ASSESSING THE IMPACTS OF PHYSICAL SENSOR DEFECTS IN SELF-DRIVING CARS

How to measure the impact on the accuracy of a distorted image feed on the results of the image detection

Team 2: Dustin Haines, Aiden Kathriner, Nisura Perera, Xuan-Dung Andrew Bui, Ian Felix - Monash University

ABSTRACT

This paper aims to increase the affirmation of the impact of image distortion on the success rate of object identification by presenting the findings of objection detection when sensor failure occurs. We discuss the systems we have tested seeing that they do not cater effectively when the image distortion occurs which can lead to catastrophic results. Through testing, cases were observed where objects were not being detected at all, objects being classified incorrectly, and new images being detected due to sensor defects when compared to a baseline run of the same input data without a defect. The introduced findings have been produced by using our own image manipulation package for ROS and running YOLOv3 on the manipulated feed. Our modules provide an easy way to apply visual effects to camera feeds used in ROS systems which can be used for further assessment into follow on effects of defective visual data.

SUPPLEMENTARY MATERIAL

Open-source package source code for ROS module developed by us: https://github.com/HainesDustin/FIT4003_T2

1. INTRODUCTION

Automobile manufacturers in recent years are aiming to move the transport industry towards self-driving cars. This industry is growing and becoming more popular at a rapid rate, seeing these vehicles making it into the hands of more people every day. Driving the development of autonomous cars is the goal to make transport safer by eliminating the human element to create a more efficient and predictable vehicle. In order to fulfil the Society of Automation Engineers (SAE) level 5 autonomy [1], no driver attention must be required. Development in this field has grown by leaps and bounds with the release of functions such as Tesla's Autopilot [2] and Alphabet's Waymo that are already being used on roads around the world. Many other companies are actively working on their own implementations of autonomous vehicles in collaboration with companies specialising in artificial intelligence to eventually deliver these products to consumers in the early 2020s.

Like any other product released to a consumer, these vehicles need to be tested and assessed for how well they accomplish what they are designed to do. For cars, safety is one of the core concerns and is seen in the extensive testing done to assess the safety of a vehicle. Unlike traditional vehicles that are assessed primarily on their mechanical capabilities, autonomous vehicles introduce a layer of complexity with software components that are used in place of human operators; and unlike other software systems, autonomous vehicles are used in a dynamic world of inputs that have a near-infinite range of possible scenarios as opposed to more simple software tools that are designed to accomplish a limited range of functionality in a fixed domain.

The implementers of autonomous vehicle systems are exploring many unique ways to address the testing of such algorithms. Modern, but complex software solutions such as neural networks [3] and artificial intelligence models are being used to replicate the thinking used by human operators when operating a vehicle. In many areas outside of autonomous vehicles, intelligent systems are surpassing humans in well-defined activities such as games to the point where they exceed human ability. A key example of this is Google's DeepMind AI used in AlphaGo, an AI trained to play the game Go. AlphaGo's greatest success was seen in 2016 where it managed to defeat the Go World Champion Lee Sedol 4-1 [4]. Such intelligent solutions are the natural approach to autonomous vehicles, with research suggesting that artificial intelligence will outperform humans in driving activities in the next 10 years [5].

As promising as these solutions are, determining the effectiveness of them is proving to be a complex issue when implemented in self-driving cars due to the challenges that are presented in busy, urban environments [6]. Environmental factors such as pedestrians, signage and weather complicate the processing needed to direct a vehicle. Success has been seen when only considering one such obstacle, but the cases of success are determined by those implementing the system and trialing it in controlled situations. Putting the system into other situations has found gaps not previously identified in the limited testing done and has led to the first fatality caused by an autonomous vehicle [7].

Refining test approaches is becoming important in ensuring the success of self driving cars. New techniques are capitalising on metamorphic testing, which generates new tests based on properties of already defined cases and relationships to modify those properties to change input but maintain output [8]. Benefits of this have been seen when applied to testing weather conditions, but are yet to see widespread usage with testing across all aspects of self driving cars. Here is where sensor failure becomes an issue. As these failures may inhibit the functionality of the autonomous car. In particular, the object detection system could be inhibited drastically and not detect at all.

This paper explores findings of sensor failure on the main front-facing camera on an autonomous system. With how much these failures affect the autonomous system, in particular, the object detection algorithms.

2. BACKGROUND

2.1 Metamorphic Testing

Research into autonomous car testing has been abundant, with many papers published. Being such a complex system with high risks involved, various methods have been developed to ensure that all aspects operate correctly. In traditional software systems, they can be tested against a test oracle to ensure that the program is working as expected. The issue with test oracles as a means of verification is that they are not applicable in some scenarios. Where computations are intense or there are a large number of scenarios to validate, a test oracle is not feasible as it is too difficult to apply [8].

A key technique used to overcome this issue of test oracles is metamorphic testing. For example, once a neural network is trained to navigate through obstacles in perfect conditions, validation must be done so that the same behaviour holds under suboptimal conditions. Typically, this would require a completely new dataset which would represent different conditions as well as an oracle that would determine the correctness of the output.

Instead, metamorphic testing is used to overcome the issue of a test oracle, which helps distinguish correct behaviour from incorrect behaviour. It has proven to be an effective solution to save time and resources. This is done through the process of generating new test cases on previously successful tests. The inputs and outputs of the original and the newly generated test cases are linked and are called metamorphic relations.

For example, many papers have been written on the subject of metamorphic relationships to test steering output during obscuring weather environments[9][10]. To save time having to collect a new dataset, transformations were placed on previously used testing data to imitate such conditions. Since the correct output of the original data is known, it can be inferred that the output of the transferred data should be the same. This metamorphic relationship is used to bypass any issues regarding a test oracle.

