CNG 492 Final Report

CID: Can It Drive

Developing and testing holistic self driving AI in simulated environments

Daniyal Selani 2106268 || Mahmoud Hamraa 2174936

Constraints

Economical Constraints

This project required the development of multiple machine learning models. This included their training, validation, and deployment. It also needed these models to run concurrently alongside other systems as well as GTA V. The hardware available to us was not capable enough to handle such a load. We did not have and were not provided sufficient financial resources to get new hardware for the project.

Ethical Constraints

Video games have recently become an increasingly popular method for training AI systems to solve complex problems[1]. However, GTA V should only be used for personal learning and experimentation purposes on autonomous vehicle systems. Developing and distributing products using GTA V is considered illegal. *GTA V's* publisher, Take-Two Interactive, has shut down multiple high-profile projects that were using the game to develop new systems for commercial endeavors. Take-Two Interactive has even gone after academic researchers in some cases.

1. Introduction

In the field of computer science, the biggest revolutions have come about when the power to develop and explore new technology is given to not only the companies with large resource pools but also to individual developers, academics and students. This is what made the initial Windows Desktop so powerful and what has allowed it to dominate the PC market till this time, it is because individuals could develop and share their ideas across an easily accessible environment. In recent years, another example of such a revolution is the emergence of Machine Learning algorithms like Neural Networks. Even though Neural Networks have existed in concept since the 1950s, and institutes had implemented examples in the 1980s[1], they have had a recent surge in popularity because they have become relatively easy to access and use. Google and Microsoft both have invested in cloud solutions to allow individual developers to access hardware to implement Neural Networks on, and libraries like TensorFlow have made it infinitely easier for individuals to experiment with and learn from Neural Networks.

The rise in popularity of Machine Learning has led to a rise in autonomous cars. Projects like Google's Waymo Car project have shown that a completely autonomous car is feasible in the real world. This leads to a question on how these cars will be implemented, how they will be tested by institutions and governments, and how these cars will handle the concept of decision making.

We believe researching and developing a solution that can make it easier for an individual developer, student or academic, to develop and test their own solution is a way to bring about the revolution of self-driving cars. Defining an approach which is modular and makes it easier for autonomous and ethical issues to be validated is at the core of our mission.

We designed a solution (i.e. a suite of working modules) for making self-driving realizable, where our solution will serve as a testbed for experimenting with new ideas in this area. The solution's architecture is modular with the intention of a flexible environment and framework in mind, where the practitioners may implement and integrate their own modules easily to our framework. For the virtual environment, we are employing GTA 5 because of the tools, flexibility and availability it provides. The objective is not to develop a fully polished and marketable solution, but rather to have the ability to test the efficacy of any new ideas regarding a self-driving solution.

NOTE: For the final report, for the sake of length and readability, we are going to give a general high-level overview of the new work done on the system this semester. We will try not to repeat the details that we already repeated in the previous 491 reports. Hence, some of the technical detail which has already been mentioned in the previous report will not be mentioned here.

2. Problem definition and scope

According to the annual report released by the European Road Safety Observatory (ERSO), in 2016 alone, there were more than 12,000 car passenger and driver fatalities. Approximately 71% of these fatalities occurred during dry weather. Which implies that nearly 71% of these fatalities were caused by human error. Self-Driving cars provide the possibility of reducing the number of fatalities caused by human error to 0. [nc]

Self-driving cars cannot only make driving safer but also cheaper and more efficient. Even with all these benefits, companies have still struggled to implement a fully autonomous driving solution for the masses. Self-driving systems are complicated, and developing and testing them requires a vast amount of resources. Hence, meaningful progress in this space has been slow.[nc]

Self-driving systems are difficult to develop because of their inherent complexity. Driving is a complex task. A driver, human or otherwise, must be responsible for multiple tasks, such as navigation, car steering, throttle control, and making decisions based on the current state of the world around it. As these tasks are highly interlinked, it is very hard to develop a stand-alone system that can complete one of these tasks on its own. It is still harder to implement and test any solution, without having either a driving simulation or an actual car with all the required sensors installed. Hence, there is a need for an approach to the development and testing of holistic self-driving car systems to be more accessible.

The objective of our research is to design a solution which makes the testing and development of a **holistic self-driving car and its systems** easier and more accessible. The goal is to not produce a fully polished and marketable solution but to make it possible for someone to implement and test their novel solutions. Our approach is to create a modular design for the system. So that anyone interested can tackle any aspect of the self-driving problem without having to change the whole system. We shall use the virtual world of GTA V as our simulation environment, for testing and developing the system. We plan to implement our design and observe how much progress can be made with this modular approach.

Our project can be broken down into 3 aspects. Researching and developing a solution that:

1. Is modular in design for easy upgradability, interpretability and debugging
2. Can be easily deployed on consumer grade hardware for testing and validation
3. Outlines an approach for debugging the behavior of the autonomous system that is developed.

Note: We define consumer grade hardware as at least one PC that is fitted with the following minimum requirements for the relevant parts:

- 16GBs RAM
- GTX 1060 Graphics Card
- Quad-Core processor from Intel or AMD (should be of a recent generation)

These requirements were decided upon after doing a literature survey into this topic.

2.1 Managing the scope of the project

We are aware that the scope of our project is broad, and it is easier if we approach the project by diving into distinct parts. Each of these parts will be discussed in more detail in the rest of the report.

1. Literature Review
2. Design and implementation of modules
3. Simulated environment and modifications.

3. Related Work

This section discusses all the important and relevant projects that are related to our own project. In this section, we will go over multiple projects being developed by both various companies and individuals and provide our criticism and view on them.

3.1 Google and Comma.Ai

Google has been developing its own system, called Waymo [26]. Their technology depends on highly accurate maps of the area their vehicle is allowed to drive in. Google has enough resources to map almost every street and alley in the world. Their maps have very precise information about road layout, stop signs, traffic signals, and everything else required for a self-driving car to navigate smoothly on the road.

Another company, called Comma.AI [25], has built an open-source modular architecture design which is very akin to our own, named OpenPilot. The system has *controlsd*, which is the main module that talks to the car. The system has other modules: *boardd*, *sensord*, and *visiond* which all talk to the outside world. *loggerd* logs all the data to use for machine learning and *plannerd* tells the car where to go. However, the entire system is still in beta and is geared more towards early adopters and enthusiasts[l1] .

Google's Waymo is struggling with lane changes [27]. Waymo still lacks a human driver's ability to anticipate other drivers' actions and squeeze into an open spot when it wants to change lanes. Their system also seems to struggle with the ability to react to events and plan the future course of actions. Comma.AI's OpenPilot is a level 3 autonomous system [28]. The company uses Hotz's version (company founder) of lane-keeping and adaptive cruise control that does not forego the human driver behind the steering wheel.

Moreover, only such large corporations with huge assets can afford to test autonomous vehicles. They are testing their cars by deploying a full set up of various complex sensors since they are not using simulation environments. Setup for one car can cost up to $75,000 according to [23]. This comes on top of the price of the vehicle itself. Consequently, developing and mass producing such a system can prove very costly and extremely challenging. Additionally, the cost of developing and testing an autonomous system is not the only problem. The industry is facing far more complex dilemmas such as setting up and maintaining a suitable infrastructure, perfecting the nuanced social interactions these systems have to take into account on the road, equipping the systems to deal with bad weather conditions and cybersecurity [24].

3.2 Princeton papers and Psyber.io

In a paper published by a team at Princeton University [30] titled “*Using Virtual Worlds, Specifically GTA V, to Learn Distance to Stop Signs*”, GTA V was used to train and test a CNN in detecting stop signs in different environments, weather conditions, and times of the day. Their model is fairly successful in recognizing various stop signs to an extent. However, the accuracy falls sharply as the distance increases.

In another paper published by a team at Princeton University [31] titled *“Beyond Grand Theft Auto V for Training, Testing and Enhancing Deep Learning in Self Driving Cars”*, the team generated over 480,000 labeled virtual images of normal highway driving in Grand Theft Auto V's virtual environment. Using these images, a CNN was trained to detect the following distance to cars/objects ahead, lane markings, and driving angle: all variables necessary for basic autonomous driving. The team obtained encouraging results when they tested over 50,000 labeled virtual images from substantially different GTA-V driving environments. This initial assessment begins to define both the range and scope of the labeled images needed for training as well as the range and scope of labeled images needed for testing the definition of boundaries and limitations of trained networks. It is the efficacy and flexibility of a "GTA-V"-like virtual environment that is expected to provide an efficient well-defined foundation for the training and testing of CNNs for safe driving.

Another project, called *Charles* and created by *Psyber.io,* uses GTA as a simulation world to train a vehicle to stay on the road, avoid obstacles and follow map routes to various pre-set waypoints, like a GPS guidance system. The model is based on a frame-by-frame basis with no memory and has shown some positive results [29].

The predictive models cannot be seen as holistic self-driving solutions on their own. They do not provide any decision-making capabilities, and they act as a black box function that takes in an input image and tries to predict the output based on the data that was used to train them. They merely imitate the behavior of a driver, rather than making any meaningful decisions from the information gathered. If these models are presented with a situation they have not encountered before in their training data, they will perform poorly. Additionally, since they behave like black boxes, they are very hard to debug and interpret. Trying to alter the behavior of these models requires altering the training dataset, or the architecture of the model and retraining it. Which is a computationally expensive, and time intensive procedure.

3.3 DeepGTAV

DeepGTAV is an open source project on Github. It is provided for anyone to try in the form of a downloadable plugin that transforms GTA V it into a vision-based self-driving car research environment. It has been used by other universities and institutions to perform research on autonomous cars using GTA V.

As this is a community-supported project that relies on GTA V, any official updates released for GTA V can break the code[2].

3.4 AirSim

AirSim is a project that can be used as a simulator for drones, cars and more, built on Unreal Engine and Unity. It is open-source, cross-platform, and supports hardware-in-loop with popular flight controllers such as PX4 for physically and visually realistic simulations. It is developed as an Unreal plugin that can simply be dropped into any Unreal environment. AirSim is a platform for AI research to experiment with deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles.

Moreover, APIs can be used to retrieve images, get state, control the vehicle and so on. The APIs are exposed through the RPC, and are accessible via a variety of languages, such as C++, Python, C#, and Java. In regards to collecting data, there are two ways to collect data from AirSim for deep learning. The first and easiest way is to simply press the record button in the lower right corner which will begin writing pose and images for each frame. The second way is by accessing the APIs. Which allows full control over how, what, where and when you want to log data.

4. Literature review

This section provides a description, summary, and critical evaluation of the most important works in relation to our project.

4.1 Five levels of autonomy

To understand what autonomous vehicles are and what is expected of them, one must understand the 5 levels of autonomy and what each level means[4].

Level One – Driver Assistance

The vehicle offers assistance with some functions, however, the driver still has to handle accelerating, braking, and monitoring of the surrounding environment. A vehicle at this level can do basic operations such as braking a little harder when there is an object in front of it.

Level Two – Partial Automation

At this level, the vehicle can assist with steering or acceleration which allows the driver to disengage from some of their tasks. But the driver must always be ready to take over control of the vehicle and is still responsible for most safety-critical functions and all monitoring of the environment.

Level Three – Conditional Automation

Starting at this level, the vehicle itself controls all monitoring of the environment using sensors like LiDAR. The driver's attention is still required at this level. However, they can disengage from "safety critical" functions like braking and leave it to the technology when conditions are deemed safe. Many vehicles at this level require no human attention to the road at speeds under 37 miles per hour.

Level Four – High Automation

At Levels 4 and 5, the vehicle is capable of almost everything. A level 4 autonomous vehicle should be capable of steering, braking, accelerating, monitoring the vehicle and roadway as well as responding to events, deciding when to change lanes, turn, and use signals.
At this level, the autonomous driving system would first have to notify the driver when conditions are safe, and only then does the driver switch the vehicle into this mode. It cannot make decisions in more dynamic driving situations like traffic jams or a merge onto the highway without feedback from the driver.

Level Five – Complete Automation

This level of autonomous driving requires basically no human attention at all. This means that there is no need for pedals, brakes, or a steering wheel, as the autonomous vehicle system has full control over all critical tasks, monitoring of the environment and identification of unique driving conditions like traffic jams or special weather conditions.

4.2 What autonomous vehicles mean for the future

According to [5] self-driving technology is becoming more popular and could completely redefine our transportation system (and by extension, our economy, and society). Based on automaker and technology company estimates, level 4 self-driving cars could be for sale in the very near future. That is several years from now.

However, the costs and benefits of self-driving cars are still largely vague. More information is needed to fully assess how they'll impact drivers, the economy, equity, and environmental and public health.

The costs and benefits of self-driving cars are still mainly hypothetical. More information is required to fully understand how they'll impact drivers, the economy, equity, and environmental and public health.

Regarding safety, thousands of people die in motor vehicle crashes every year in the United States (more than 30,000 in 2015). Self-driving vehicles could, essentially, bring that number down. The software could prove to be less error-prone than humans. However, cybersecurity will cause another major concern whenever software is introduced.

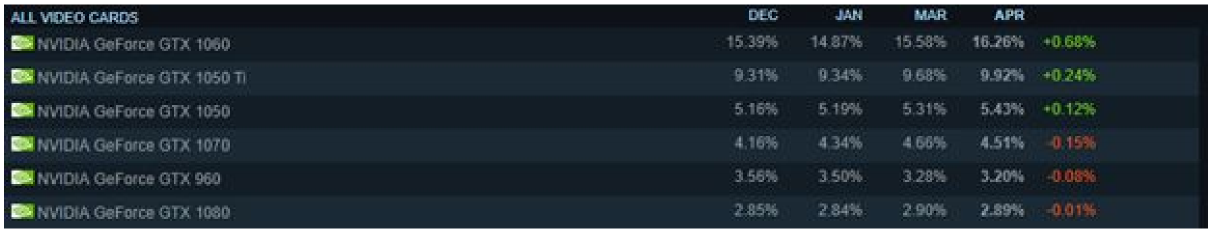
Equity is another major consideration. Self-driving vehicles could help individuals who are unable to drive themselves, such as the elderly or disabled. But the introduction and widespread adoption of autonomous vehicles could also lead to millions of Americans employed as drivers to lose their jobs.

In addition, environmental impacts are a serious concern according to ("Self-Driving Cars Explained", 2018). Accessible, affordable, and convenient self-driving cars could increase the total number of miles driven each year. If those vehicles are powered by gasoline, then transportation-related climate emissions could dramatically increase. On the other side, if the vehicles are electrified and become more widespread then transportation emissions could drop, perhaps significantly as the paper argues.

4.3 Most commonly used hardware

8.3.1 By PC Gamers

Steam has 1 billion users and 90 million active users [7]. In a survey conducted by the video game digital distribution platform, the most used GPUs among PC gamers were collected [6]. The results are as follows:



ALL VIDEO CARDS	DEC	JAN	MAR	APR	
NVIDIA GeForce GTX 1060	15.39%	14.87%	15.58%	16.26%	+0.68%
NVIDIA GeForce GTX 1050 Ti	9.31%	9.34%	9.68%	9.92%	+0.24%
NVIDIA GeForce GTX 1050	5.16%	5.19%	5.31%	5.43%	+0.12%
NVIDIA GeForce GTX 1070	4.16%	4.34%	4.66%	4.51%	-0.15%
NVIDIA GeForce GTX 960	3.56%	3.50%	3.28%	3.20%	-0.08%
NVIDIA GeForce GTX 1080	2.85%	2.84%	2.90%	2.89%	-0.01%

Fig 4.1: Graphics Card usage statistics

As can be seen from the numbers, GTX 1060(6 GB VRAM) is the most popular graphics card among PC gamers.

