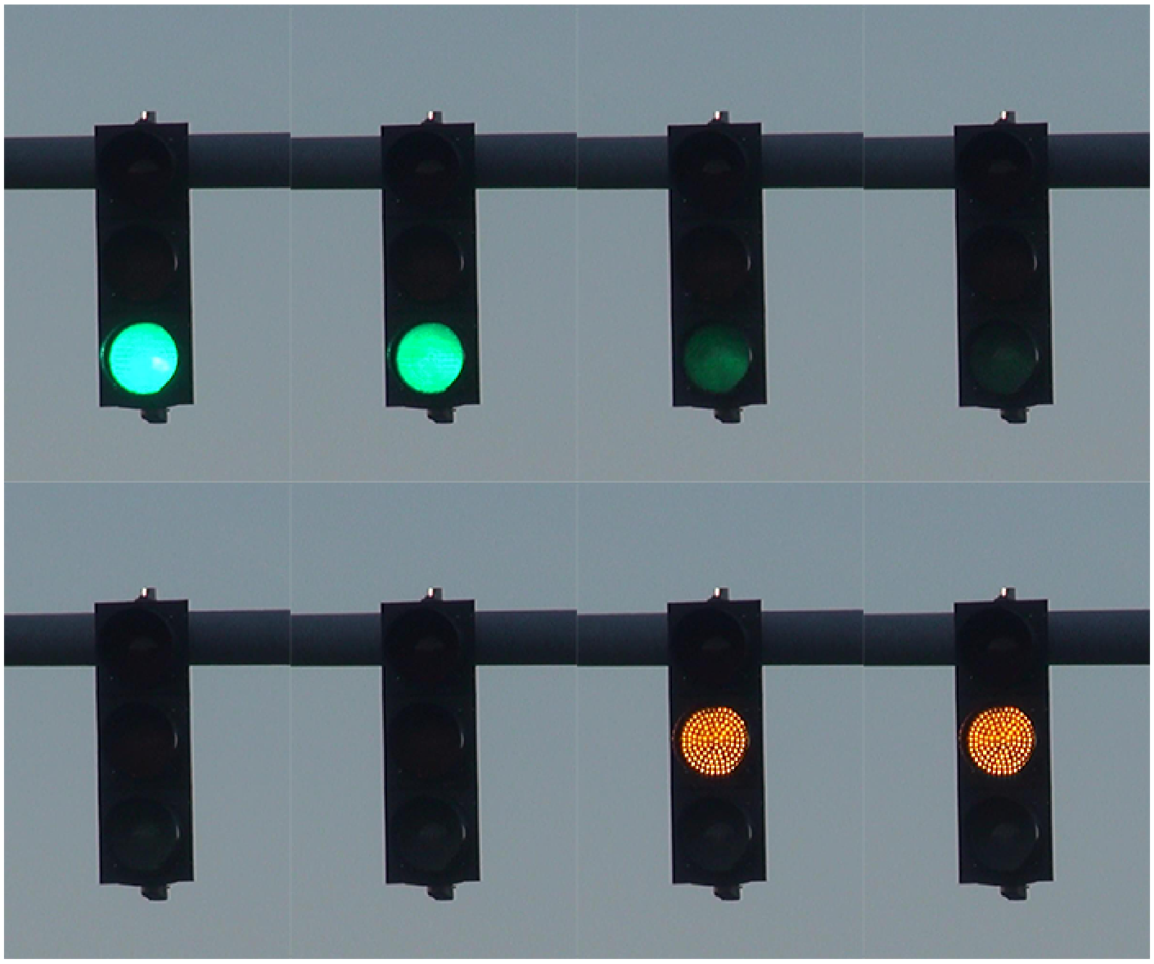

Figure 10. The original frames on a green to yellow change.

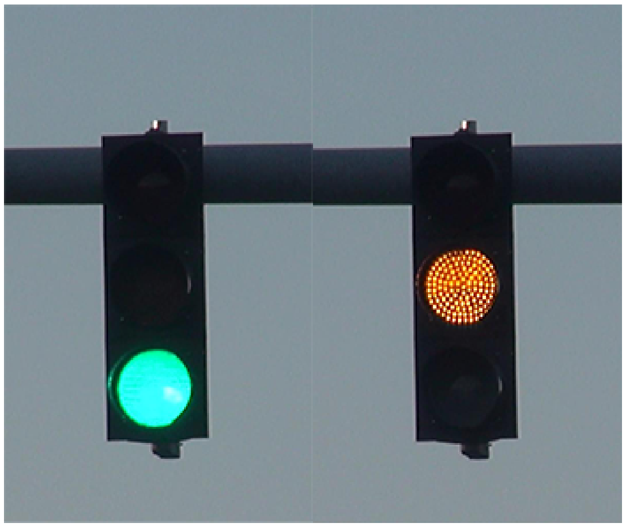

Figure 11. A Type 2 fabricated attack on a green to yellow change.