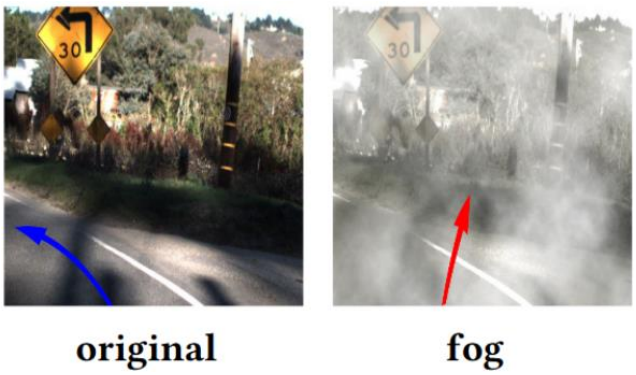


Figure 2.1 Path finding original vs fog[10]

Figure 2.1 is a case used by a research paper to try and understand the different effects weather has on the path planning of autonomous vehicles. The original image on the left is successful and the correct path is shown by the blue line. By transforming that image to mimic the effects of fog, a new test case is created. Since we know the correct output of the original image the metamorphic relationship should still hold with the new case. However, we see with the red line that the same output is not given. Thus the case can be considered as a failure.

2.2 Defective Data

Testing the robustness of a self-driving car has often come down to the effectiveness of the system sensors under different driving conditions. Weather condition filters such as rain and fog have been applied onto initial test data to ensure the correctness of the expected output and the actions that the car would take given the condition that it was in [9]. Furthermore, when testing, researchers have majorly looked into testing a particular functionality of a self-driving car. Functionality like lane detection is often tested as an essential safety feature since failing to follow lanes would eventually lead to accidents with other vehicles [11].

The common assumption in research so far is that the sensors onboard the vehicle are optimally performing. With the expansion of self-driving cars, this may not always be the case. Much like cars now, drivers maintain their vehicles at varying levels, so to expect self-driving cars to be in ideal conditions is unrealistic. This opens up an area of testing autonomous cars that have not been thoroughly investigated. During the journey of a car, situations where cameras and sensors are damaged may occur. Since these vehicles operate in high risk environments, hazardous inputs need to be adequately identified. Resolving them on the other hand proves to be difficult due to the large number of defects which may occur. Total sensor failure is easy to determine by lack of data received by the computer system managing the vehicle, but cases of partial failure are harder to work with. These situations are complex in that defective data is unique to the situation and environment that the car is driving in. Although the sensors may pass in data, there is no clear research into how automated cars should interpret defective data. From a hardware perspective, this problem can be overcome with autonomous vehicles utilising an extensive array of sensors for redundancy. Having multiple sensors and cameras reduces the probability of a car receiving an incorrect input unknowingly, by identifying differences between input data. Detecting a mismatch in data when two sensors are known to return identical data in optimal conditions allows for recognition of potential issues, however the research into a software solution has not come about and is a topic that needs to be more thoroughly discussed. Further to this, the approach is less prevalent than other options due to the additional costs, complexity and computing power required to support a greater number of sensors.

Research into autonomous vehicle testing operates under the assumption that all sensors are performing optimally. This may not always be the case especially when day to day driving conditions are factored in. Total failure of a sensor can easily be identified by the lack of outgoing data, however more subtle effects like cracks on a lens lie in a gray area between this and normal operation. This is the gap in knowledge which we have discovered in current research.

3. RESEARCH QUESTIONS

1. How can metamorphic testing strategies be used to measure the impact of physical sensor defects in self-driving cars?
2. How accurate are object detection models in self-driving cars when presented with defective data?

4. METHODS

4.1 Data

A Monash University student team Monash Connected Autonomous Vehicle (MCAV) is actively developing an autonomous vehicle by augmenting a Subaru Forester with off-the-shelf components connected to a range of hardware running Autoware.ai by the Autoware Foundation [12]. Autoware.ai is powered by the robotics middleware solution Robot Operating System (ROS)[13]. A benefit of using ROS as the basis for the system is the ability to record both input and output data in realtime to files known as ROSBAGs, which can later be replayed for further analysis or processing. The MCAV team have graciously provided access to a collection of ROSBAGs captured during their testing with a range of data including camera feeds, LIDAR point cloud maps, GPS readings, etc. Given the size of these files (which varies based on data captured and duration), a select few samples have been chosen which include camera data capturing different driving areas and situations. Samples used range from 2 to 10 minutes, and involve the car being driven around the streets close to the University's Clayton campus. This data has been accepted as-is, and has not been subject to any further processing, such as stutter reduction or image improvement, prior to the investigations detailed. The used data is the same as if it were just received from the camera in realtime.

A range of images (Appendix A) were sourced to use as the basis for defects. These were obtained through Google Image searches with queries relating to the defect required such as cracks and scratches. The requirement for these images was that they

needed a be a png file allowing for a transparent background such that they can be overlayed in processing detailed below. In addition, the image resolution needed to be greater than the image feed to ensure pixelation doesn't occur when resizing the filter image to the size of the feed.

4.2 Data Processing

In order to create testing scenarios close to what would be seen in a real-world scenario, we created a method to apply image transformations over a video feed to simulate a visual defect caused by hardware damage which could fit in the middle of a fully connected Autoware.ai system running ROS Kinetic. Modules used in ROS are easy to create using C++ or Python programming languages, which then connect to other running ROS modules through topics which are named buses carrying a predefined data type. For simplicity, the created image transformation module and others used have been written in Python to allow for debugging and modification without recompiling.