The picture below shows a comparison between the Quadro K620, which is the GPU provided in the laboratory, and the most popular GPU on the market (GTX 1060). The latter outperforms the former by a quite substantial margin.

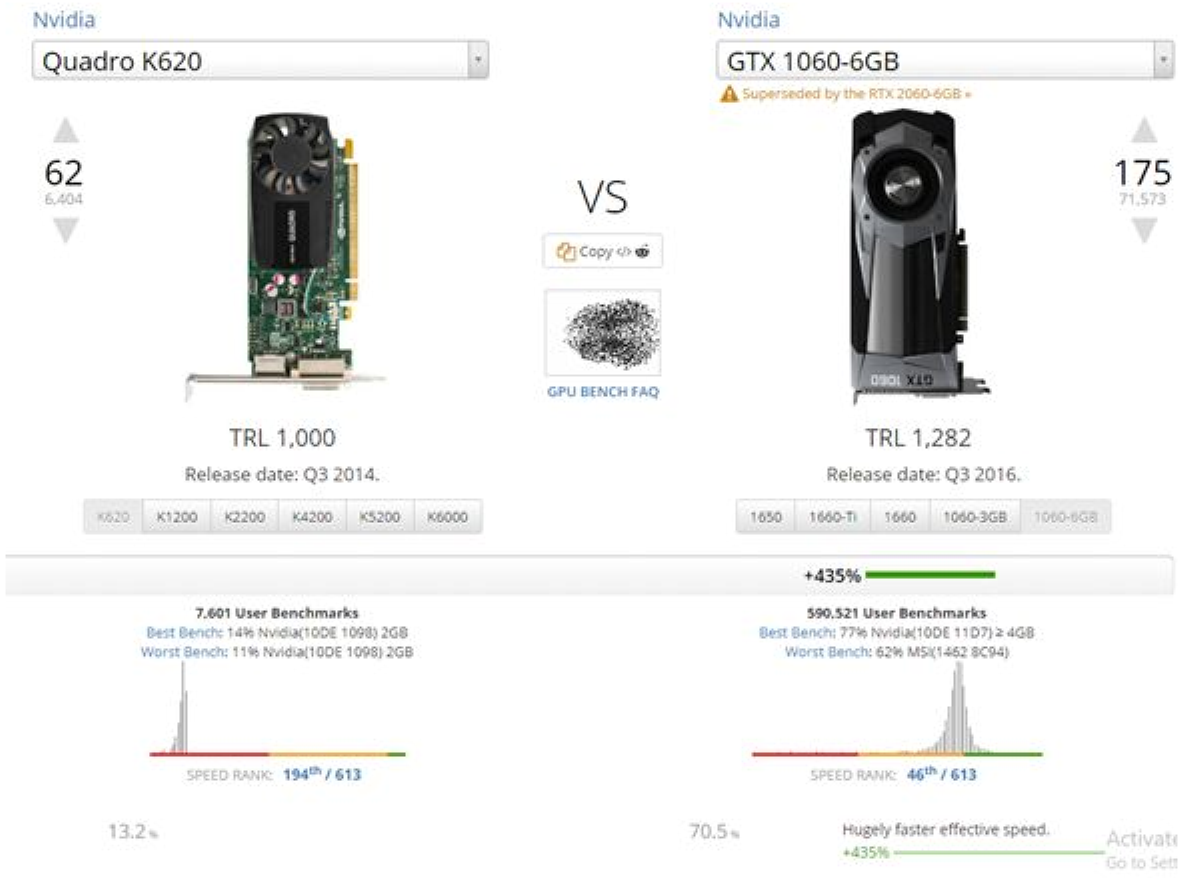


Fig 4.2: Graphics Cards Comparison

8.3.2 By ML/DL enthusiasts

Lambda labs have tested the best GPUs for machine learning and deep learning have come out with a couple of interesting conclusions [8]. The GPUs tested were:

- Titan RTX
- 1080 Ti
- Titan Xp
- Titan V
- 2080 Ti

· V100

After rigorous testing, they reached the following conclusions:

· RTX 2080 Ti is the best GPU for Machine Learning / Deep Learning if 11 GB of GPU memory is sufficient for your training needs. The 2080 Ti offers the best price/performance among the Titan RTX, Tesla V100, Titan V, GTX 1080 Ti, and Titan XP.

· Titan RTX is the best GPU for Machine Learning / Deep Learning if 11 GB of memory isn't sufficient for your training needs.

· Tesla V100 is the best GPU for Machine Learning / Deep Learning if the price isn't important, you need every bit of GPU memory available, or time to market of your product is of utmost importance.

Note: Please refer to their website for a more detailed explanation of the testing methodology.

5. Research Method

Our approach to research has changed from what it was during the previous semester. This is mostly due to feedback from our advisors and professors, from observations made during an extensive literature survey, and from our experience working on this project. Previously, our focus was on a system that somehow eased the development and testing of ethical behavior in self-driving cars. However, we realized that this approach was too broad and somewhat out of the scope of computer engineering, as we would first need to define ethical behavior. Instead, we shifted our focus on the individual tasks that autonomous driving systems perform.

Deploying and testing autonomous vehicle systems in GTA V has been attempted before (As explored in the related work section). However, these projects only complete some or one of the tasks required for self-driving systems. Our aim is to outline how a **holistic self-driving system** might be developed, deployed and tested, all within the limitations of consumer-grade hardware and GTA V.

We define a holistic self-driving system as a system that can perform all the tasks that are necessary for a self-driving vehicle to execute to function correctly. We have identified the tasks as follows in sequential order [9]

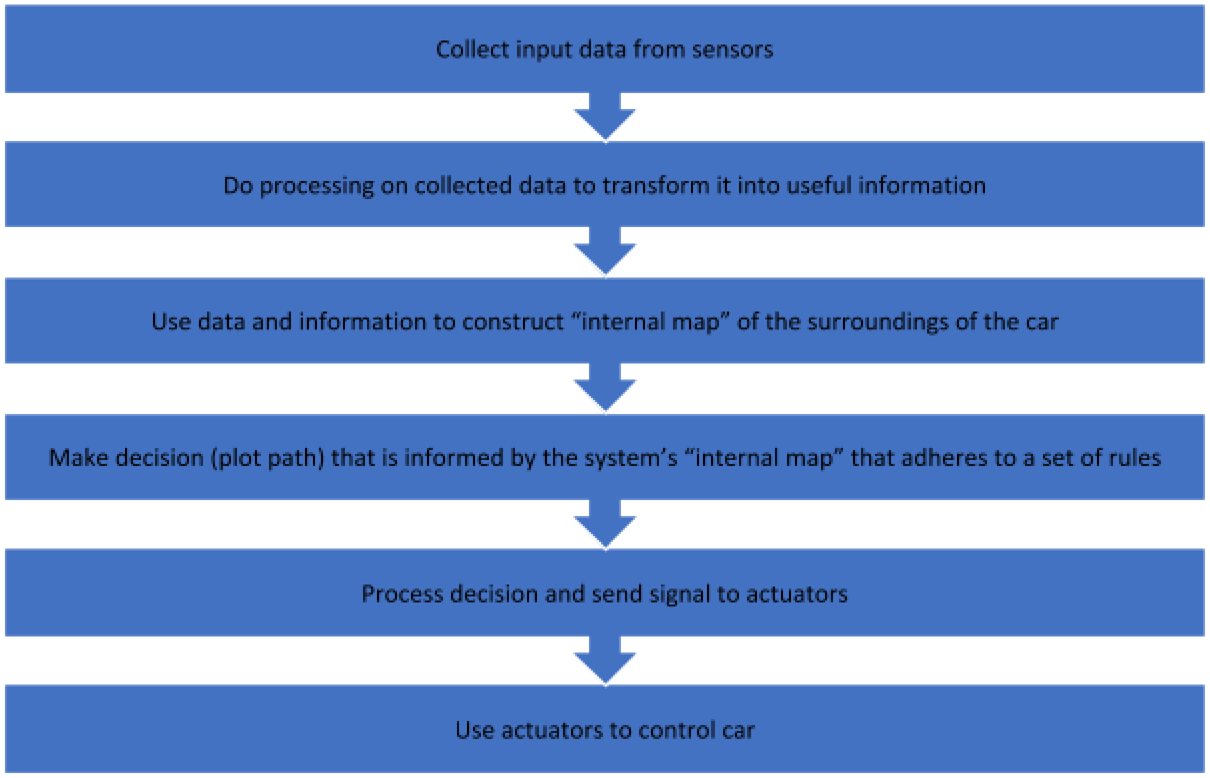

Collect input data from sensors
Do processing on collected data to transform it into useful information
Use data and information to construct “internal map” of the surroundings of the car
Make decision (plot path) that is informed by the system’s “internal map” that adheres to a set of rules
Process decision and send signal to actuators
Use actuators to control car

Fig 5.1: Tasks for self-driving systems

We define “internal map” as the abstracted information that CID sees about its surroundings in relation to itself.

The modifications that we have made to the above sequences of tasks, is that we establish a base level autonomy for CID. Any decision made is then used to alter the behavior of this base level behavior.

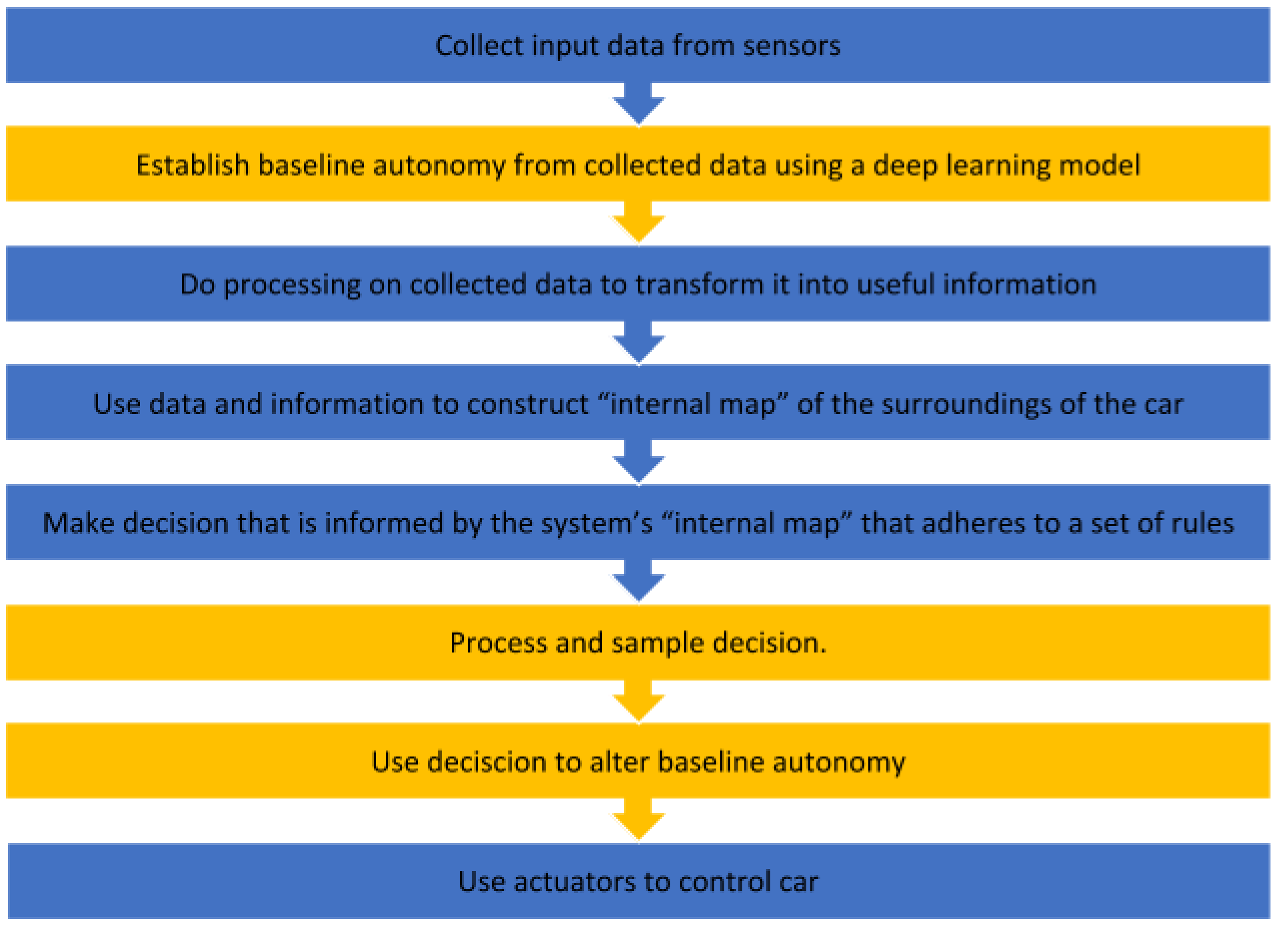

Collect input data from sensors
Establish baseline autonomy from collected data using a deep learning model
Do processing on collected data to transform it into useful information
Use data and information to construct "internal map" of the surroundings of the car
Make decision that is informed by the system's "internal map" that adheres to a set of rules
Process and sample decision.
Use deciscion to alter baseline autonomy
Use actuators to control car

Fig 5.2: Tasks for proposed self-driving system

These new tasks is our major innovation in this research project. It is easier to alter and build upon the behavior of an already autonomous model, rather than trying to integrate all decisions made into a cohesive behavior. It is easier because, as mentioned in the related work section, there already have been projects that use deep learning tools to make predictive models for self-driving cars in GTA V.

This assertion can be justified by giving the following example: A neural network trained to mimic the driving behavior of a user already knows how to apply the brakes. It is relatively easy to train this neural network to obey a brake command and to brake appropriately (given the brake command and its current input from the game). It is harder to make a system that makes multiple decisions based on the data provided and then integrates the decisions into cohesive behavior. Driving is a complex task, and simply braking at the appropriate time is not good enough. These smaller decisions can be learned as behavior by the neural network, while the larger decisions can be made by an external system/agent specifically designed to make these decisions. If these decisions are provided to the trained model as additional information, the baseline behavior of the model can be altered. We know this to be true due to the literature survey done (Related work), and our experience from work already done on this project.

Our design shall complete all the tasks mentioned above, and we will implement modules in our system, that demonstrates (at least on a superficial level) how these tasks can be executed. We went with a breadth vs depth approach when implementing the system and its modules. We focused on broadening the breadth of the system's capabilities, rather focusing on the depth of the system's capabilities. We implemented modules to demonstrate the completion of as many tasks as possible, rather than trying to make each individual module as capable and efficient as possible. The system's design and the design of its modules were altered from the previous iteration so that each module would be responsible for at least 1 distinct task.