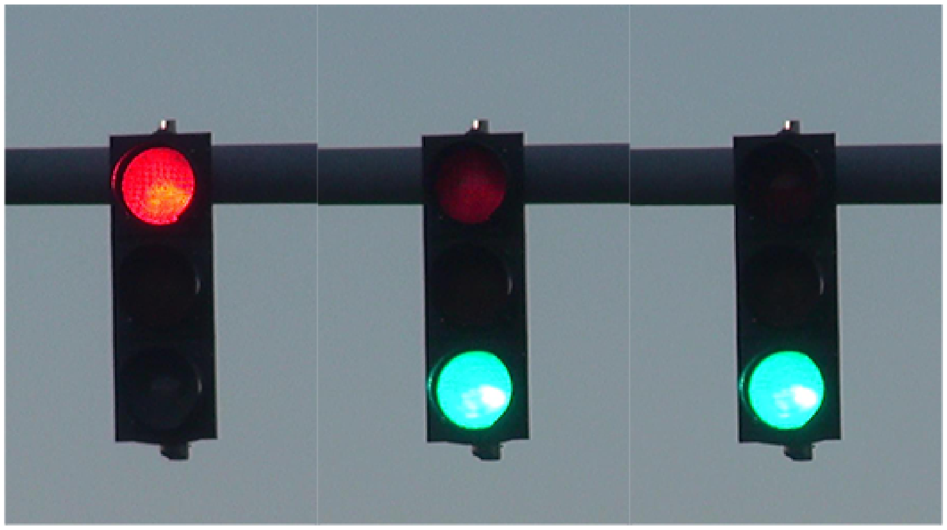

Figure 12. The original frames on a red to green attack.

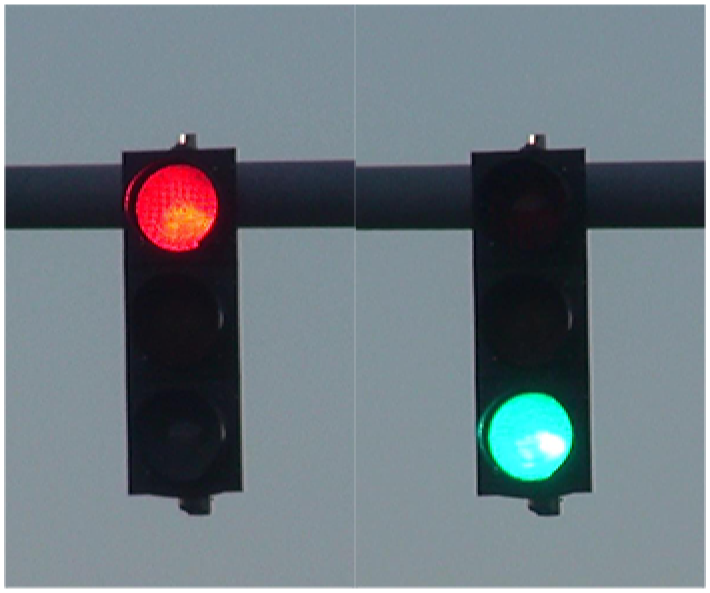

Figure 13. A Type 2 fabricated attack on a red to green change.