The image transformation module subscribes to a topic holding image data, such as a camera feed, then uses OpenCV [3] and a supplied image file to overlay the image onto the received image before broadcasting the result onto another topic for further use. Applying a supplied, transparent image (filter image) to the received image is accomplished through the following process:

1. Resize the filter image to the same dimensions as the received image
2. Extract the alpha (transparency) channel from the filter image, removing it from the image
3. Use the alpha amounts to create a mask of which parts of the filter are opaque (alpha != 0)
4. Black out the elements of the received image at locations specified by the mask
5. Add the colour values from the filter image at the mask locations to the received image

This approach means there is no colour defects caused by adding colour values of two different pixels, since the pixel values of the filter image effectively replace the pixels in the same location of the received image.

Autoware.ai includes a few open source tools to accomplish detection of objects from sensors. Based on MCAV's approach to autonomous vehicles, the object detection module for camera feeds to be used is You Only Look Once v3 (YOLOv3) by Joseph Redmon of the University of Washington [15][16]. YOLOv3 is distributed as a pre-trained convolutional neural network[17] capable of recognising 80 distinct object classes, including common objects seen in driving situations like cars, trucks, people, and traffic lights. An adaptation of the original implementation is used in Autoware for ROS functionality. This specific implementation (bundled with autoware) has not been used, but instead a similar package created by Marko Bjelonic of ETH Zurich[18] which uses the same code created by Redmon modified to work with ROS. This was done to give us greater configuration of the tool, but the pretrained network is still the same. All implementations are using the same pretrained model supplied by Redmon and accessible online [19]. Training is accomplished through multiple convolutional layers which learn to recognise objects, classify them then identify the size of objects with probability [15].

For all ROSBAGs to be used, a baseline run was recorded by replaying the ROSBAG and having a YOLOv3 instance read the supplied images with no processing. Output of YOLOv3 was captured to be used later. For each filter image tested, a subsequent replay of each ROSBAG was done, with the image transformation module subscribing to the ROSBAG image feed, publishing to a new topic which a YOLOv3 instance was configured to listen to. Again, the outputs of YOLOv3 are recorded to compare to the baseline and other filter runs.

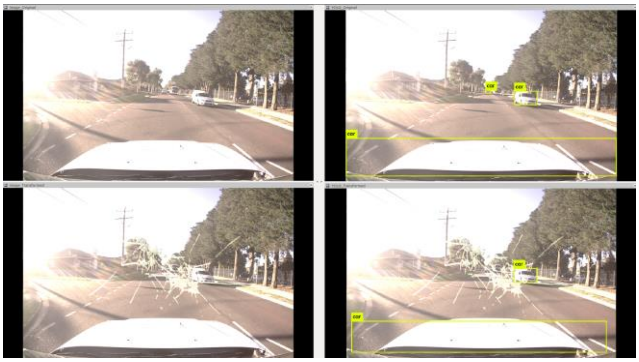


Figure 4.1 Screenshot of all video feeds considered. Top left: Original from ROSBAG. Top right: YOLOv3 output from original input. Bottom left: Crack transformation applied to original input. Bottom right: YOLOv3 output from transformed input

4.3 Extraction

Bjelonic's YOLOv3 publishes detection results onto 3 ROS topics. One is an integer with the total number of objects detected in a frame which was not used. The next is an augmented image feed, showing Bounding Boxes around classified objects.

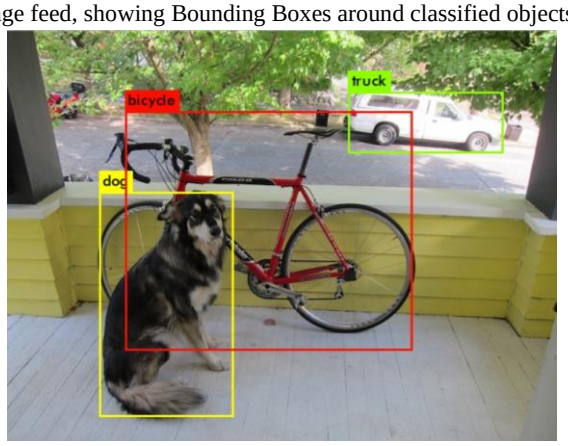


Figure 4.2 Example of YOLOv3 visual output clearly showing bounding boxes around recognised objects[19]

Finally, a custom defined message type is published including a range of data. A header object contains information about what message has been sent from YOLOv3, with a growing sequence number for every message sent; another header is included which corresponds to the header of the original image that YOLOv3 received; an array of BoundingBox objects is returned, one for every object detected in the frame. Breaking it down further, a BoundingBox object holds 6 properties, the object class identified, the confidence/probability the model has in the detection, and two sets of X/Y ordinates for the top left and bottom right pixels of the image to be used to draw a box over an image.

To read in data from the above topic, another module for detection results was created which accepts BoundingBoxes images and outputs to two comma separated files The first file contains a summary of each message received during a run with the supplied image sequence number, total number of objects

detected and the average probability of all objects detected. The other is a detailed breakdown of each image received with a row for every object detected (class and probability).

A further script is used to perform some final aggregation of results from the output csv files. For summary files, it exports out the total number of objects detected and the overall average probability calculated as:

$$\frac{\sum_{m=i}^{n} i_{Average\ Frame\ Probability}}{n - m}$$

where n is the last sequence number recorded and m is the first sequence number recorded

The above is required as the first captured sequence number is not always 1. Actual implementation counts every encountered row so in an exceptional case where a message is missed, the correct number of records is captured.

For detailed files, a similar aggregation occurs based on object class. Object classes are stored in a Python dictionary with each key holding a list of encountered probabilities. Once the detailed file has been read, for each distinct class encountered the number of detections is recorded with the average probability for that class.