For a lot of the modules, we employ pre-existing work that was previously done and that is modified to suit our need. In fact, that is one of the strengths of our approach, to be able to quickly integrate already existing work into our system.

6. System Design

In this section, we will go over the intended design of our system.

6.1 Architecture

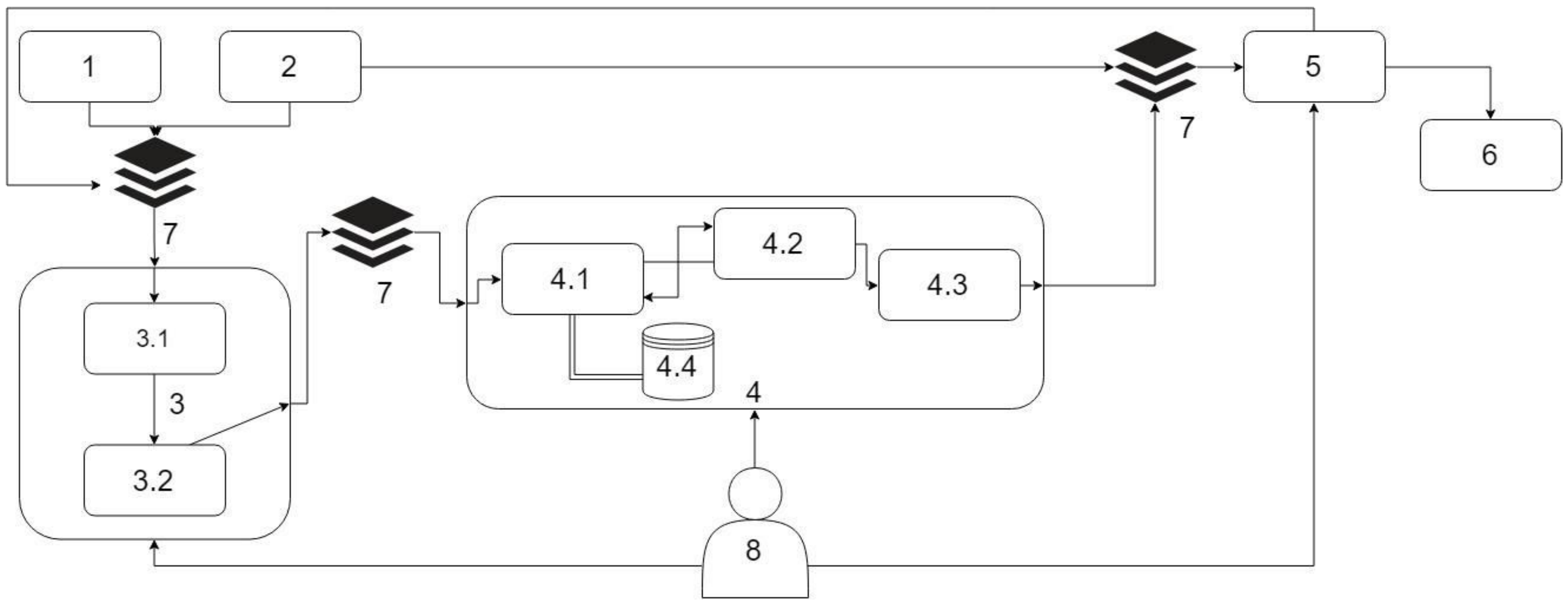


Fig 6.1: System architecture

6.1.1 Modules and their descriptions

We define a single unit of time for CID as a single frame that is captured from GTA V. This is how all the modules are kept in sync, each module executes its described task at every unit of time.

1. Data capture module

    Communicates with MODS implemented in GTA V to collect in-game data.

2. Screen capture module

    Captures frames from GTA V and transforms it to the correct dimensions.

3. Data extraction module

3.1 Data filter

    Receives data from 1, 2, and 5, and filters in only relevant data

3.2 Data processing

    Receives filtered data and transforms it into useful information

4. Decision-making module

4.1 Internal map constructor

Takes in processed information for each frame and constructs an internal map

4.2 Decision-making agent

Agent exists within the constructed internal map and makes decision/plots path.

4.3 Decision sampler

After the agent has plotted path/made its decision, the sampler outputs the final decision to the next module

4.4 Knowledge base

Structured representation of knowledge that the Map constructor uses to construct an internal map.

5. Navigator

Establishes baseline autonomy based on captured frames from the game. Its behavior can be modified by the output from the decision-making module

6. Control module

Takes outputs from the navigator and turns it into signals that the “actuators” in the car can interpret to alter the car’s movements.

7. System buffers

Buffer the information passed between modules for the previous n units of time. This allows CID to have a memory of previous events.

8. Human input

Input from system developer relevant to the system at run time, that cannot be programmed into the system.

7. System implementation

In this section, we will go through, in detail of the reasoning behind the design and implementation (or lack thereof) of our system and it’s modules. We will also mention other possible implementations of the same module.
Also it is to be noted, that we could not integrate all the modules into one system because of the hardware available to us.

7.1 Modules

The data collection and screen capture modules perform the task of collecting data from the “sensors” of the car. In our implementation, we substitute sensors with mods which can collect in-game data, and capture the frames of the game itself.

7.1.1 Data collection module

We will discuss the exact MODS used to extract in-game data from GTA V, and how these MODS function in a later section. In this section, we will only discuss how CID communicates with these MODS.

We created a client-server system for the mods to communicate with a custom python script via UDP messaging protocol. The script sends an initial message to the mods to establish a connection and to inform the mods (server side) that the module (client side) is ready to receive data. This can be seen as a simplified version of the handshake protocol. The module always initiates the handshake, and the connection is terminated if either a stop message is sent, or the client side or server side shuts down. Only the client side can send a termination message. The client side can also specify certain parameters that the mod must return. Such as:

1. Frame data
2. Throttle activation
3. Steering angle
4. Vehicle speed
5. Vehicle orientation
6. Rendered objects orientation
7. In game location
8. In game time

The module collects data, performs some error handling on the received data and it passes it forward to the next module as a python dictionary.

7.1.2 Screen capture module

This module is a python script that captures the screen image of the computer upon which GTA V is being run on. It captures a region of the screen within which the game is being played. The maximum and minimum frame rates of the screen capture can be set so that the captured frames are being sent to the next modules at a consistent rate

7.1.3 Data extraction module

This module performs the task of taking in data from “sensors” and transforming into useful information. We define useful information as information that describes the state of the environment the car is in, relative to the car. The module also is passed the most recent action taken by the system, to provide a possible feedback loop for the system to evaluate its previous actions. The first stage is to simply filter out the data that is seen as extraneous and not necessary.

It is important to ensure that this module provides a certain level of abstraction from the raw data that is being passed to it by the “sensors”. This is because not all of the data is relevant to the decision-making process, and filtering out unwanted data reduces the complexity of the modules that have to make decisions based on this data. Additionally, the abstraction of data makes it easier to interpret the state of the environment of the car and to debug and possible bad decisions made by the system using this transformed information.

When a Tesla vehicle detects a crash/anomalous behavior that has occurred while the car was in autonomous mode, it captures a snapshot of the environment leading up to that event. This data is meant to be sent to Tesla's central servers for processing and debugging. However, the data is abstracted first. A crude 3D rendering of the snapshots is produced, and that rendering, along with relevant car information (speed, steering angle, throttle position), is sent to the servers [10]. We intend to implement something similar to this module to ease interpretability and debugging for the system.

From the data collected from mods, we filter out parameters such as pitch/yaw rate of the car. The elevation of the car, in-game time and the exact location of the car within GTA V's world. We filter out this information as we do not use it in the decision-making process.

Since the data is collected from 2 different sources (mods and screen capture frames from GTA V), we split the data extraction task accordingly.

GTA V mods

The filtered in data included lane markers of the road the car was traveling on and the speed of the car itself.

We created a function that scored the deviation of the center of the car from the center of the lane.

```
def laneDelta(carX, laneX):
    return(carX - laneX)
```

The more positive the score, the more the car had deviated to the right of the lane, and the more negative the score, the more the car had deviated to the left of the lane.

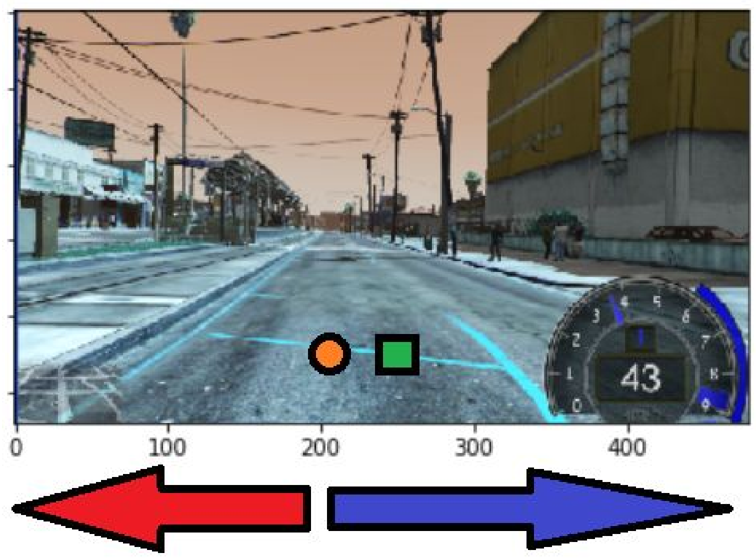


Fig 7.1: Visual representation of lane centering

The orange circle in the picture above represents the center of the lane, the green square represented the center point of the car. The score is calculated between these 2 points, the arrows were added to indicate that the input has a positive value (larger blue arrow), and so the car is deviating to the right.

We used the instantaneous speed of the car to detect any collisions between the car and it's surroundings. We do this by calculating the impulse acting on the car at the current time.

Impulse = $\Delta p = p_{final} - p_{initial}$

Where

$$p = mass * velocity$$

We substitute Pfinal with Pt, where Pt is the instantaneous speed of the car at time t, and Pinitial with Pa, where Pa is the average instantaneous speed of the car at time t. We simplify this equation by ignoring mass (as we cannot ascertain the mass of the object that the car collides with, and we only care about the existence of an impulse, not it's magnitude). If the resulting impulse > Threshold T, we can assume a collision has occurred. Furthermore, by factoring in the distance vector, we can perform this calculation for the true velocity of the car, and estimate the direction of impulse, hence the nature of the collision as well.

Unfortunately, due to reasons that will be described later, we were not able to fully use these methods or develop more methods using mods in our module and hence they had to be discarded from the final demonstration and implementation.

Screen captured frames

Autonomous systems throughout the industry, including those in productions, all rely heavily on machine vision to perform core tasks. Hence, we spent a lot of time on how to mine as much data as possible from each frame.

The first hurdle facing us was simple. Modern cars with autonomous systems have multiple cameras installed at multiple angles to get the highest fidelity inputs (Tesla Model 3 has 8 cameras installed). While playing GTA V, we have only one camera perspective, the perspective of a camera installed on the windshield.

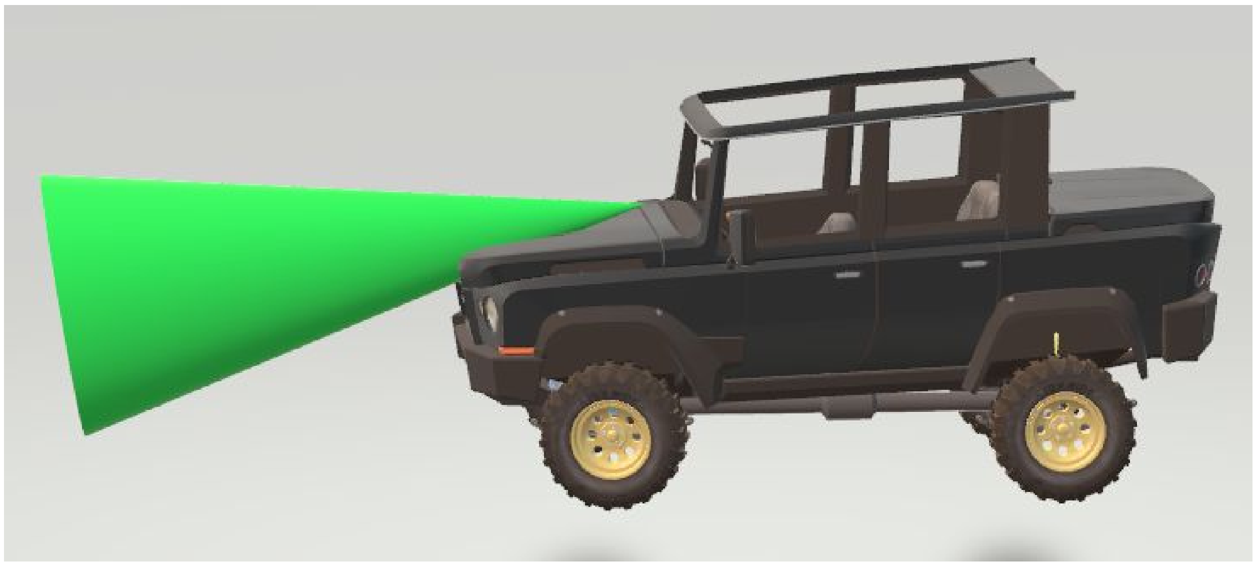


Fig 7.2: Visual representation of in-game camera angle

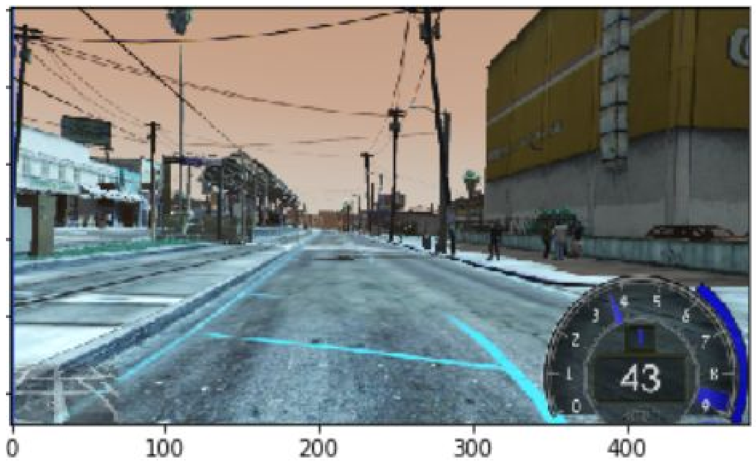


Fig 7.3: Sample   image from the default camera angle

This limits the amount of data that can be extracted from each frame. We struggled especially with segmenting lanes through computer vision because of this fixed POV.