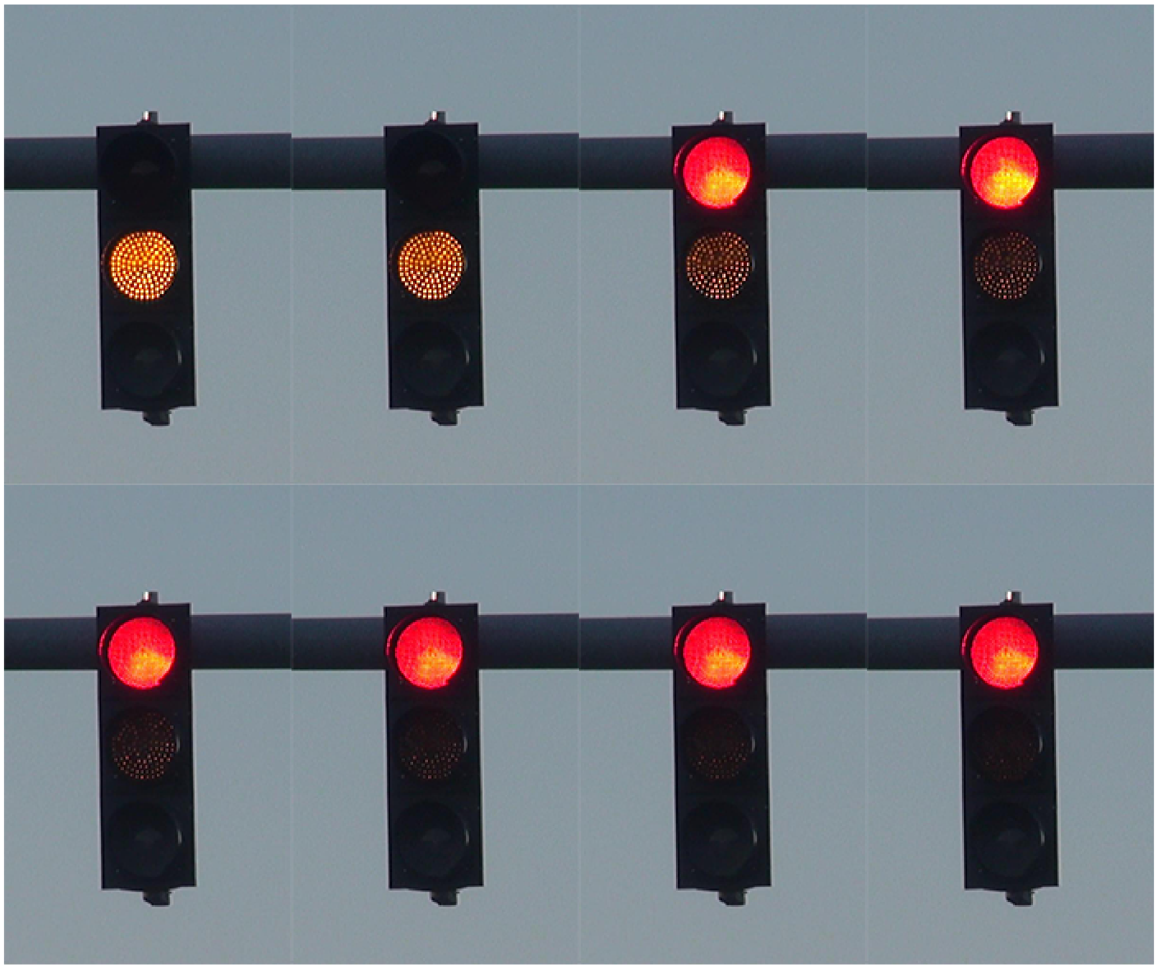

Figure 14. The original frames from a yellow to red change.

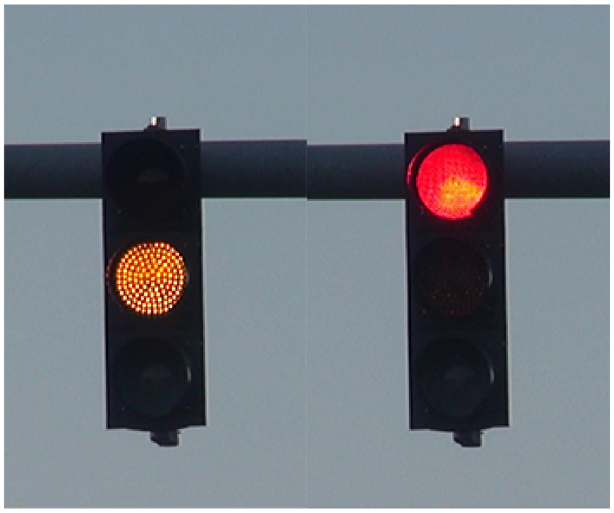

Figure 15. A Type 2 fabricated attack on a yellow to red change.

The third and final type of attack, Type 3, was created by taking the last fully lit frame before the color change started occurring in the stoplight and selecting a percentage of pixels to retain their color instead of dimming with the rest of the light. In the created attacks, the pixels were randomly selected from the first image by using the random class in python to generate a number between 0 and 1 and if the number was greater than 0.75, then the pixel would be selected. The result was that approximately 25% of the pixels would be chosen to remain unchanged. Figures 16, 18, and 20 show the original frames and figures 17, 19, and 21 shown the frames after the changes have been applied. The images shown in these figures were taken from the videos at the stoplight at the intersection of Dickson Street and Arkansas Avenue in Fayetteville, AR.