4.4 Execution

To complete a testing run, each component is run in order from back to front (detection results, YOLOv3, image_transformation). All will then wait for input from the previous node. Once the ROSBAG begins playing, data will flow through each node and output to the csvs created. An overview of this model can be seen in Appendix B.

Files are then processed through the final analysis script with output figures grouped together by ROSBAG and/or filter for side-by-side comparison.

Given the intense computation required by YOLOv3, reasonably powerful computer equipment is required. In particular, Nvidia CUDA drivers are required to perform YOLOv3 computations on a GPU to provide better performance that is magnitudes faster than when performed on a CPU. All tests were carried out on a system with the following configuration of hardware and software.

CPU	Intel Core i7 8700 6C 12T@ 3.20GHz
GPU	Gigabyte GeForce GTX 1070 Ti Gaming 8G
Memory	16GB Corsair Vengeance LPX DDR4 2400MHz
Storage	Samsung 870 EVO 1TB NVME
OS	Ubuntu 16.04 LTS
ROS	Kinetic
Autoware.ai	v1.12.0
Nvidia Driver	384.130
CUDA Driver	9.0.176

5. RESULTS

From our findings, we have found that defects to the sensors resulted in two main pathways for how the object detection algorithm behaved. A subset of the filters that were applied to the input data resulted in generic obstruction of key objects present, which leads to a decrease in the objects detected. While the other subset of filters had an opposing effect and created objects that were not in the original camera feed, creating more false positives. These results were dependent on the type of filter that was used as well as the object that was detected. A filter that was obscuring for a particular object had the possibility of being misleading for another. Based on these two avenues, we compared the similarities these filters had on how YOLOv3 behaved and the amount of impact that these filters applied from the original input. The figures presented throughout this section are obtained from the results of the same ROSBAG.

5.1 Object Obstruction

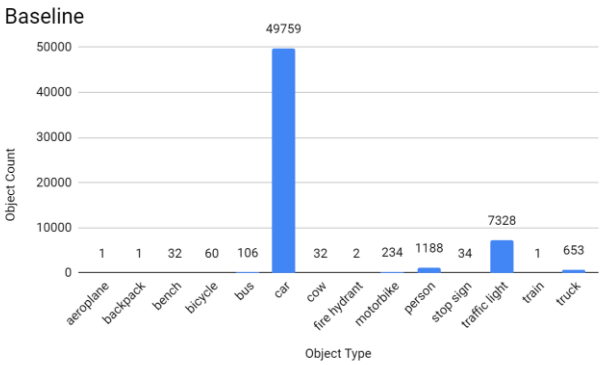
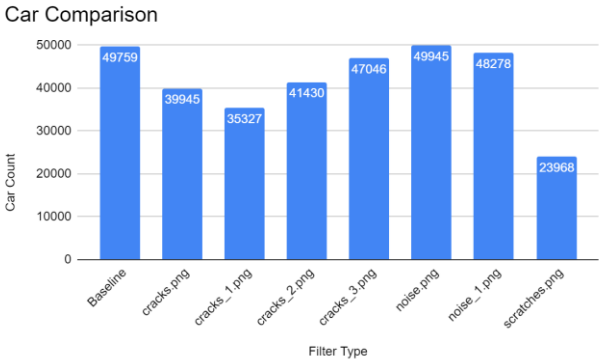


Figure 5.1 Number of object instances detected with no filters applied

Although most filters produced a decrease in some form or another, filters classified under this category present a similar trend in the decreases of key objects such as vehicles and road signs found. For each camera feed, an initial run without any filters gave results like the graph presented in Figure 5.1. We could then compare this data alongside any filter that was applied. Out of the filters applied, those that were generally larger defects would obscure the view of the camera more, which in turn obstructs objects that would normally be in view. Looking at Figure 5.2, different filters had varying effects on the number of cars that were detected. Applying the scratches.png for example resulted in only 48% of the original amount of cars

Figure 5.2 Number of cars detected across all filters for given ROSBAG (See Appendix A)



detected. Along with scratches.png, cracks_1.png also had a large decrease in the number of objects detected. Although not a direct correlation, this aligns with the amount of obscuration that a given filter had on the camera feed as seen in Table 5.1. More often than not, filters with a higher obscuration percentage were responsible for larger differences from the baseline run. This further illustrates the reasoning that this behaviour is mostly due to the larger defects that block out sections of the camera feed, hiding objects that would normally be detected.

As part of our methodology, runs were configured to have two different confidence thresholds, where by objects detected under this threshold are not reported. Using both 50% and 60% as confidence levels, larger filters exhibited similar behaviour across two different thresholds. As presented in Table 5.2, it can be seen that this large drop in crucial objects generally follows regardless of the threshold configured. Despite this general behaviour, it did not prohibit filters from also creating false positives, which can be seen by the number of instances of buses from the cracks_1.png filter.

Filter Type	Obscuration Percentage
cracks.png	3.48%
cracks_1.png	10.46%
cracks_2.png	5.85%
cracks_3.png	8.57%
noise.png	7.05%
noise_1.png	11.25%
scratches.png	11.81%

Table 5.1 Percentage of obscuration of each filter from Appendix A

Object Type	Cars 50%/60%	Trucks 50%/60%	Buses 50%/60%	Traffic lights 50%/60%
Baseline	49759/43680	653/475	106/101	7328/6553
scratches.png	23968/20314	523/317	85/71	5486/3768
cracks_1.png	35327/29612	385/277	1292/948	5373/4665

Table 5.2 Comparison of vehicles and traffic lights at 50% and 60% confidence across two filters

5.2 False Positives

False positives were more prevalent in filters that were not specifically obstructive of the view, but more so interfered with the overall quality of the camera feed. Due to this interference, objects that were not present or unclear objects in the baseline end up being detected as existing objects, such as cars or people. The noise filters was prevalent in acquiring objects that were not physically there. This is most likely due to the small scattering of dots and lines and the merging of them with other things to create false objects. With the introduction of false positives, that means that object detection methods can still be improved, as the introduction of objects that are not really present can cause the vehicle to abruptly stop or slow down when not needed. Furthermore, this could also disrupt other drivers around the autonomous vehicle.