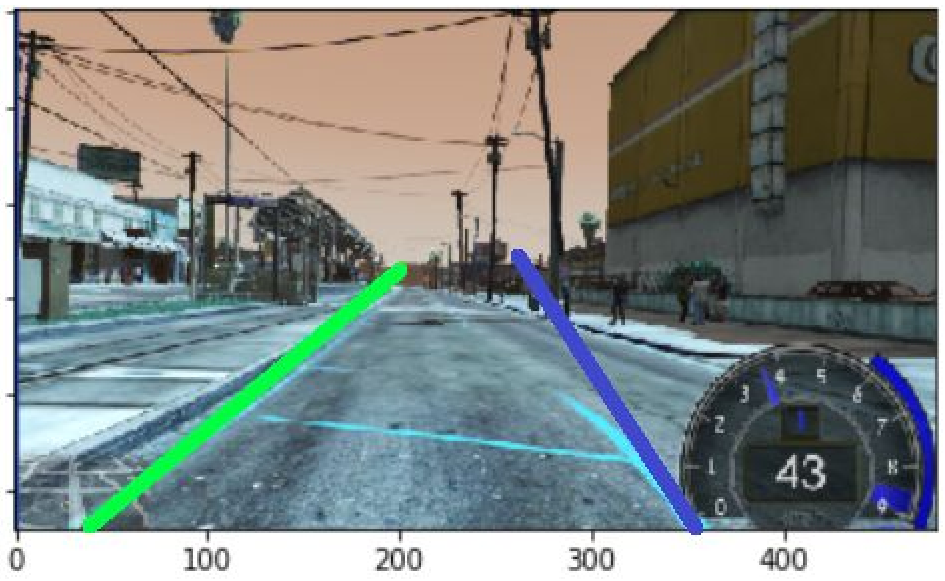


Fig 7.4: Lane prediction using a hood mounted camera angle

Even though we could roughly estimate the boundaries of the lane, the algorithm could not accurately segment lane area and struggled massively in circumstances with intersecting lines, such as at a crossing.

As you can see in the image above, the algorithm does find the boundary of the lane at some points but extends the boundary line to the horizon.

The solution is to imitate a new perspective, that provides a near bird's eye view of the road lanes ahead. This is done in traditional cars by having a wide eye lens camera installed near the bumper.

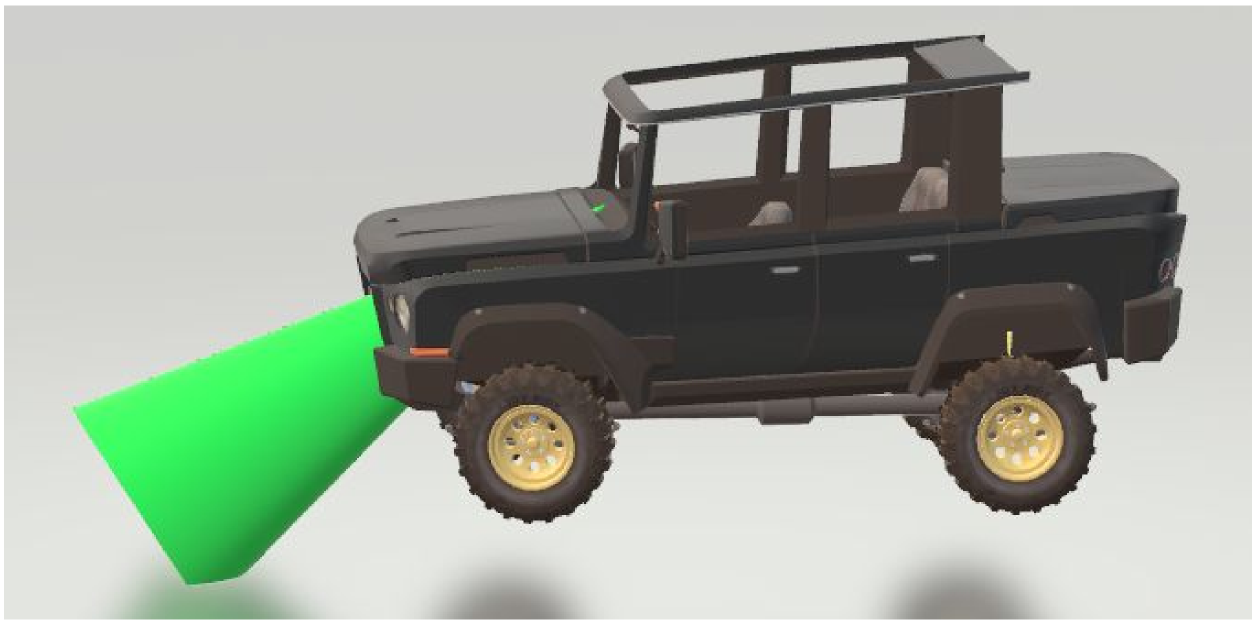


Fig 7.5: Wide angle perspective from the bumper mounted camera

We can achieve this effect by performing a perspective transform onto the frame just like in [11].

Perspective transform is defined by:

$$dst(x,y) = src(fx(x,y), fy(x,y))$$

Where fx and fy are specified by the transformation matrix. The transformation matrix is calculated using:

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Where

$$dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2, 3$$

The above equation is solved for map_matrix, which becomes our transformation matrix.

Using perspective transform, we can transform the above image to:

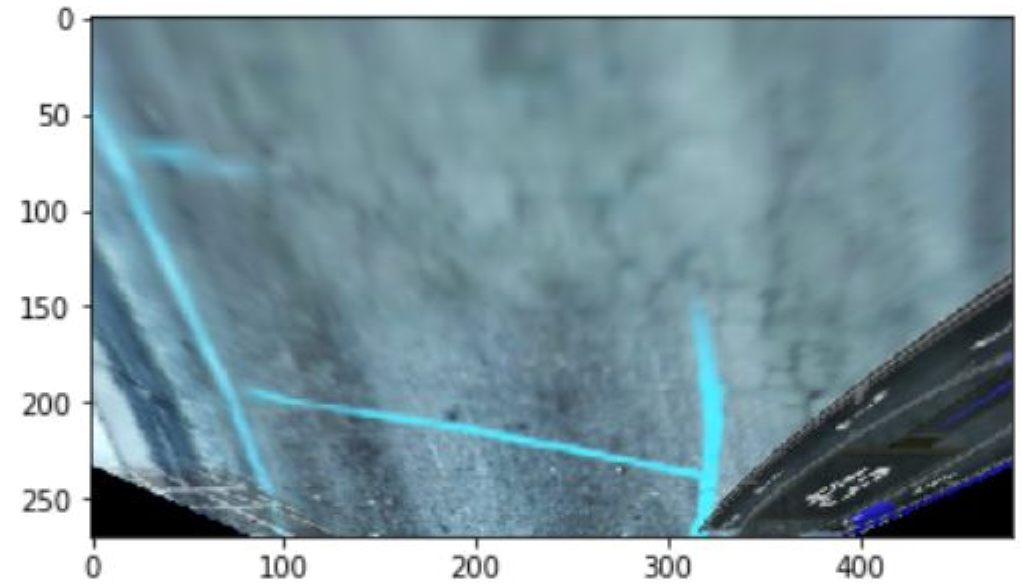


Fig 7.6: Perspective transform on Fig 7.3

Now the algorithm can more easily identify the lanes without any intersecting lines from the horizon or other features in the image.

We can use this technique to provided dedicated perspectives for side views and rear view for our system when the system is driving the car in game in 3rd person view.

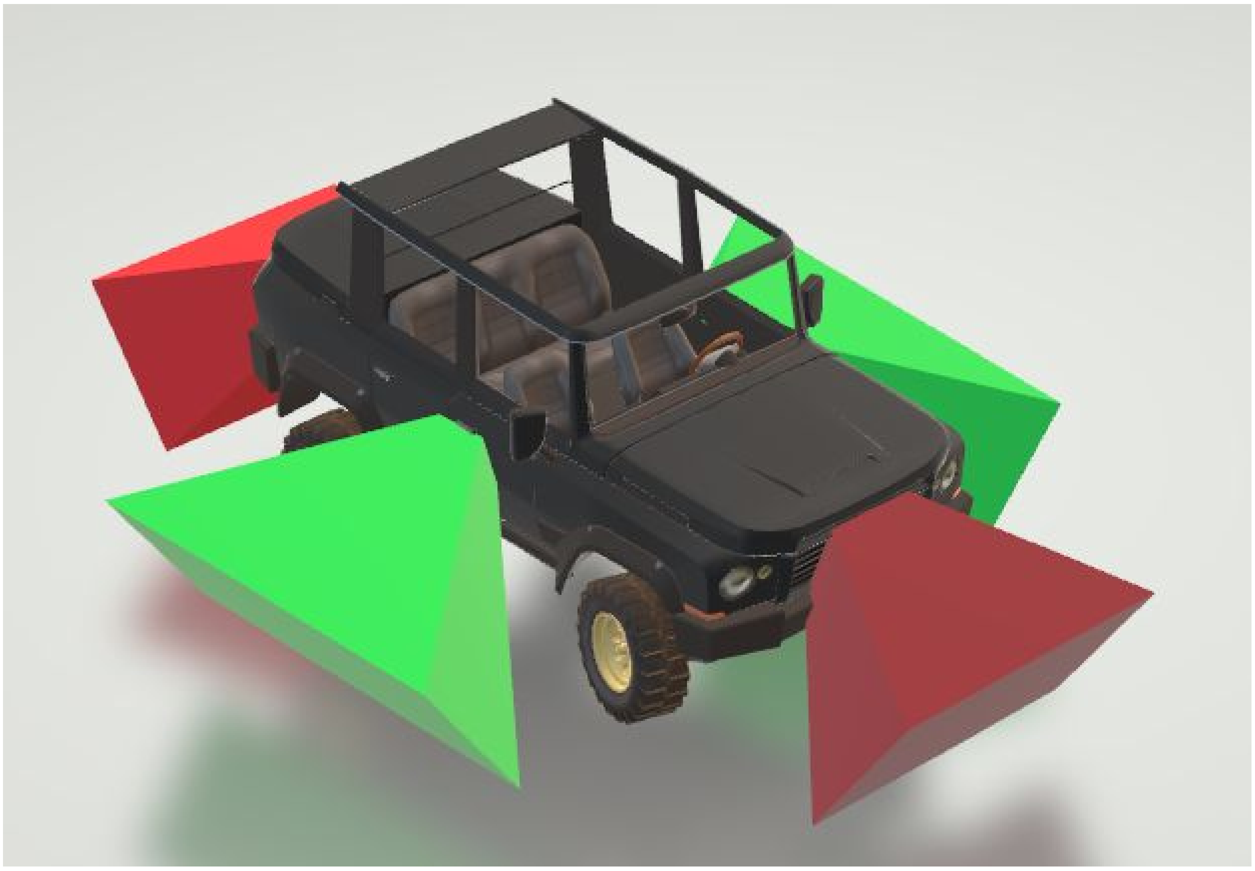


Fig 7.7: Other possible camera angles

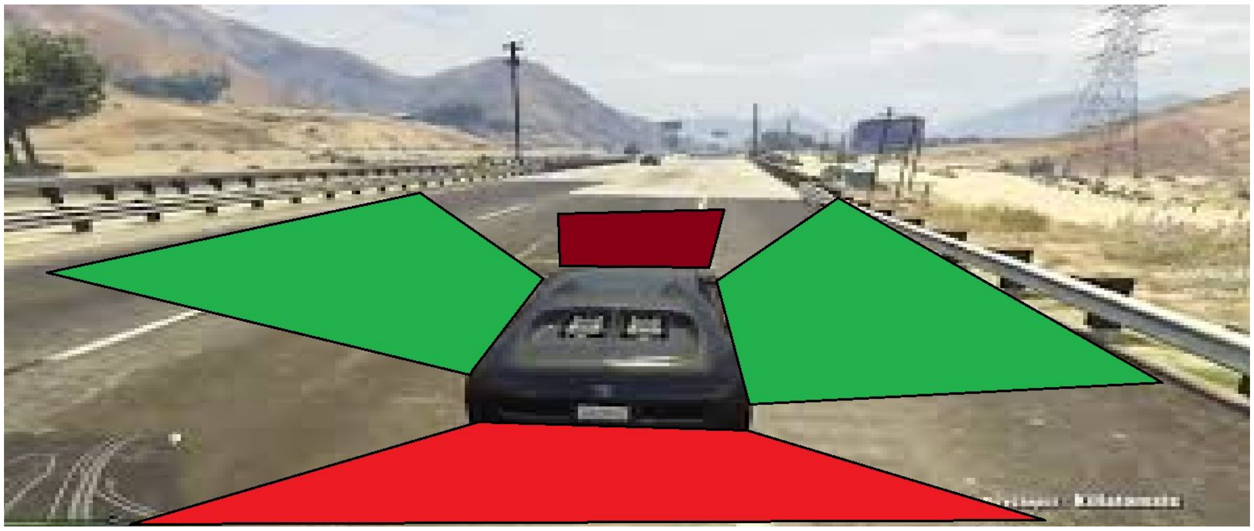


Fig 7.8: Camera angles mapped onto different sections of the road

In the image above, we postulate how a single frame from GTA V can be sectioned to approximate multiple perspectives from different angles using. perspective transform. With multiple side views (green), a rearview (red), and a front/lane view (maroon). There is a noticeable drop in resolution whenever an image area has a perspective transform performed on it. In an ideal case, we could run the game at Ultra High Resolution (4096 x 2160), but due to the hardware limitations, we can only run the game at a resolution of 1280x720, and process the frames at the resolution of 480x270. Sectioning of the image as shown above dropped the resolution too much to be usable. Hence, we only implemented the front/lane view in 1st person perspective to carry out lane detection.

The next challenge is to find out which area to perform the perspective transformation on for lane detection. A possible solution is to use a pre-trained convolutional neural networks to perform image segmentation on the transformed image (if trained on the correct dataset, this method does not need perspective transformation to work).

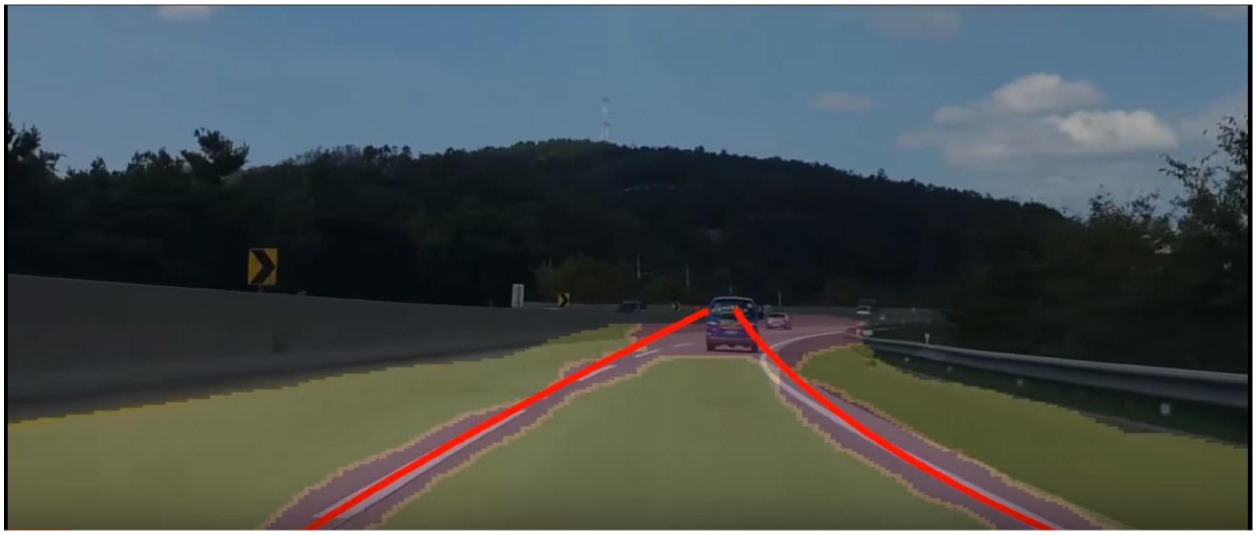


Fig 7.9: Lane segmentation using Convolutional Neural Network masking.