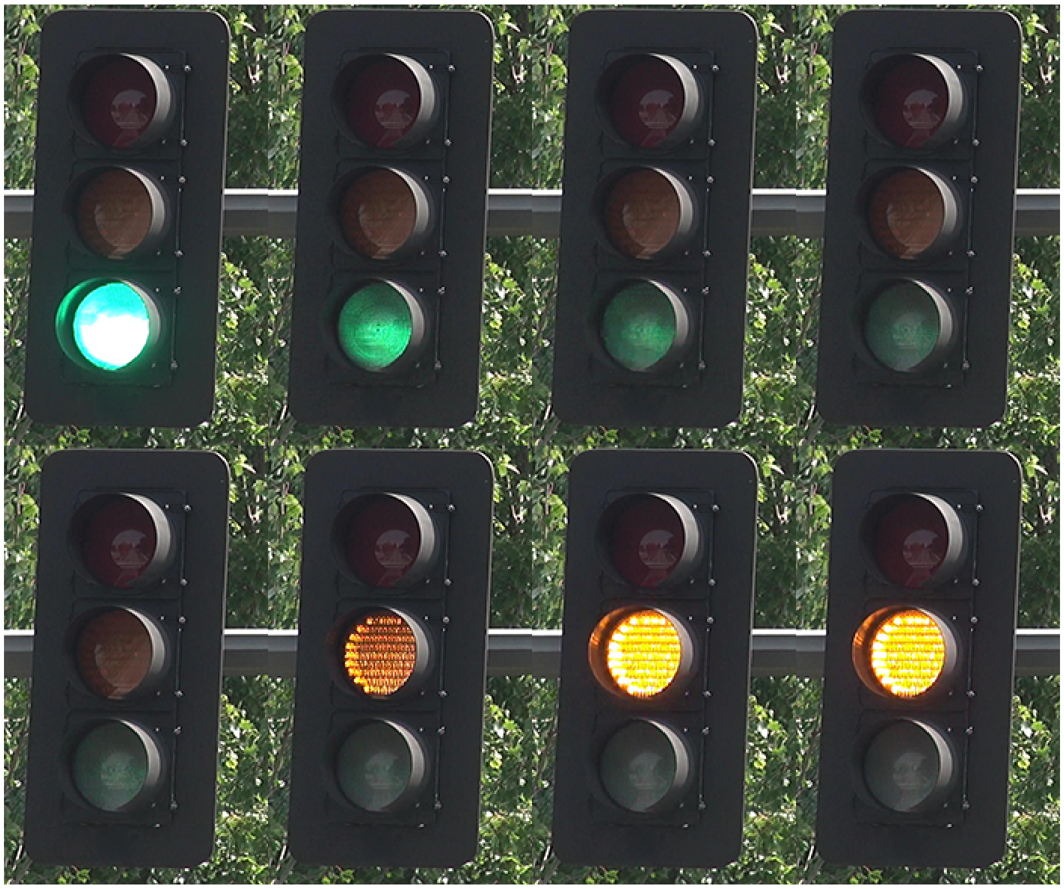

Figure 16. Original frames on a green to yellow change.

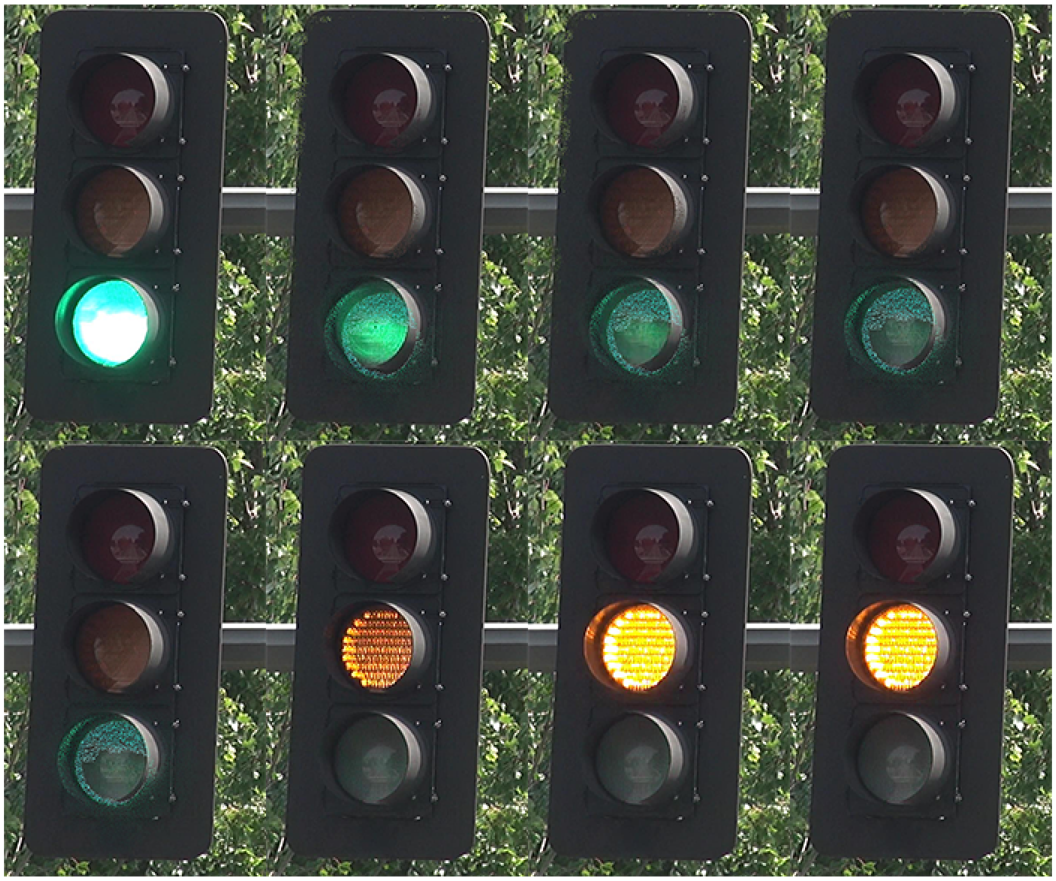

Figure 17. A Type 3 fabricated attack on the previous green to yellow frames.