The appearance of false objects can be seen in Figure 5.3. When using the noise filter on the same camera feed as the baseline, there is an increase in the number of people. Furthermore, filters that were more obstructive for cars such as scratches.png are now more misleading, detecting almost double the amount than the baseline. By introducing up to double the instances of people, these can affect the decision making of an autonomous car into braking when it would not have to. The presence of more unnecessary objects also reinforces the differing effects that filters have on object detection based on the object that it is being detected.

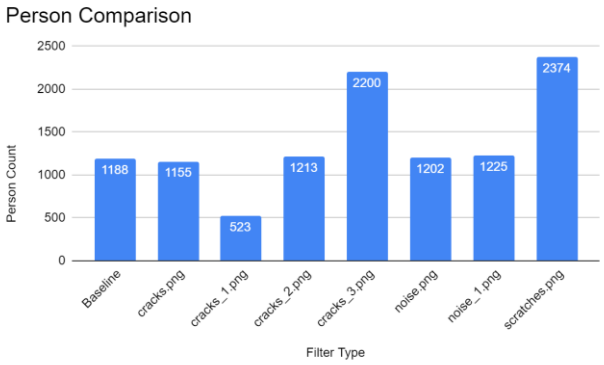


Figure 5.3 Number of people detected across all filters for a given ROSBAG (See Appendix A)

These results indicate that the nature of these types of filters are capable of maintaining a reasonable amount of confidence of the number of objects that are detected when compared to the baseline, however the additional outputs of other objects could cause issues in regards to decision making for the system as a whole.

5.3 Average Probabilities

When comparing the average confidence levels that YOLOv3 returned with different filters, most of them provided similar results to the baseline of a given run. For key objects that had larger object counts, the variance in average probabilities, albeit a few cases. As shown in Figure 5.4, the difference between the average probability from the baseline as compared to most of the filters is minimal. In this specific run, scratches.png created a large drop in confidence, this can mostly be attributed to a large increase in low confidence detections of people present. Despite this, across most objects, the change in average probability is not drastic for crucial objects. With the average confidence level

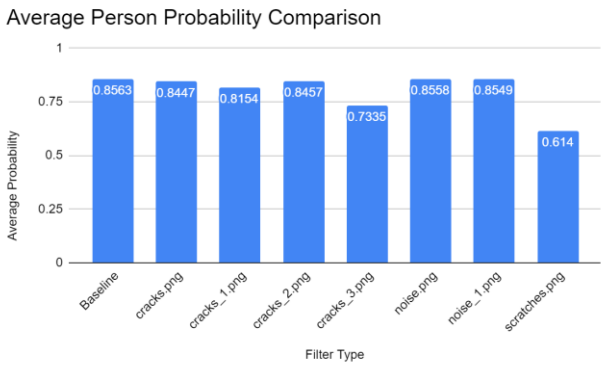


Figure 5.4 Comparison of average confidence level of people detected across all filters (See Appendix A)

having minor variances regardless of the more drastic changes in object counts demonstrates that YOLOv3 has a reasonable confidence of objects that it does detect.

5.4 Threshold Changes

Changes to the threshold were made to determine the behaviour of YOLOv3 at varying confidence levels and how it would impact our results. When comparing the same two runs with one threshold of 50% and the other with 60%, the average probabilities of crucial objects detected in the latter had increased from those detected at 50%. This was expected because as you increase the minimum threshold, calculation of the average has more occurrences of higher values. Changing the threshold also removed occurrences of false positives from the data. As seen in table 3, increasing the threshold has the capability of completing disregarding objects detected that would have otherwise been noticed at a lower confidence level. Yet although increasing the threshold, there are still occurrences of unnatural objects as the implementation of the object detection algorithm can still be improved to better replicate the decision making of human beings. Overall, increasing the threshold does accommodate in identifying and removing false positives, however the impact of defective data still remains prevalent in skewing data.

Object Type	Count/Average Probability at 50%	Count/Average Probability at 60%
Aeroplane	8/0.5724	3/0.6212
Bird	635/0.6012	275/0.6715
Boat	11/0.5434	0/-
Cow	31/0.7124	24/0.7627

Table 5.3 Comparison of irregular object count and average confidence level at 50% and 60%

6. RELATED WORKS

6.1 Automated Vehicle architecture

Success with autonomous vehicles in recent years can be attributed to deep learning models. [3] Research has been conducted into the effectiveness of existing deep learning architectures and has found that convolutional neural networks (CNN) such as AlexNet, VGG-16, and Inception-V3 show particular promise with regards to path planning.[20]
Taking in data streams from sensors mounted on a vehicle the CNN is able to make decisions on the appropriate steering input. [3] Since these networks play an integral part in the system of autonomous vehicles, its decision making capabilities must be thoroughly tested.