However, this is computationally very expensive, for a task that can be accomplished by using traditional machine vision methods [12]. We decided to run the segmentation CNN once in the city area of GTA V’s map, and averaged the Region of Interest (ROI), for which the segmented lanes are detected in each frame. ROI is defined as the region in a frame where the features that the algorithm is looking for are found in. We used this averaged ROI to create a perspective transformation matrix to transform that region to a bird's eye view perspective, and on that transformed bird's eye image we perform lane detection and mark the detected area using a polygon. An inverse perspective transform is then applied to this newly created polygon, to map it back onto the original POV, and then it is imposed onto the final image [13].

An alternative solution to the one mentioned above is to use a window search function over the whole image, and see which window provides the best lane prediction. This method does not require any use of a neural network but is more complicated, and we were not able to make this solution work reliably enough.

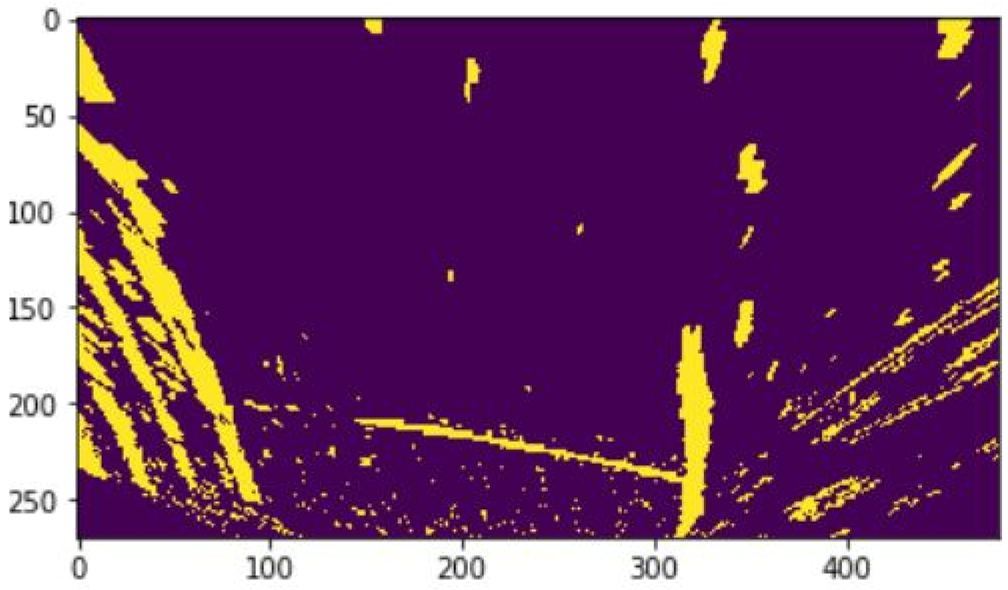


Fig 7.11: Simplified binarized version of Fig 7.3

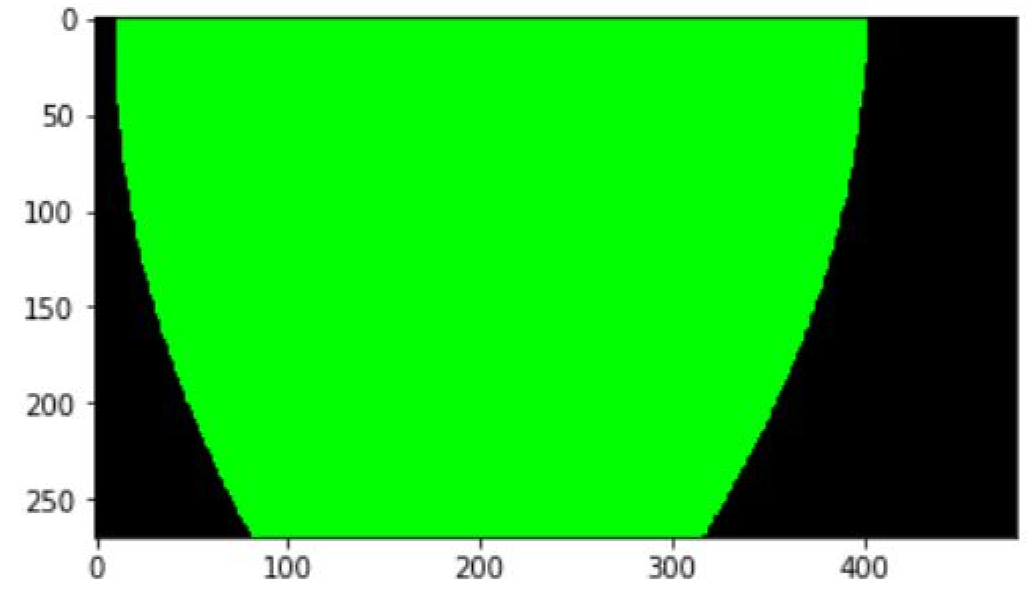


Fig 7.12: Lane prediction is done on Fig 7.11

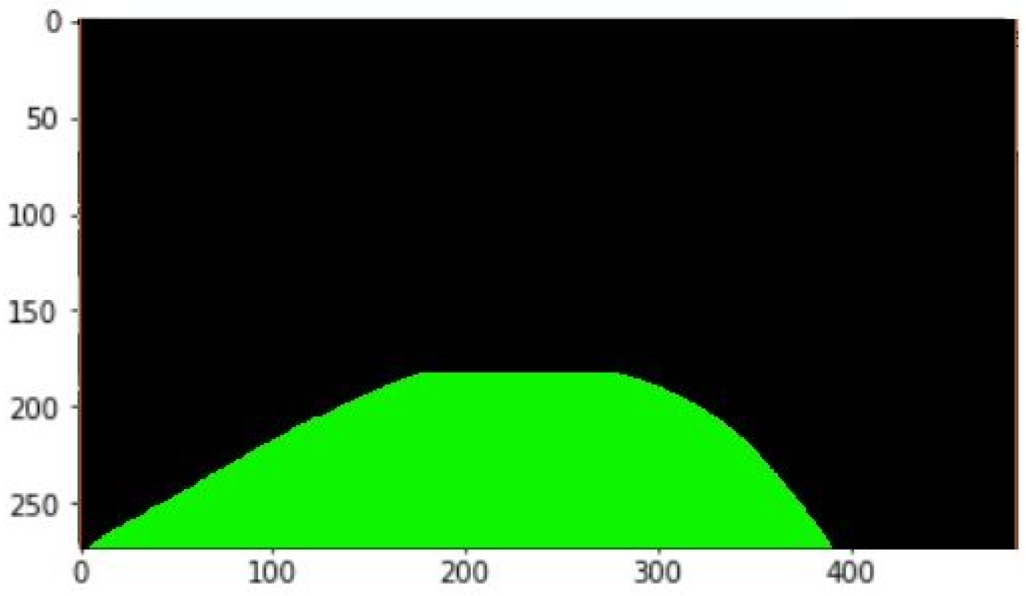


Fig 7.13: Predicted lane transformed back onto the original image

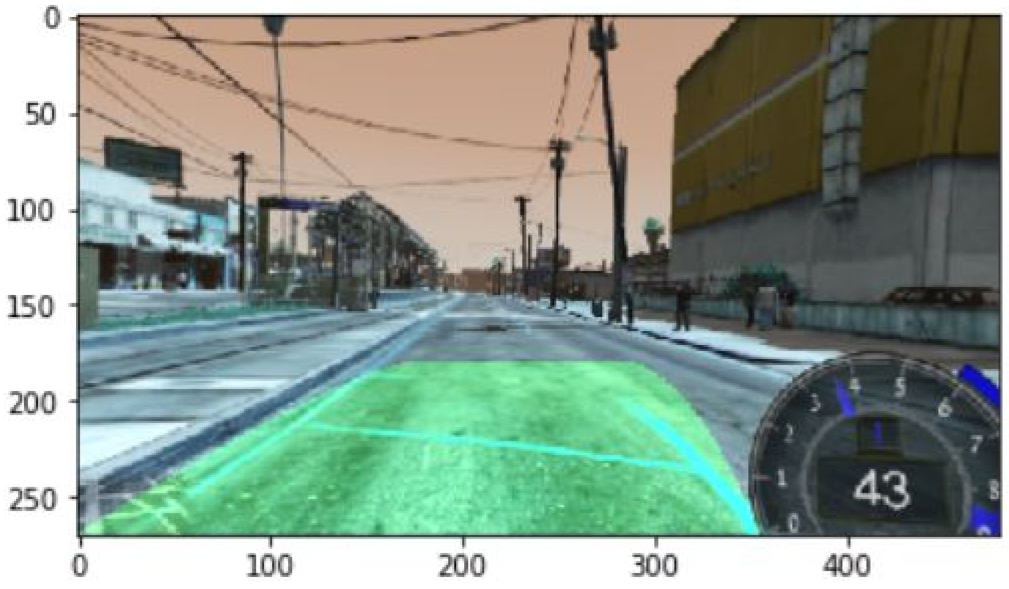


Fig 7.14: Predicted lane imposed on original image

We use other image processing technique to calculate the offset of the vehicle within the predicted polygon (this is the same as lane deviation score calculated using data from mods), and the curvature of the lane detected lane itself.

Vehicle offset is calculated as:

$$offset = fx\,(lane\ polygon) - fx(image)$$

Where fx returns the center x value of the passed in polygon.

Lane curve is calculated by taking the edges of the lane polygon that are closest and furthest away from the center of the curve and measuring the radii of each point from that center. The equation is too verbose to be explicitly written here.

Both equations require calibration to output accurate results.

We also calculate the “momentum” of the car by calculating the pixel difference between the frame at time t, and the frames at time t -1 to t – n. We can also calculate the rate of change between momentum at time t and time t-n to calculate the impulse on the car. Some preprocessing is done

Momentum

$$p = Pixels_t - \frac{1}{n}\sum_{i=1}^{n} Pixels_{t-i}$$

Impulse

$$i = \frac{d}{dt}(p(t))$$

Each frame is also processed to find any objects, classify such objects and create a bounding box around them. The model used to perform this task is TinyYoloV3 [14] trained on the CoCo dataset [15].

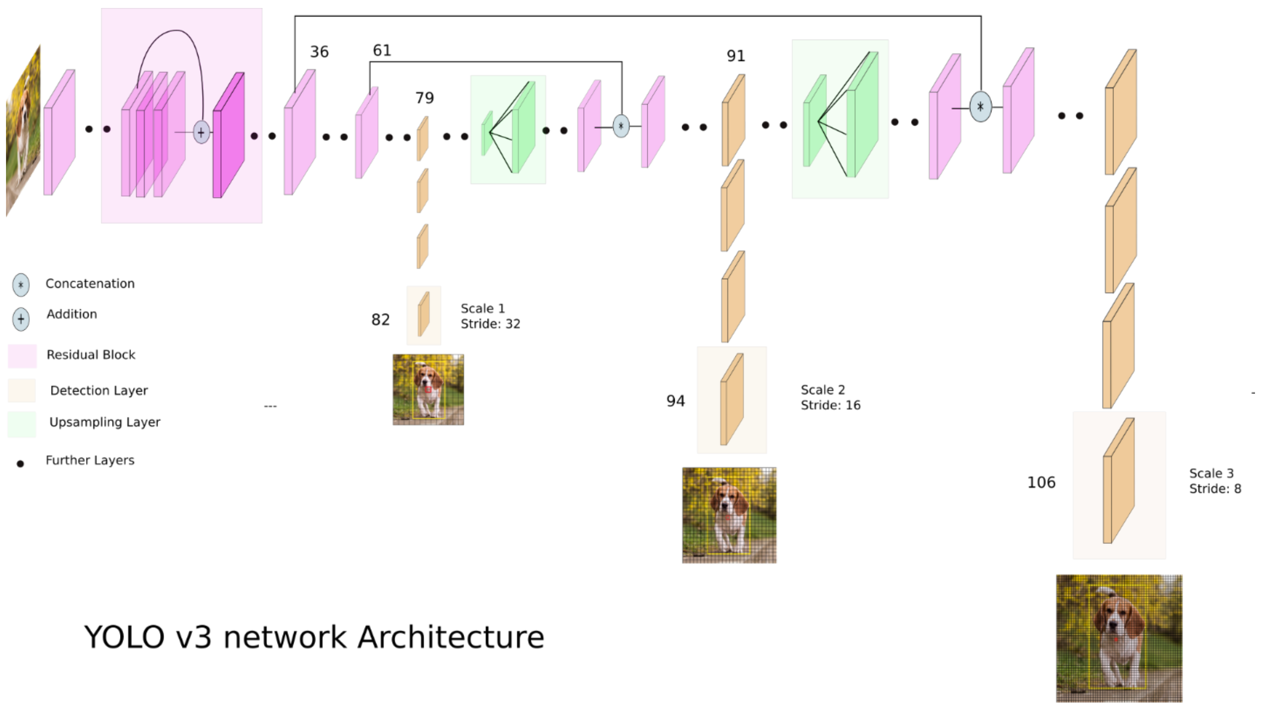


Fig 7.15: YoloV3 architecture

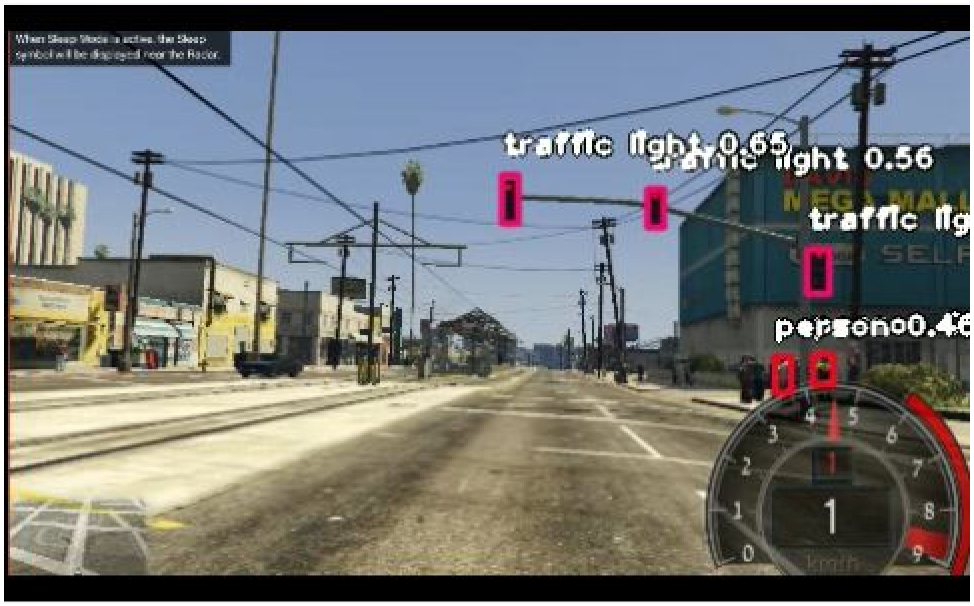


Fig 7.16: Detected objects with bounding boxes

Finally, we also want to calculate the distance between the car and the objects in front of it. GTA V mods do provide 3D bounding boxes and orientation of all objects spawned in the game.