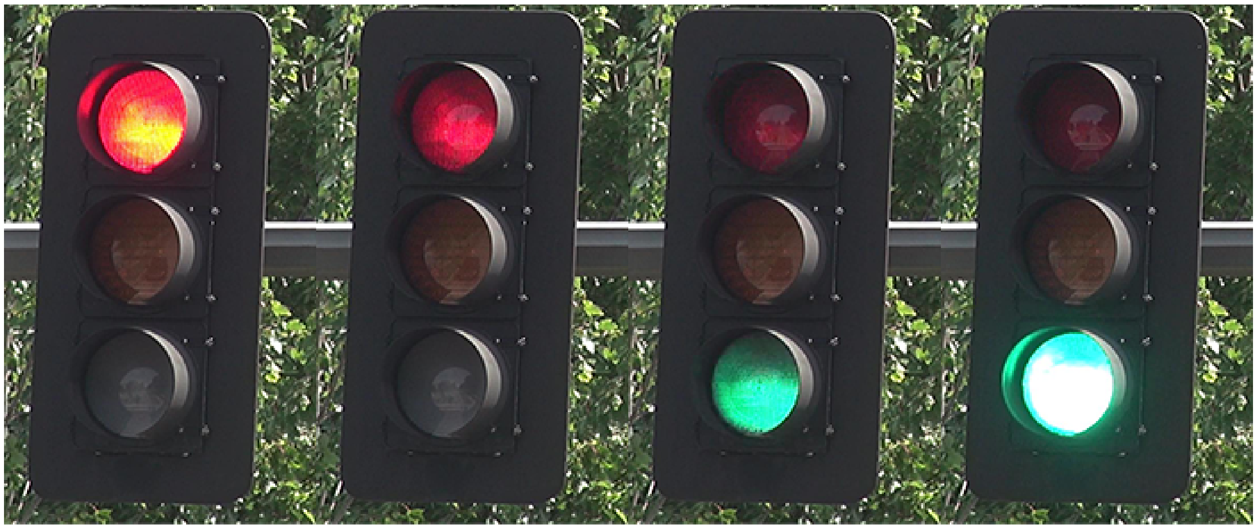

Figure 18. Original frames on a red to green change.

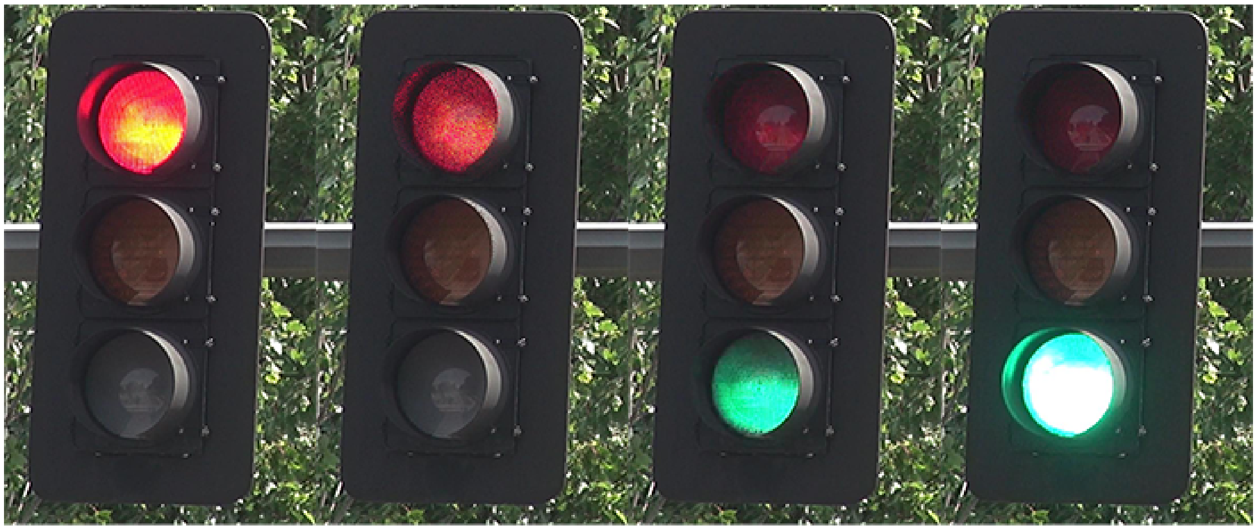

Figure 19. A Type 3 fabricated attack on the previous red to green frames.

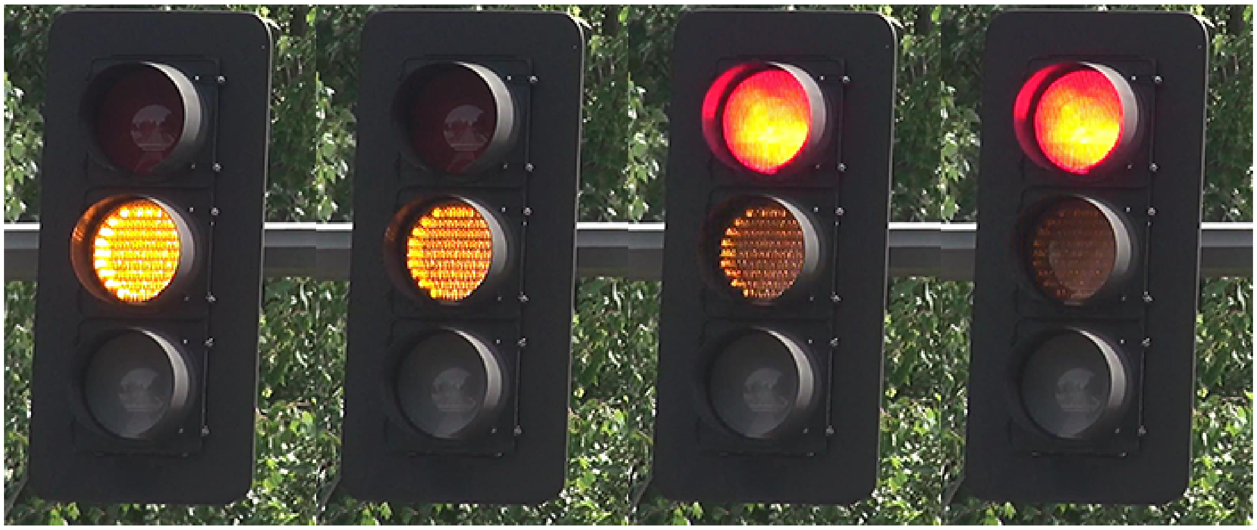

Figure 20. Original frames on a yellow to red change.