Other hardware required for autonomous vehicle systems include GPS, gyroscope, wheel speed sensors, cameras and radar systems. Cost effective off the shelf solutions are generally available with acceptable performance levels. However, issues with robustness come to the forefront if only a limited number of hardware is used. By increasing the number of devices, a layer of redundancy can be created providing a solution to this problem. [21]

6.2 Automated DNN testing

With deep neural networks playing a crucial role in the decision making of autonomous vehicles, plenty of research has been done to identify the robustness of such models. Tools such as DeepTest and DeepRoad have been developed to automate this process. With deep learning requiring minimal human interference, it can be difficult to understand the logic of a created model. This in turn makes identifying corner cases during testing even more difficult. However, neuron coverage has been identified as an effective metric over traditional code coverage. By leveraging the metamorphic relations between a system's behavior over testing images, new cases can be created to activate different neuron pathways.[22] [23]

Another technique of automating the testing process of DNNs is through the use of procedurally generated content. Randomly generated maps produced from simulators can be used to test the model of an autonomous vehicle. In particular, some papers examine the effectiveness of this by using the ASFAULT simulator. This was used to test the lane retention ability of a model. By stitching together pieces of road made from polylines random road networks were created. Another algorithm was created to check sections of the road which caused the car to deviate from the centre of the lane. Implementing these together provides an alternative to currently existing methods.[25]

Automated whitebox testing of deep learning systems also represents an effective method of testing and detecting incorrect corner cases. DeepXplore is a white box testing framework for autonomous vehicles which shows how to systematically test real world deep learning systems by introducing a neuron coverage metric. This measures the proportion of nodes activated within a DNN. Once test case creation is automated, this framework can be leveraged to automate the testing process of autonomous vehicle models.. Furthermore, DeepXplore uses metamorphic testing to demonstrate how to find inputs for DL systems which achieves high neuron coverage and corner case exploration. The problem with testing existing deep learning systems is that it depends heavily on manually labeled data which therefore fails to expose erroneous behaviours for rare inputs which helps to highlight the importance of our research gap. As previously stated, our filters will account for rare inputs to help understand exactly what is the effect on the data produced under certain circumstances such as damaged or obstructed camera lenses. [25]

6.3 Metamorphic Testing

Automating the testing process requires a method of easily creating new test cases. Where traditionally test cases are designed manually, DeepRoad, DeepTest and Deep Xplore[1] investigate techniques of transforming images to simplify the manual method. For example DeepRoad used this method to mimic weather conditions as mentioned in the background section of the paper. Specifically, general adversarial networks were used to automate this transformation process. This method of automated DNN testing pronounced promising results with the vehicle's models able to identify 100% of rainy, 85% of snowy images and 21% outliers among sunny images as invalid inputs.
[1][25]

The approach of comprehensively testing self driving cars can be tricky considering the near infinite number of potential scenarios from the combination of all the decision points within the convolutional neural network. [1]A branch of testing known as metamorphic testing involves evolving test cases based on output from previous test cases, significantly reducing the overall number of test cases having to be considered. This testing method can also be used to detect an error that was missed by previous cases. More importantly, metamorphic testing does not rely on a test oracle which is defined as a mechanism to determine whether a software has been executed successfully and this helps to once again reduce the number of test cases involved.
[1][26]

6.4 Image Distortion Algorithms

Research into image distortion algorithms is still in preliminary stages, where most image sets used are binary black and white.

Currently, as explained in our methodology above, the type of image distortion focused in this study is simulating life-like damages on the camera lens feed by applying filters layered on top of a live feed. These may include scratches, noise and cracks, simulating real scenarios. These distortions consistently alter the rational object detection mechanism which would be crucial in determining the decisions made by the car. [1]Calibration algorithms for high distortion lenses for TV screens have been researched, and with more time and resources these findings could potentially be used to generate a model which will help structure the layers of filters being applied to our input. The neural networks within autonomous vehicles are not currently able to qualitatively identify distortions within an image, unlike the human eye. [2] Hence, applying a universal image quality index would allow the system to have a reference point for what a clear object detection mechanism should look like as well as the basis for what a distorted object detection mechanism would appear to be. This index splits the distortion into 3 separate categories for easier identification. [1][27][2][28]

7. CONCLUSION AND FUTURE WORK

The amount of fatal and near fatal accidents involving autonomous vehicles has risen in recent years along with the increased adoption of them in society. With this comes the requirement to test these systems thoroughly to ensure these vehicles are in fact better than the average driver. Current testing revolves around general test cases where all systems are functioning. An area not deeply explored is where these the system is not operating at 100% where a sensor or multiple sensors could be defective with a crack on a camera lens being an example. The risk of lens crack resulting in the previously detected object not being detected anymore, now being classified incorrectly or even detecting objects that don't actually exist, have been discovered as per our findings. Even with having multiple cameras to improve detection. The defective cameras' feed will still be thought to be correct and used in classifying objects.

It has become prevalent in our findings that in order for autonomous cars to reach level 5 autonomy, improvements need to be made about reducing the impact of these failures. As these failures reduce the accuracy of the object detection drastically, potential consequences may be fatal.

The package presented in this paper, developed for ROS Kinetic, brings more awareness to such issues . It is readily available on GitHub and is a new testing approach which allows autonomous vehicle developers to test how their cars behave when sensor failure occurs. Due to the time constraints of this project, efforts were focused on the effect of a defect on a single camera feed and its effect when detecting objects on that feed. The codebase has been future proofed to easily allow for additional defects to be added in later.

Even with the preliminary results with a single transformed camera feed, differences in detection rates were observed. This work only touches on a single defect on a single sensor where there are theoretically infinite. In the future, without the time restriction placed on this project, planning has been done to implement other sensor defects such as additive drift in GPS data. The available package would then cover a large percentage of possible defects, but not all as there are infinite scenarios. Further automating our system and creating more test cases are potential improvements to be made, with the end goal being a ROS package which allows autonomous vehicle manufacturers test the robustness of their systems. This ensures that sensor defects do not inhibit the function of the vehicle and behaviours are adjusted under such conditions.