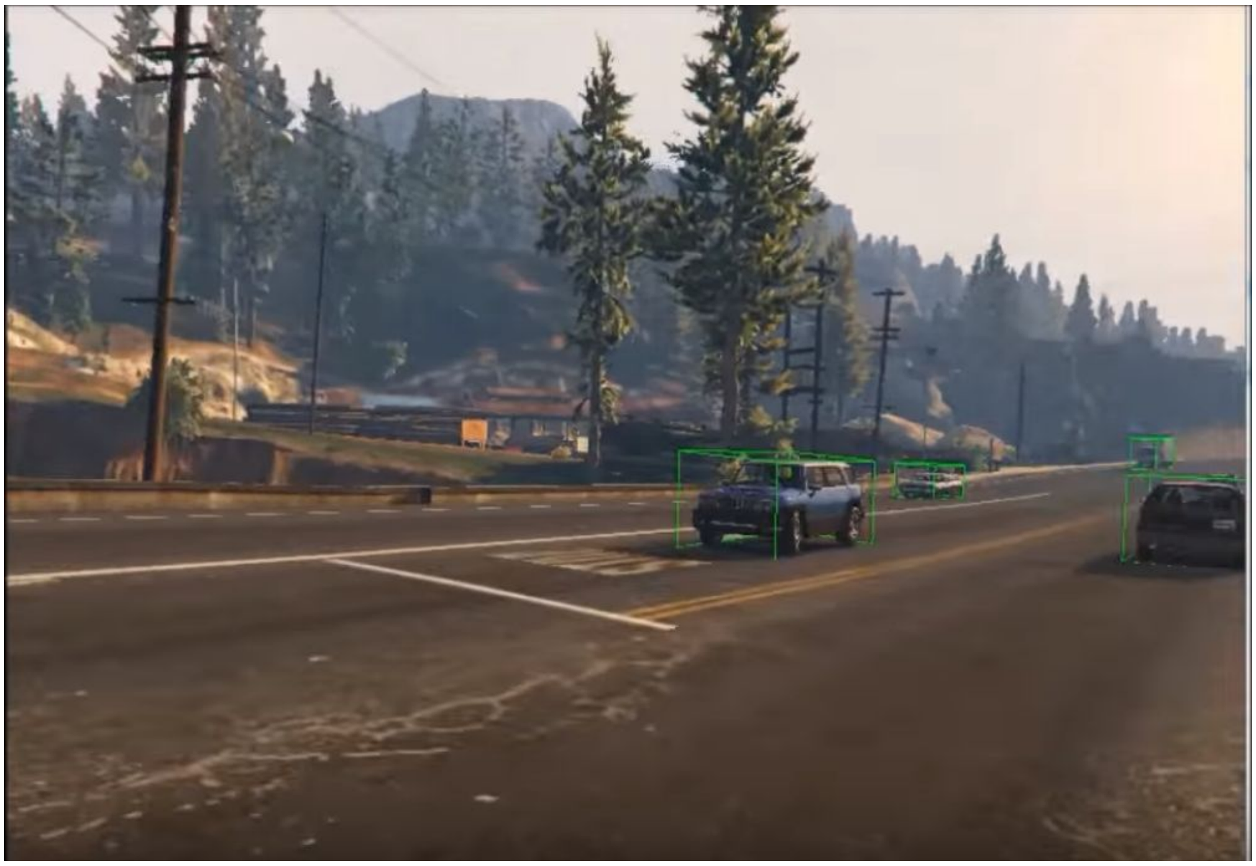


Fig 7.17: 3D bounding boxes on objects extracted using DeepGTAV mod

We can use the volume of each bounding box along with its orientation vector to calculate the distances. However, we could not use this mod for this purpose due to complications (as it will be discussed later). We developed an alternative solution of using our object detection system to create a heat map of all the objects detected and using the area in the heat map to approximate distance. This method required additional calibration. The calibration was done using the following basic relationship:

$$Distance \propto \frac{1}{Area}$$

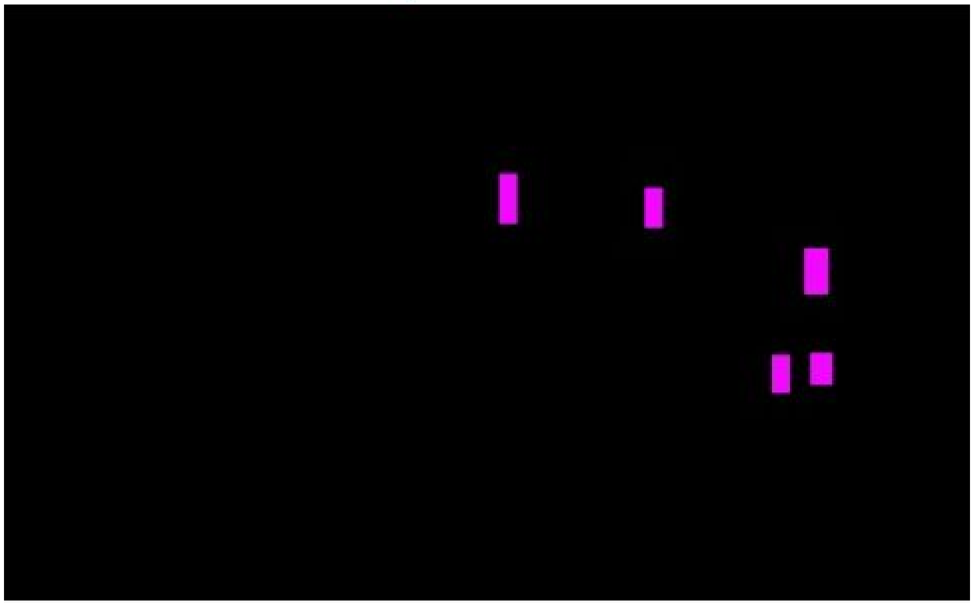


Fig 7.18: Heat map of detected objects in Fig 7.16

All these separate functions were implemented into a single pipeline, which took in the filtered data (for each unit of time) and output all calculate values and information in a single data structure (we store it in a python dictionary). The data extraction module would pass this data structure to the next module. However, due to the hardware limitations, we could not run the data extraction module on the PCs provided to us. Instead, to demonstrate this module's capabilities, we collected data for a driving session and compiled into a single file. We then iterated through each frame in the file, and passed the frame to the module, and compiled the outputs for each frame into a json file.

7.1.4 Decision-making module

This module executes 3 tasks. Building the internal map of the system, making decisions based on the information in that internal map, and finally sampling those decisions.

These 2 tasks were distributed amongst 4 submodules.

Internal Map Constructor:

As the name suggests, it uses the information passed from the previous module and uses it to build an abstracted representation of the environment of the car. Within this abstracted environment, the submodule must also provide the following features:

1. Decision space: We define this as the range of available options/decisions for the agent that will exist within the environment.
2. Rewards: We define this as relevant feedback to the agent based on its action in the environment. (Colliding with an object might return a negative reward while staying in the lane will return a positive reward).
3. Observations: The environment itself that is provided to the agent.

We modeled this submodule on the popular python reinforcement learning development library gym.openai [16].

This module was implemented using a library called pygame[17]. This library is used to create games using the Python programming language. The internal map can be seen as the game and the agent as the player. Pygame provides us with the tools to iterate through the data sent from the previous module and create the game for each unit of time of the system. It also allows us to define the agent itself and the actions available to that agent. Just as a game has a score, the system's abstracted environment has rewarder functions.

Our implementation of this submodule constructs a game in which the agent must plot a path from its current location (at the bottom of the screen out view), to the top of the screen while collecting the most amount of points, as defined by the rewarder functions. The environment itself is a top-down version of the current frame being processed by the system, and it shows the predicted lane and positions of the relevant objects in that frame from a top-down perspective.

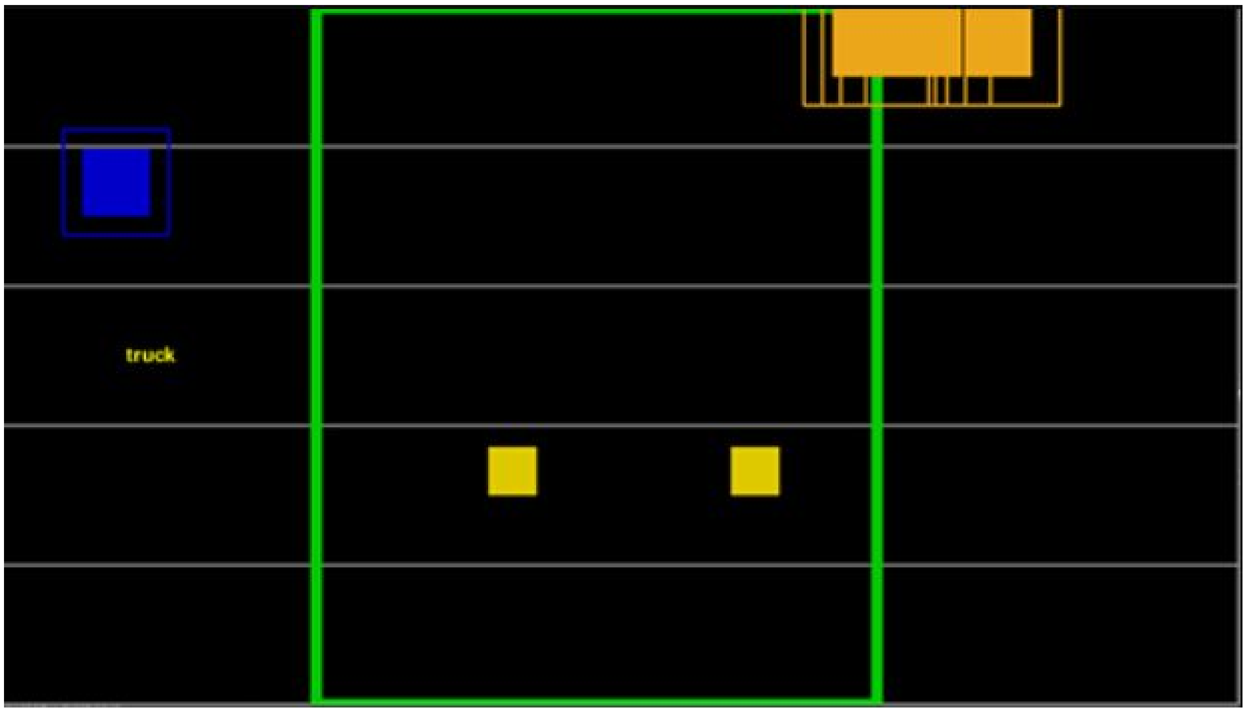


Fig 7.32: Internal map without the agent

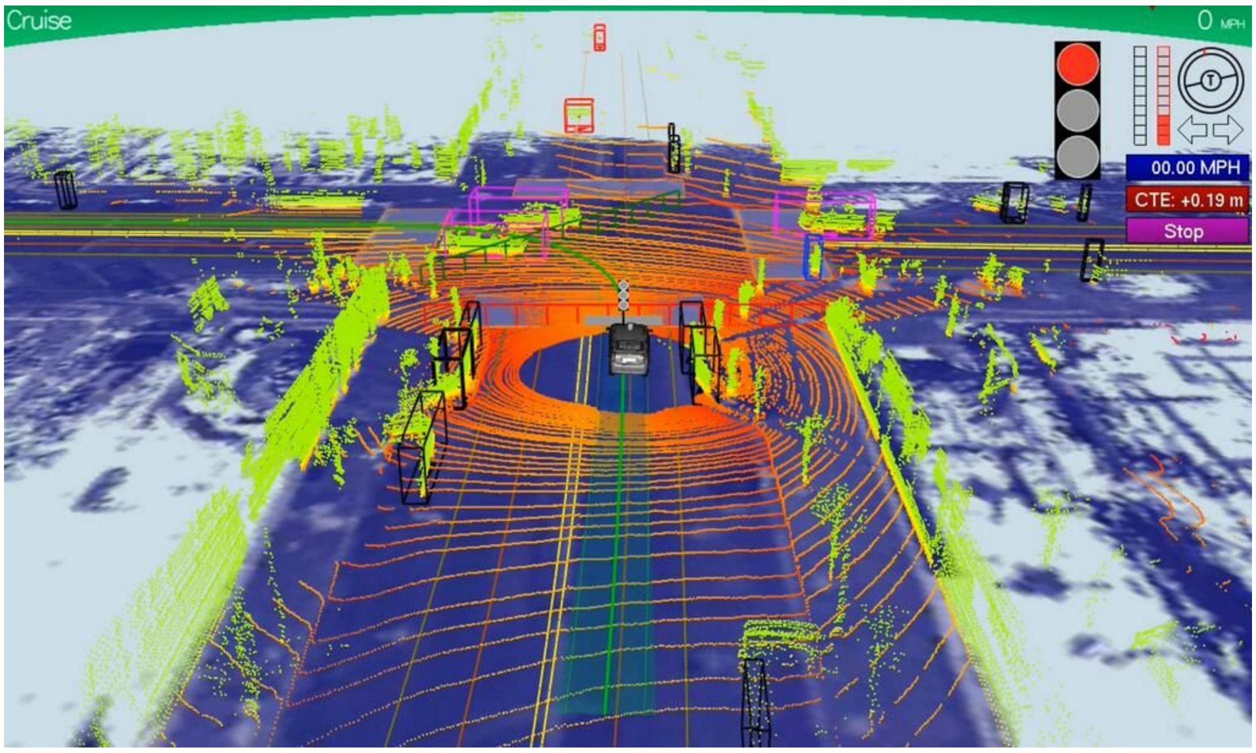


Fig 7.34: Internal map of deployed self-driving car

Objects that the car, and hence agent, can crash into (other cars and people) are represented by a square with a boundary defined around the square to indicate that the agent will get a negative reward for coming to close to said object. Traffic lights cannot be crashed into and hence do not have this boundary. In our implementation, the size of these crashable objects indicate their priority values.

Hence persons have bigger squares than cars, and crashing into a person has a larger negative reward than crashing into a car.

The rewarder values, shape and size values for all entities (an entity can be an object or the lane) are initialized with default values which can be modified by passing in python dictionary to the submodule with the following attributes:

```
object_1 = dict()
object_1['proximity_reward'] = proximity_reward
object_1['collision_reward'] = collision_reward
object_1['proximity_box'] = [x, y]
object_1['collision_box'] = [x1,y1]

map_attributes = dict()
map_attributes['object_1'] = object_1
```

Map_attributes is a dictionary of dictionaries, each dictionary in map_attributes represents an entity, and contains its relevant attributes.

This is just one possible implementation of this sub-module. The modularity of the system allows a different implementation without having to make a significant change to other parts of the system.