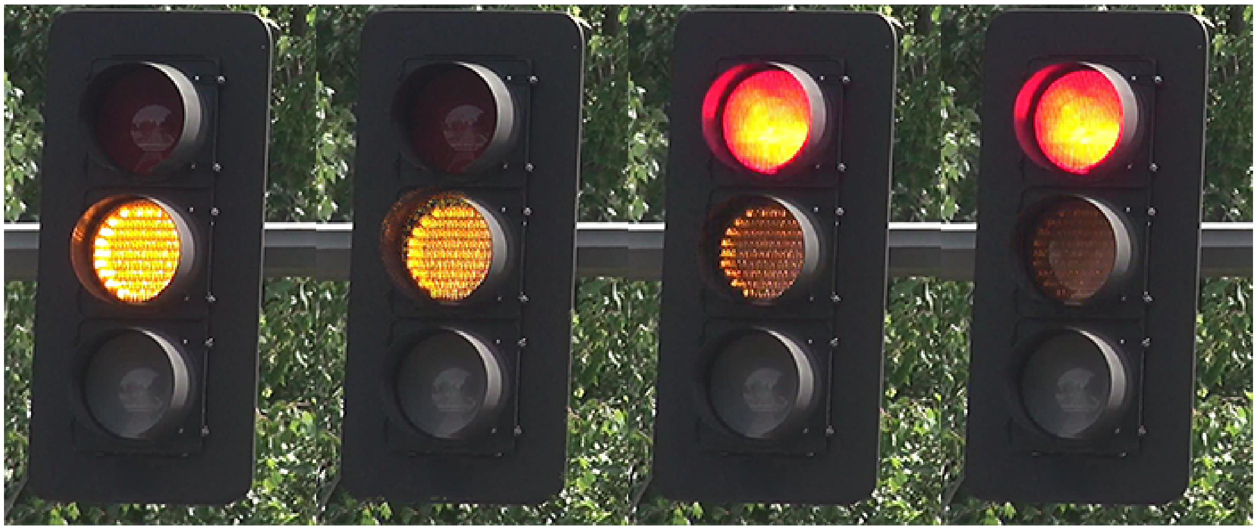

Figure 21. A Type 3 fabricated attack on the previous yellow to red frames.

For each of the 3 types of forgery attacks, 10 attacks were created for each type of change: green to yellow, yellow to red, and red to green. This resulted in 30 total attacks for each type.

The types of attacks used were chosen because of their simplicity. These types of attacks are simple enough that they can easily be defined as forged when looked at by a user. Given the time limitation, more advanced attacks will be studied in future work.

6. Stoplight Detection without and with Attacks

6.1 Detection Algorithm

Once all the data was collected on types of attacks, the next step was to create a program to determine what type of change was occurring in a set of data that was collected by the data preprocessing program. A K Nearest Neighbor (KNN) algorithm was determined to be the best way to accomplish this task. The KNN algorithm was designed around the assumption that similar types of data exist in close proximity to each other (Harrison). This is why the KNN algorithm is a beneficial method for classification problems. One of the benefits of using this algorithm is that it does not require a large amount of time that must be dedicated for training the system. All of the data will be stored locally and the training phase of the algorithm occurs each time the algorithm is called in the program. However, one of the drawbacks of using this method is that computation time can increase as the size of the data becomes substantially large (Harrison). However, in the case of this project, the size of the data did not appear to cause any significant increase in computation time.

The K Nearest Neighbor algorithm begins by loading in a data set for training the algorithm and another data set for testing. For each data entry in the testing data set, the Euclidean distances between the test data entry and all of the entries in the training data set will be calculated. The k variable in the KNN algorithm represents an integer value of the number of neighbors closest to the testing data entry that the program will find. Using the found nearest neighbors function, the program will look at the label and classification of each neighbor to determine what classification should be given to the test data entry. In the case of this program, the program will find the k nearest neighbors in the data set and will look at what type of changes had occurred. The most common classification among the neighbors will then be chosen for this test data entry and the program will be able to predict that the data is showing that type of change with a certain percentage of confidence. This confidence value is calculated by how many of the neighbors are of the same classification as the chosen label.

6.2 Results without Attacks

Using the data obtained from the videos of the seven stoplights and the three types of attacks that were forged, the accuracy of the K Nearest Neighbor algorithm as a classifier can be tested. The first test performed was to take all of the data (not including forged attacks) and check the accuracy of the program with varying numbers of k. For this test, there was no specified set of test data. Instead, this non-attack set of data was fed into the algorithm and was split using a random object in python and splitting the data on the value 0.85. Therefore, approximately 85% of the data would be assigned to the training data set and the rest of the data was assigned to the testing data set. For the purpose of testing the different numbers of k, the random variable was given a seed of 1000 so that number of data entries in the training and testing data sets would be the same regardless of the value of k. This test was performed with the following values of k: 2, 3, 4, 5, 6, and 7. The results are shown below in Figure 22.