Observing the impact of a single sensor defect has on multi sensor systems such as Tesla's multi camera object detection would be the next step of research in this area.

8. ACKNOWLEDGEMENTS

This work was supported by the Faculty of IT, Faculty of Engineering and MCAV (Monash Connected Autonomous Vehicles) team at Monash University. We'd also like to thank our supervisor Aldeida Aleti for her support and guidance as we explored this field and research processes in general.

References

[1]"SAE International Releases Updated Visual Chart for Its “Levels of Driving Automation” Standard for Self-Driving Vehicles", Sae.org, 2018. [Online]. Available: https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles. [Accessed: 09- Nov- 2019].

[2] S. Ingle and M. Phute, "Tesla Autopilot : Semi Autonomous Driving, an Uptick for Future Autonomy", International Research Journal of Engineering and Technology, vol. 03, no. 09, pp. 369-372, 2016. [Accessed 6 September 2019].

[3] M. Bojarski et al., "End to End Learning for Self-Driving Cars", 2016. [Accessed 6 September 2019].

[4] H. Soo Chang, M. C. Fu, J. Hu and S. I. Marcus, "Google DeepMind’s AlphaGo", ORMS-Today, vol. 43, no. 5, 2016. Available: https://www.informs.org/ORMS-Today/Public-Articles/October-Volume-43-Number-5/Google-DeepMind-s-AlphaGo. [Accessed 6 September 2019].

[5] K. Grace, J. Salvatier, A. Dafoe, B. Zhang and O. Evans, "Viewpoint: When Will AI Exceed Human Performance? Evidence from AI Experts", Journal of Artificial Intelligence Research, vol. 62, pp. 729-754, 2018. Available: 10.1613/jair.1.11222.

[6] C. Urmson and W. Whittaker, "Self-Driving Cars and the Urban Challenge", IEEE Intelligent Systems, vol. 23, no. 2, pp. 66-68, 2008. Available: 10.1109/mis.2008.34.

[7] Z. Zhou and L. Sun, "Metamorphic testing of driverless cars", Communications of the ACM, vol. 62, no. 3, pp. 61-67, 2019. Available: 10.1145/3241979.

[8] Z. Zhou, D. Huang, T. Tse, Z. Yang, H. Huang and T. Chen, "Metamorphic Testing and Its Applications", 2004. [Accessed 7 September 2019].

[9] M. Zhang, Y. Zhang, L. Zhang, C. Liu and S. Khurshid, "DeepRoad: GAN-based Metamorphic Autonomous Driving System Testing", 2018. [Accessed 7 September 2019].
[10] Y. Tian, S. Jana, K. Pei and B. Ray, "DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars", pp. 6-9, 2018. [Accessed 7 September 2019].

[11] U. Lee, J. Jung, S. Jung and D. Shim, "Development of a self-driving car that can handle the adverse weather", International Journal of Automotive Technology, vol. 19, no. 1, pp. 191-197, 2017. Available: 10.1007/s12239-018-0018-z.

[12] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda and T. Hamada, "An Open Approach to Autonomous Vehicles", *IEEE Micro*, vol. 35, no. 6, pp. 60-68, 2015. Available: https://ieeexplore.ieee.org/document/7368032. [Accessed 9 November 2019].

[13] M. Quigley et al., "ROS: an open-source Robot Operating System", *ICRA Workshop on Open Source Software*, 2009. Available: http://www.willowgarage.com/papers/ros-open-source-robot-operating-system. [Accessed 9 November 2019].

[14] I. Culjak, D. Abram, T. Pribanic, H. Dzapo and M. Cifrek, "A brief introduction to OpenCV", *2012 Proceedings of the 35th International Convention MIPRO*, 2012. Available: https://ieeexplore.ieee.org/document/6240859. [Accessed 9 November 2019].

[15] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", 2016. Available: https://arxiv.org/abs/1506.02640. [Accessed 9 November 2019].

[16] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement", 2018. Available: https://arxiv.org/abs/1804.02767. [Accessed 9 November 2019].

[17] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks", *2017 International Conference on Communication and Signal Processing (ICCSP)*, 2017. Available: https://ieeexplore.ieee.org/document/8286426. [Accessed 9 November 2019].

[18] M. Bjelonic, "YOLO ROS: Real-Time Object Detection for ROS", *GitHub*. [Online]. Available: https://github.com/leggedrobotics/darknet_ros. [Accessed: 09- Nov- 2019].

[19] J. Redmon, "YOLO: Real-Time Object Detection", Pjreddie.com, 2019. [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed: 09- Nov- 2019].

[20] M. Teti, W. Hahn, S. Martin, C. Teti and E. Barenholtz, "A Systematic Comparison of Deep Learning Architectures in an Autonomous Vehicle", 2018. [Accessed 9 November 2019].

[21] M. Lundgren, E. Stenborg, L. Svensson and L. Hammarstrand, "Vehicle Self-Localization Using Off-the-Shelf Sensors and a Detailed Map", 2014. Available: https://ieeexplore.ieee.org/abstract/document/6856524. [Accessed 9 November 2019].

[22] Y. Tian, K. Pei, S. Jana and B. Ray, "DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars", 2018. Available: https://arxiv.org/abs/1708.08559. [Accessed 9 November 2019].

[23] M. Zhang, Y. Zhang, L. Zhang, C. Liu and S. Khurshid, "DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems", 2018. Available: https://dl.acm.org/citation.cfm?id=3238187 [Accessed 9 November 2019].

[24] A. Gambi, M. Mueller and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation", 2018. Available:https://dl.acm.org/citation.cfm?id=3330566. [Accessed 9 November 2019].