Decision-making agent

This sub-module communicates with the map constructor in order to make a decision. The relationship between the 2 is as follows:

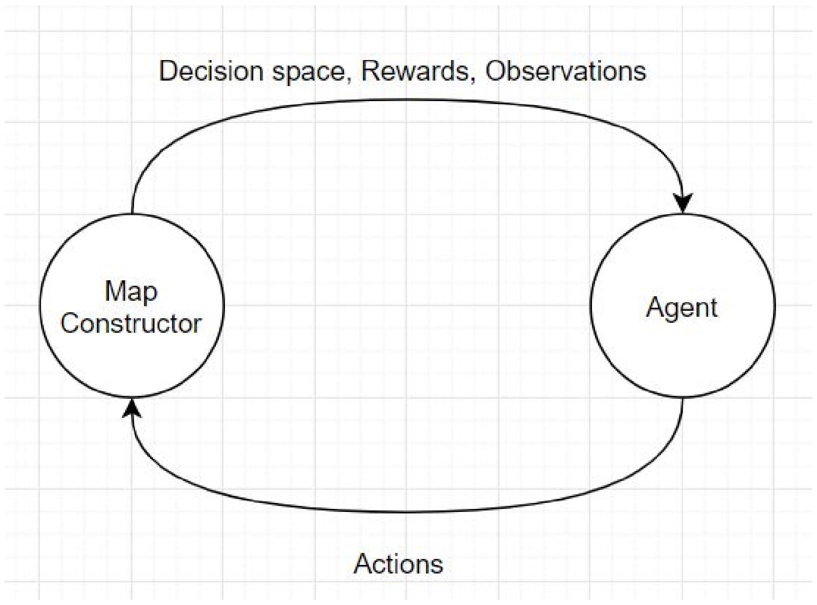


Fig 7.19: Relationship between agent and environment (Map constructor module)

The agent can be implemented in a multitude of ways. The agent can have a simple if-else implementation or a more complex reinforcement learning model. To train such a model before integration into the system, data from a driving session can be collected beforehand, and put through the data extraction module. The information from the module can be used to build the internal map by the map constructor. The agent can now train in this isolated internal map, and after it is done training, it can be deployed into the system. This is yet another benefit of the system’s modularity.

The agent is restricted by the decision space defined by the map constructor, hence it’s development is dependent on the design of the map constructor as well. We did not implement a trained agent into our final demonstration as we poured most of our time resources into building the map constructor. Learning and non-learning agents are   areas of AI, and deploying an agent should be easy given that we have already defined the environment that the agent will operate in.

The agent should also be restricted to a maximum of m moves for every unit of time in the map constructor. This is so the agent prioritizes efficiency of the path as well.

Multiple instances of the agent can be used to do separate tasks. One agent can be specified to plot a path, while another can be specified to keep the car within the lane. Our implementation for the agent for the demonstration is an agent which must keep itself within the boundaries of the predicted lane.

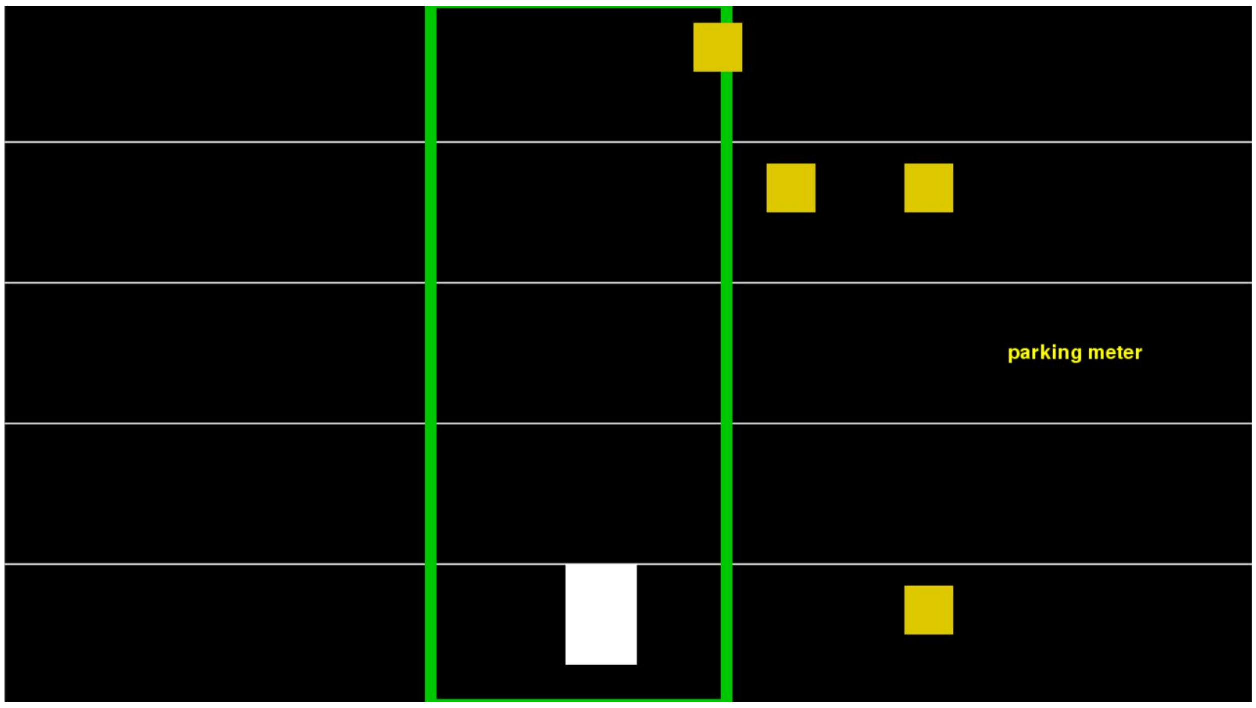


Fig 7.33: Internal map with agent (white box)

[https://robohub.org/how-do-self-driving-cars-work/]

Decision sampler

The role of the agent is to plot the best possible path for the car given the decision space provided by the map constructor. However, this path needs to be converted to a decision which the next module can interpret. We designed the output to be a flag array which can be sent to the navigator.

Mode of operation	Normal	Brake
Flag	0	1

In the above example, the decision output can be either be "normal", i.e the navigator does not have to alter its behavior, or "brake", i.e, the navigator must brake.

N decisions can be sampled from an m set of moves by the agents, and the average of the first n moves can be taken. If that average is above or below some threshold level, then the flag array can be altered from normal to brake.

For example: If the decision space for the agent is defined as: {forward, left, right}, then the absence of a forward movement can be interpreted as the agent's decision to remain still, and hence to brake. Since we are only concerned with the brake flag in this example, we can assign a +1 value to the forward movement in the y-axis (f). The formula for the brake flag (y) in the decision array will then be as follows:

$$x = \frac{1}{n}\sum_{i=1}^{n} f_i$$

$$y = 0,\ x \geq T,\ y = 1,\ x < T$$

Similar functions can be modeled for more flags to alter the behavior of the navigator function more comprehensively.

Mode of operation	Normal	Brake	Right	Left	Accelerate	Emergency
Flag	0	1	1	0	0	0

A logic can then be implemented upon the base function to allow the activation of multiple flags using the output from multiple agents at the same time. However, it is to be noted that the neural network model in the navigator must be trained accordingly to be able to understand and interpret the input from these flags.

A more complex model may even allow linear activation between 0 and 1, rather than a binary one. A neural network should be able to compensate for linear activations in the flag array without any additional modification.

It should be noted that although we have tested how a flag array will alter the behavior of the navigator module, the testing was done in isolation in an ideal scenario. We did not have the processing power available to do extensive functional testing on this feature, and we suspect that deploying a complex flag array might require further calibration to work as intended.

Knowledge Base

The knowledge base contains user-defined structured data regarding the possible states of the environment of the car. In simple terms, it is a database which contains information about the objects and scenarios the car might interact within its environments.

Our intended implementation is a hierarchical tree which we call a semantic tree. Every object belongs to a class and is placed under hierarchical categories. This implementation helps from a debugging and interpretability standpoint as it allows us to see visually how each object is categorized and where new objects should be placed. An object's place in the graph dictates it's priority level with the car. Any object in the 'Living" branch will be given greater priority than an object in the 'Non-Living' branch.

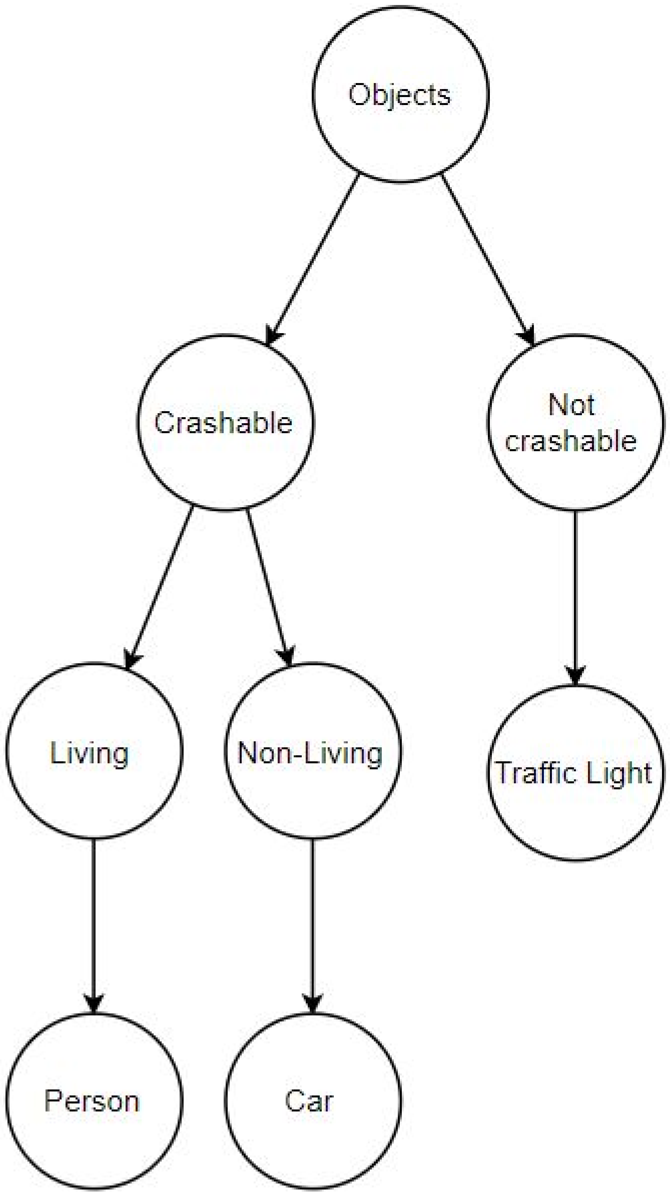


Fig 7.20: Sample semantics tree

Each leaf node in the semantics tree is an object the car can interact within its environment and contains the default values for itself for the map constructor sub-module.

This module is not critical, as such data can be hard-coded into the sub modules mentioned above. However, implementing this sub-module makes alteration and debugging much easier.

7.1.5 Control module

The control module takes in the output from the navigator and converts that output into a signal that can be understood by the actuators. In GTA V, we had two possible methods of sending control signals to operate the controls of the car. The first option was to communicate with a mod in GTA V via a server-client protocol (much like that described in section 7.1.3). The message contains the throttle position of the car, which is a value between -1 (reverse) and 1 (forward), and the steering angle of the car, which is a value between -1 (left) and +1 (right). We could not use this method because of issues that will be discussed later on.

The other method was to simply emulate button presses on the keyboard. 'W' for forward, 'A' for left, 'S' for brake/reverse, 'D' for right. A combination of those presses would cause the car to go in a new direction as well. Pressing 'W' and 'A' would cause the car to go forward left. However, this method results in a very jerky operation of the car. The button presses are binary, 'W' indicates a throttle position of +1, 'S' indicates a throttle position of '-1'. The same goes for the steering angles and controls. There is no half throttle activation or half steering angle input.

The solution to this was to emulate an analog output which can provide linear activation between the 2 extreme values for throttle position and steering angle. We did this by using a community built python library [18].

This makes training the navigator module easier, as we do not have to record inputs for key combinations (like 'W' + 'A') when recording training data. Instead, we can limit the possible outputs from the navigator function to 4 basic commands: Forward, Reverse/Break, Left, Right. The navigator module outputs an array with the activation level of all these commands. The analog output operates over 2 axes, x and y, and the value for each is given by:

$$y = (forward - reverse)$$

$$x = (right - left)$$

A positive y value indicates a forward movement and vice versa. A positive x value indicates the car turning to the right and vice versa.

7.1.5 Navigator

The navigator module is responsible for establishing a baseline autonomy. Any decision made by the previous modules must go through the navigator module and alter the behavior of the navigator module to affect the movement of the car.

When planning out the milestones for this semester for our project, we decided that the navigator module would be given a lower priority for the following reasons:

1. Lack of processing power available to us to implement a complex neural network model
2. The TfLearn API used to develop the initial models, some of its features had become deprecated and hence there was a need to shift the development to keras.
3. The major work done on self-driving cars in GTA V is using neural networks to make a predictive model to imitate the behavior of a driver (as mentioned in the related work section). Hence, there already exists a wealth of resources like tutorials on how to build this particular module.
4. Lack of processing power available also meant that we could not fully integrate this module with the previous modules.

For these reasons, manpower and attention were given to other areas of the project that we deemed more important.

At the end of the last semester, we had developed a model that could somewhat drive in a convincing manner in the highway region. The model is similar to the model described in the image below, but was implemented in TfLearn, and designed to work with the control module as described in the control module sub-section. This meant that we could simplify the outputs from the model to the following:

1. Forward
2. Brake/Reverse
3. Left
4. Right

Additionally, since we could interpret these outputs as linear activations between 0 and 1, rather than single-label multi-class classification, we could use a sigmoid function rather than a softmax function for the final output layer.

This is a meaningful change as softmax is used for classification rather than regression. This is because the softmax function distributes the activation probability across all classes, treating all outputs as dependent events. This has the following effect:

$$\sum_{i=1}^{n} f(x_i) = 1$$

Where n is the total number of outputs from the model, and f(x) is the activation at each output node x. Their sum has to always be 1, hence the model always has to choose one output over another. This results in very unsmooth driving experience.