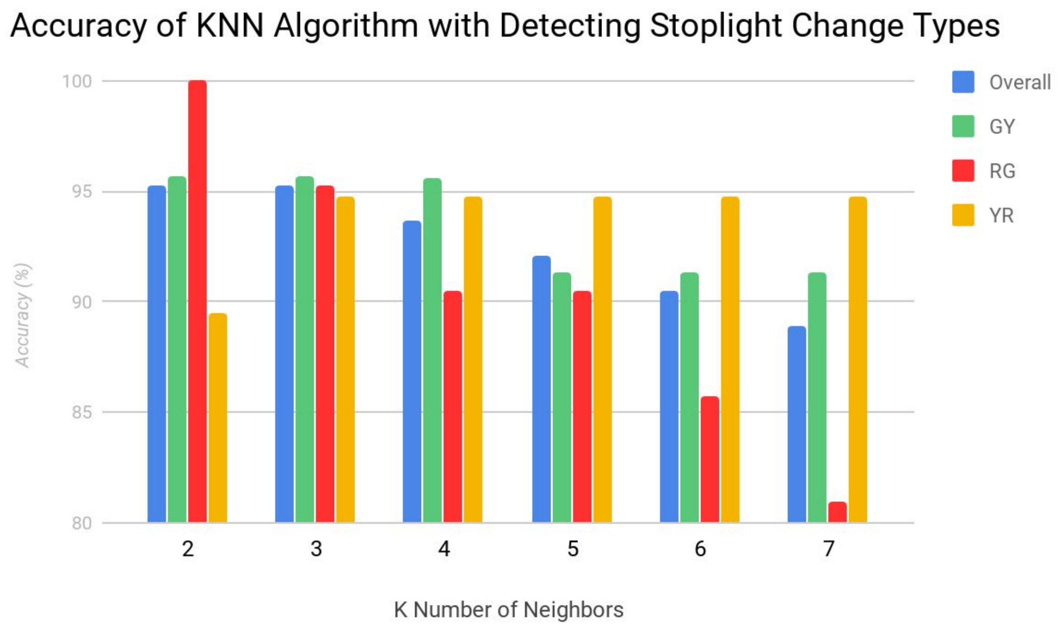

Figure 22. Accuracy of KNN Algorithm with Detecting Stoplight Change Types.

By looking at the graph, it can be determined that the best number of k to use for this K Nearest Neighbor algorithm is 3. The reason why the value of 3 was chosen is that out of all 6 values of k, 3 has the most consistency between the accuracy of the three types of changes and the overall accuracy. Each of the other k values has an outlier in the data, meaning that for one type of change, the KNN algorithm with be noticeably less efficient. With an overall accuracy of 95.23809524%, the accuracy of this KNN algorithm can be verified when it considers the number of neighbors, k, to be 3.

6.3 Results with Attacks

The next three tests that were performed on the K Nearest Neighbor algorithm were to add the forged attacks into the data set to determine how accurate the program would still be able to detect the type of change occurring in an entry of data.

The first of these three tests was performed by adding all of the collected data for the Type 1 attack into the csv with all of the non-attack data entries. Then, a group of seven videos were chosen to serve as the test data. For each of these seven videos, the data for a green to yellow, yellow to red, and red to green were included in the test data. This resulted in a test data set with 21 entries in it. The second of the three tests was performed by adding all of the collected data for the Type 2 attack into the csv with all of the non-attack data entries. The same 21 data entries were used as the test data. Finally, the last of the three tests was performed in the same fashion as the last two tests, except this time all of the collected data for the Type 3 attack was added into the csv with all of the non-attack data entries. The results from running the K Nearest Neighbor algorithm under these three tests are shown in Figure 23.