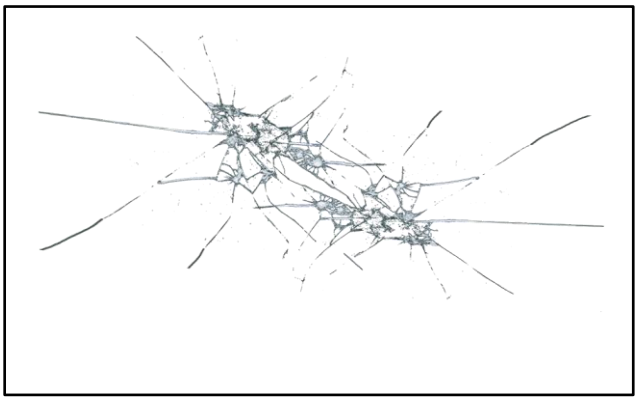
[25] "DeepXplore: Automated Whitebox Testing of Deep Learning Systems", 2017. Available: https://dl.acm.org/citation.cfm?id=3132785. [Accessed 9 November 2019].

[26]T. Chen, S. Cheung and S. Yiu, "Metamorphic Testing: A New Approach for Generating Next Test Cases", 1998. Available: https://www.cse.ust.hk/~scc/publ/CS98-01-metamorphictesting.pdf. [Accessed 9 November 2019].

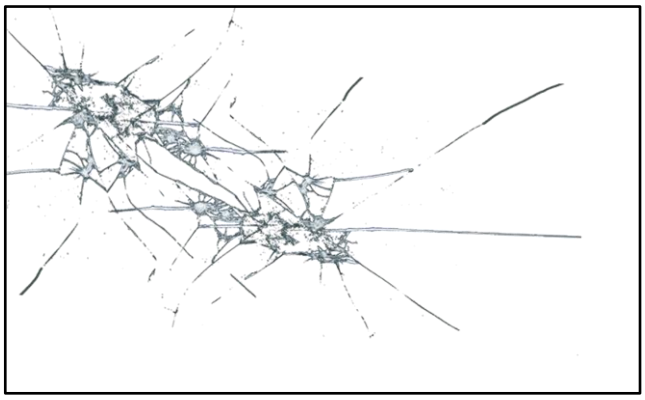
[27] M. Sagara, Y. Nomura, A. Ide and H. Naruse, "Simple Calibration Algorithm for High-Distortion Lens Camera", 1992. Available: https://www.computer.org/csdl/journal/tp/1992/11/i1095/13rRUyYjK5R. [Accessed 9 November 2019].

[28] Z. Wang and A. Bovik, "A universal image quality index", 2002. Available: https://ieeexplore.ieee.org/abstract/document/995823. [Accessed 9 November 2019].

Appendix A



cracks.png

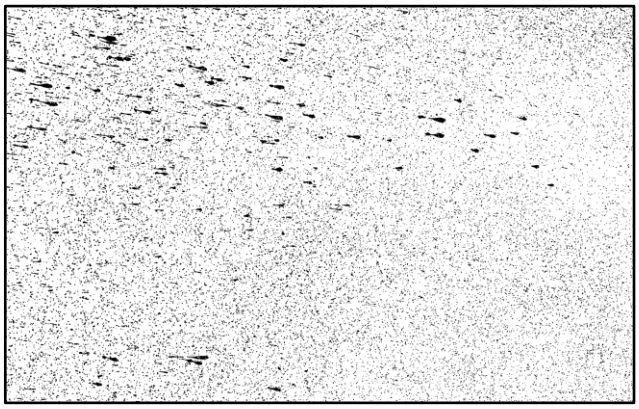


cracks_1.png

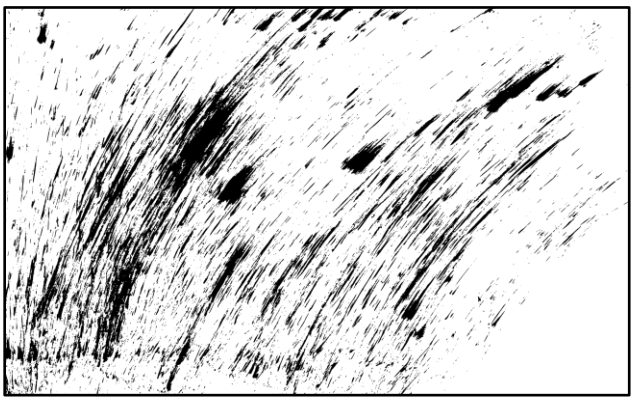
cracks_2.png

cracks_3.png

noise.png

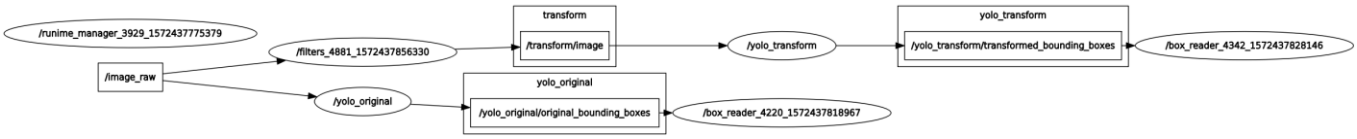


noise_1.png



scratches.png

Appendix B


/runime_manager_3929_1572437775379
/image_raw
/filters_4881_1572437856330
transform
/transform/image
/yolo_transform
yolo_transform
/yolo_transform/transformed_bounding_boxes
/box_reader_4342_1572437828146
/yolo_original
yolo_original
/yolo_original/original_bounding_boxes
/box_reader_4220_1572437818967