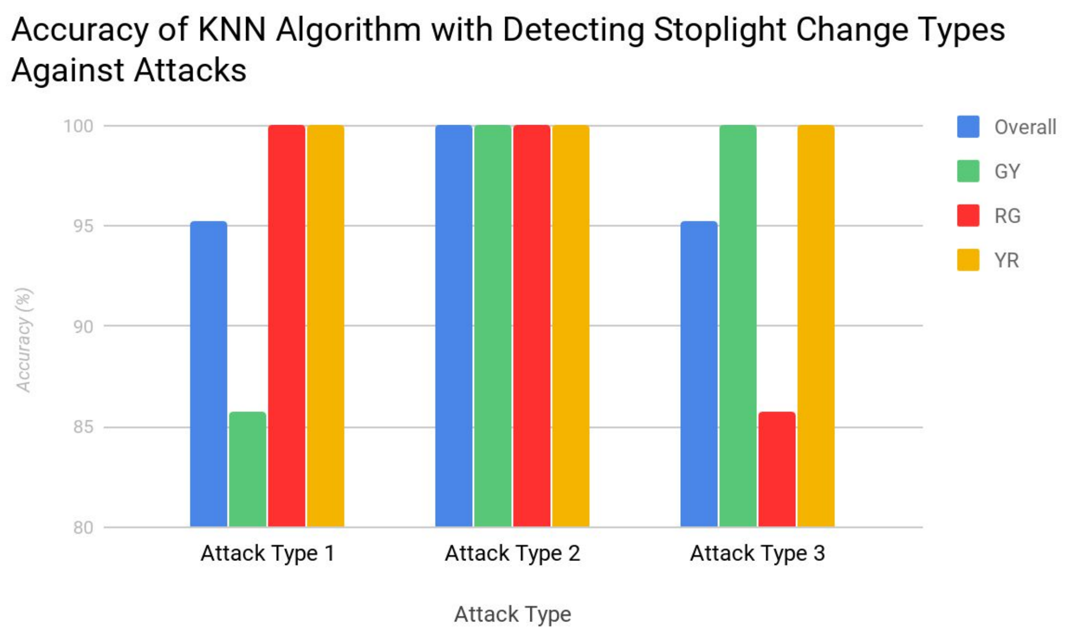

Figure 23. Accuracy of KNN Algorithm with Detecting Stoplight Change Types Against Attacks.

As the results in the above graph show, the KNN algorithm suffered a loss in accuracy when it was tested with the Type 1 and Type 3 attacks. The similarity in this data can be attributed to the fact that both of these forged attacks were created by manipulating pixels in the original image. The Type 2 attacks did not cause a drop in accuracy for the algorithm. This is due to the dynamic aspect in how frames were all together deleted to create the attack. Detecting a lack of changing frames is much easier when calculating the Euclidean distance than detecting data with only altered frames, not removed frames.

The algorithm was also tested to see if the program could accurately categorize an incoming attack of the three types. The program, however, struggled to accurately categorize an attack as such. This can be explained by the small amount of data that was present for each type of attack and that the attacks were not all created on the same stoplight location.

7. Conclusion

The goal of this project was to create a system that would receive images from a camera and use Tensorflow's Object Detection API and image processing to detect stoplights in an image and record the difference in the RGB color values between frames of the change in the color of a stoplight. This collected data was then used, alongside the set of data for three different forged attacks, to create a K Nearest Neighbor algorithm that would be able to accurately classify what type of color change was occurring in a set of data. The found results prove that the KNN algorithm is accurate in correctly classifying the type of change a stoplight undergoes and only suffers a slight decrease in accuracy when the three attacks are introduced into the KNN algorithm's training data.

However, the system can still be improved upon. This could be done by collecting more data and including a large data set for each of the forged attack types. The accuracy of the system also needs to be further validated before it can be implemented for use in autonomous vehicles.

The next step towards improving this project is to increase the efficiency of the data preprocessing system. Currently, the program takes around 15-20 minutes to process a set of around 35 frames. This is acceptable for testing the program in the context of this project. However, to implement this system in an active autonomous vehicle would require the processing time to be reduced significantly. By working to improve upon these inefficiencies in the future, this program has the potential for use in the real-time environment of an active autonomous vehicle and drastically improve their security and functionality.

In addition to increasing the efficiency of the data, the results obtained from this project can be further validated by increasing the size of the test data for both the standard stoplight data and the three types of forgery attacks. This would help to both improve the quality of the identification algorithm’s results and would allow for future tests in detecting the occurrence of a forged attack. The data could be further expanded by creating a stronger variety of types of forgery attacks. The overall intelligence of the algorithm will increase if it is able to correctly detect a more efficient attack against the autonomous vehicle camera system.

Lastly, the accuracy of the system can be further improved by exploring different methods of learning, such as neural networks, that can be used to classify what type of change a stoplight is undergoing and detecting if an attack being forged on the system.

Works Cited

de Charette, Raoul, and Fawzi Nashashibi. "Traffic Light Recognition Using Image Processing Compared to Learning Processes." *ResearchGate*, Nov. 2009, https://www.researchgate.net/publication/224090421_Traffic_Light_Recognition_using_Image_Processing_Compared_to_Learning_Processes.

Fu, Junsheng. "Traffic Light Detector." *GitHub*, GitHub, 18 Dec. 2017, https://github.com/JunshengFu/traffic-light-detector.

Gupta, Vikas. "Color Spaces in OpenCV (C / Python)." Learn OpenCV, Big Vision LLC, 7 May 2017, https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/.

Haridas, Nikhila, and Sandhiya S. "Traffic Light Detection Using the TensorFlow* Object Detection API." *Intel® Software*, Intel, 27 Apr. 2018, https://software.intel.com/en-us/articles/traffic-light-detection-using-the-tensorflow-object-detection-api.

Harrison, Onel. "Machine Learning Basics with the K-Nearest Neighbors Algorithm." *Towards Data Science*, 10 Sept. 2018, https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

Mahdavi Tabatabaei, Naser, et al. "Autonomous Vehicles : Intelligent Transport Systems and Smart Technologies." Nova Science Publishers, Inc, 2014. EBSCOhost, http://0-search.ebscohost.com.library.uark.edu/login.aspx?direct=true&db=nlebk&AN=809607&site=ehost-live&scope=site.

Map of Fayetteville, AR. *Google Maps*, 30 Aug. 2018, www.google.ca/maps.