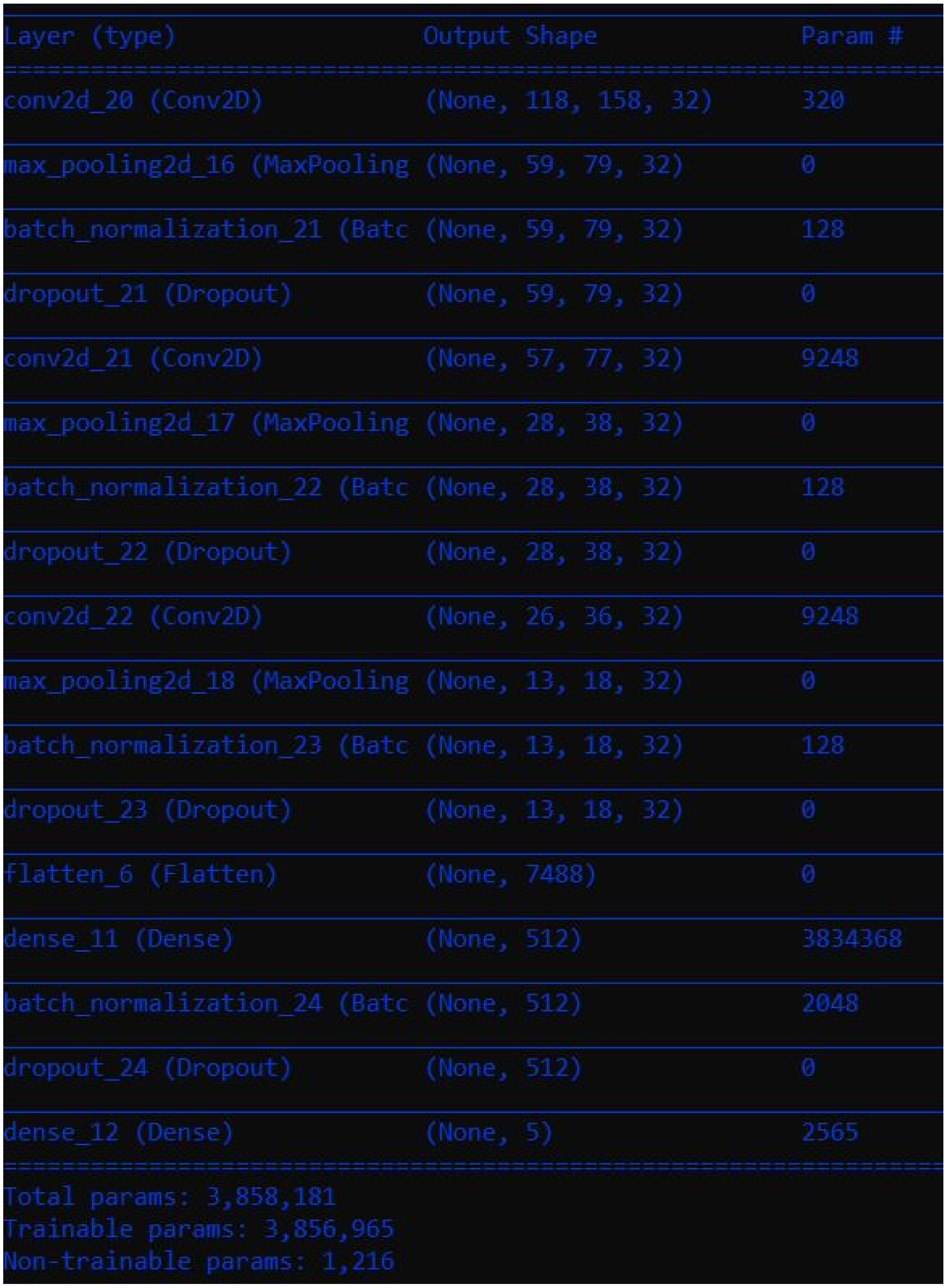
Sigmoid treats each out as an independent event.

$$\sum_{i=1}^{n} f(x_i) > 0$$

The sum of all activations at each node as an independent event. This means that the model can output that turning left with the same confidence as moving forward, and the net effect (along with the implementation of the control module) is that the system moves forward left. This should result in a smoother driving experience. However, our model was still very jerky and unsmooth, we think this is a limitation of the design of the model. Essentially, we are changing the model to do regression, rather than to do classification.

We also were using modifying makers to alter the behavior of the model. Modifier markers were features imposed on an additional dimension of the image, and passed through a convolutional neural network, alongside the convolutional neural network which takes in screen frames from the game. We replaced the modifier markers with flag arrays and the secondary convolutional neural network with a multi-layer perceptron. This change simplifies training and data collection while still maintaining the ability to modify the behavior of the model.

The design of the model is limited by the hardware available to us. The input image cannot be larger than 160x120 (black and white), as then the model would require too much memory to run (we were only limited to 2 Gb of ram in our development PCs). The design itself is inspired by AlexNet [19]. . The reason for this architecture is detailed in the previous semester's report.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 118, 158, 32)      320

max_pooling2d_16 (MaxPooling (None, 59, 79, 32)        0

batch_normalization_21 (Batc (None, 59, 79, 32)        128

dropout_21 (Dropout)         (None, 59, 79, 32)        0

conv2d_21 (Conv2D)           (None, 57, 77, 32)        9248

max_pooling2d_17 (MaxPooling (None, 28, 38, 32)        0

batch_normalization_22 (Batc (None, 28, 38, 32)        128

dropout_22 (Dropout)         (None, 28, 38, 32)        0

conv2d_22 (Conv2D)           (None, 26, 36, 32)        9248

max_pooling2d_18 (MaxPooling (None, 13, 18, 32)        0

batch_normalization_23 (Batc (None, 13, 18, 32)        128

dropout_23 (Dropout)         (None, 13, 18, 32)        0

flatten_6 (Flatten)          (None, 7488)              0

dense_11 (Dense)             (None, 512)               3834368

batch_normalization_24 (Batc (None, 512)               2048

dropout_24 (Dropout)         (None, 512)               0

dense_12 (Dense)             (None, 5)                 2565
=================================================================
Total params: 3,858,181
Trainable params: 3,856,965
Non-trainable params: 1,216
```

Fig 7.21: Summary of the newly implemented model in the navigator module

The total number of parameters in our network is 3,858,181, compare this to a number of parameters in AlexNet: 62,378,344, and the number of parameters in Google’s inception v3: ~23,000,000.

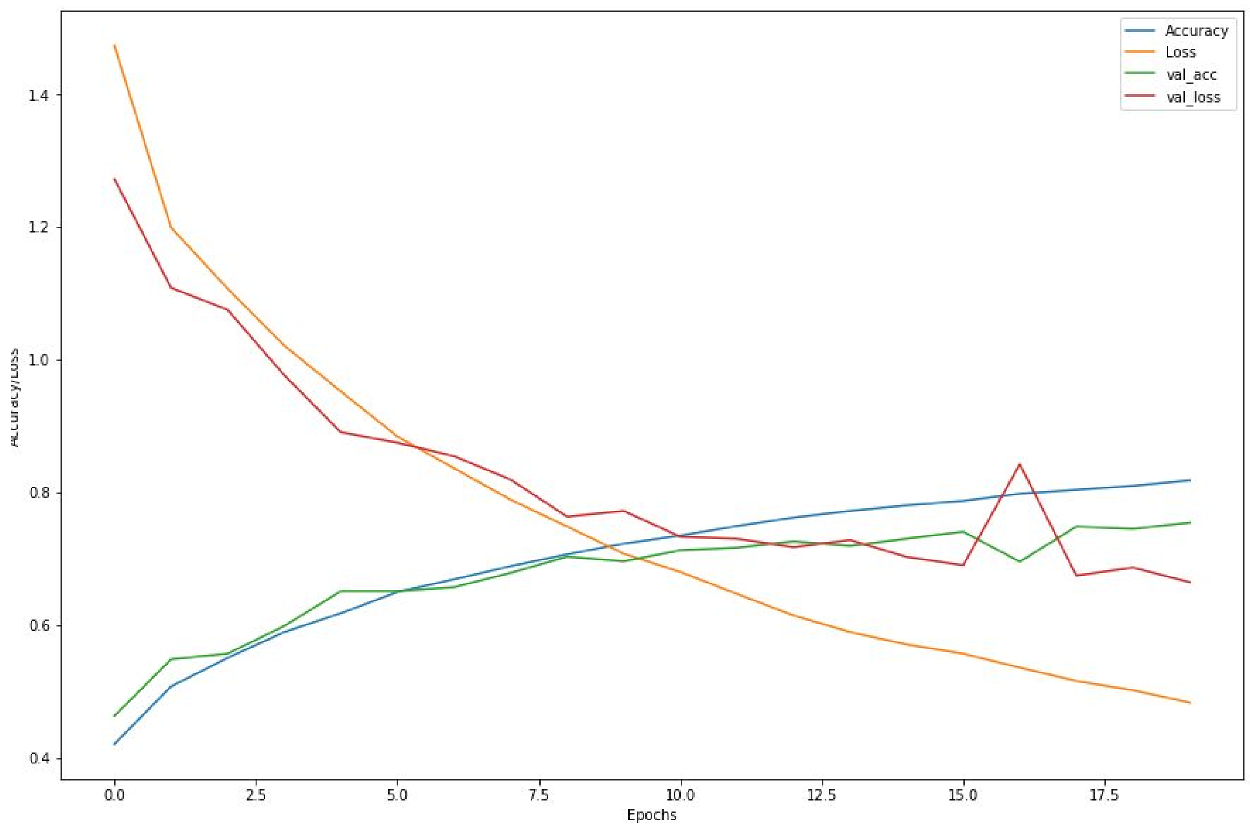


Fig 7.35: Model training

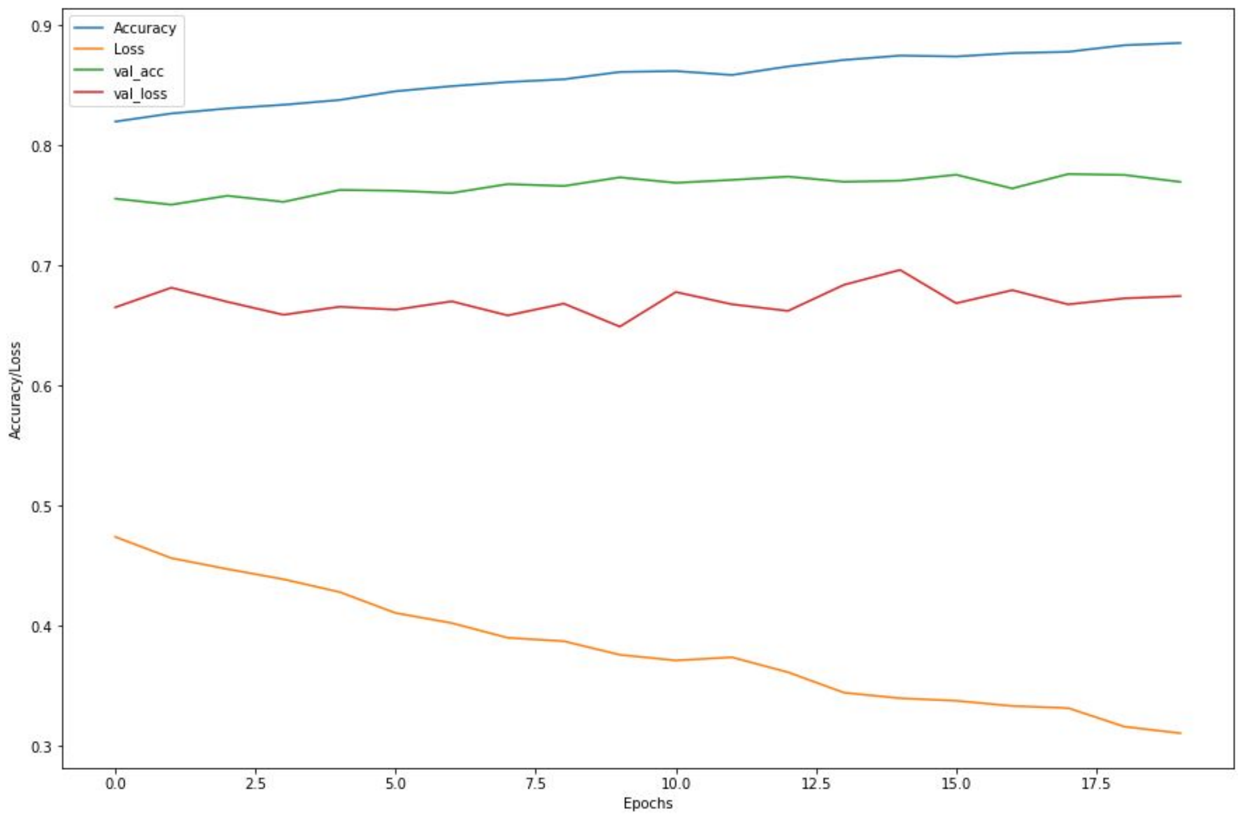


Fig 7.36: Further training with hyperparameter turning (reveal overfitting)

A sample output from the Navigator:

Forward	Brake/Reverse	Left	Right
0.4	0.1	0.7	0.2

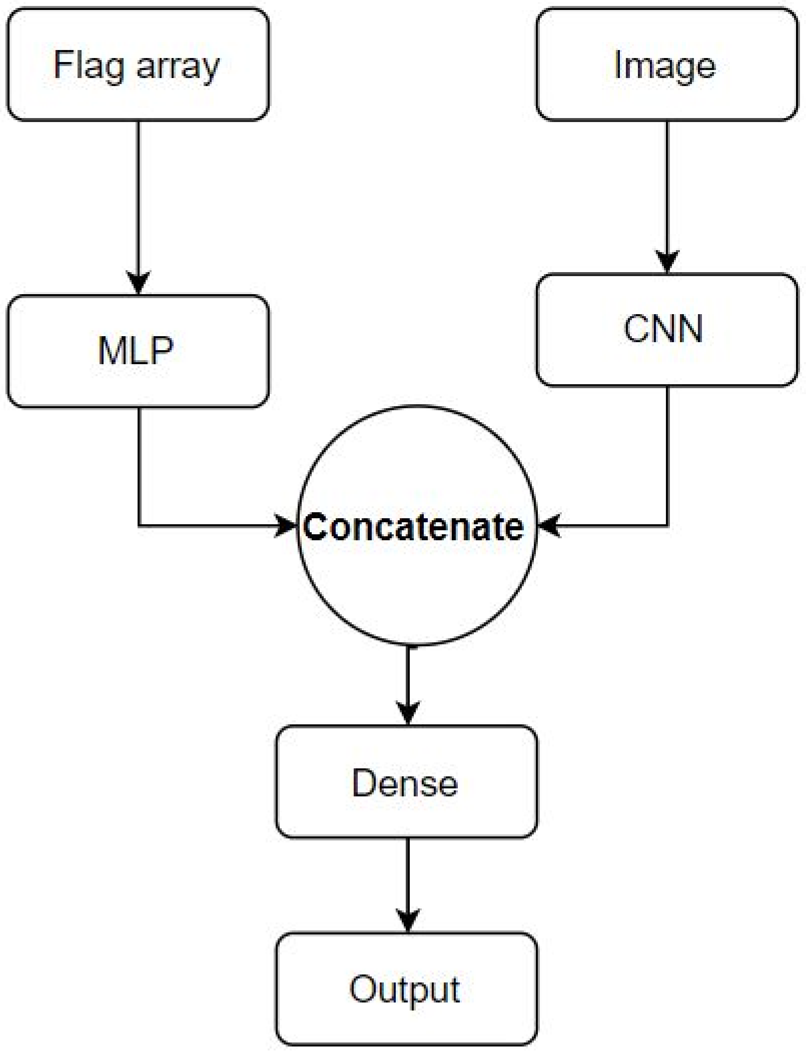


Fig 7.22: Idealized High-level architecture of model in the navigator module

The above diagram shows the idealized high-level architecture of our module. The CNN and MLP can be as simple or as complex as desired (in our case the CNN had to have a simple design). This module's output does not rely in on previous inputs, it has no memory and the output is only dependent on the image passed at the current time.

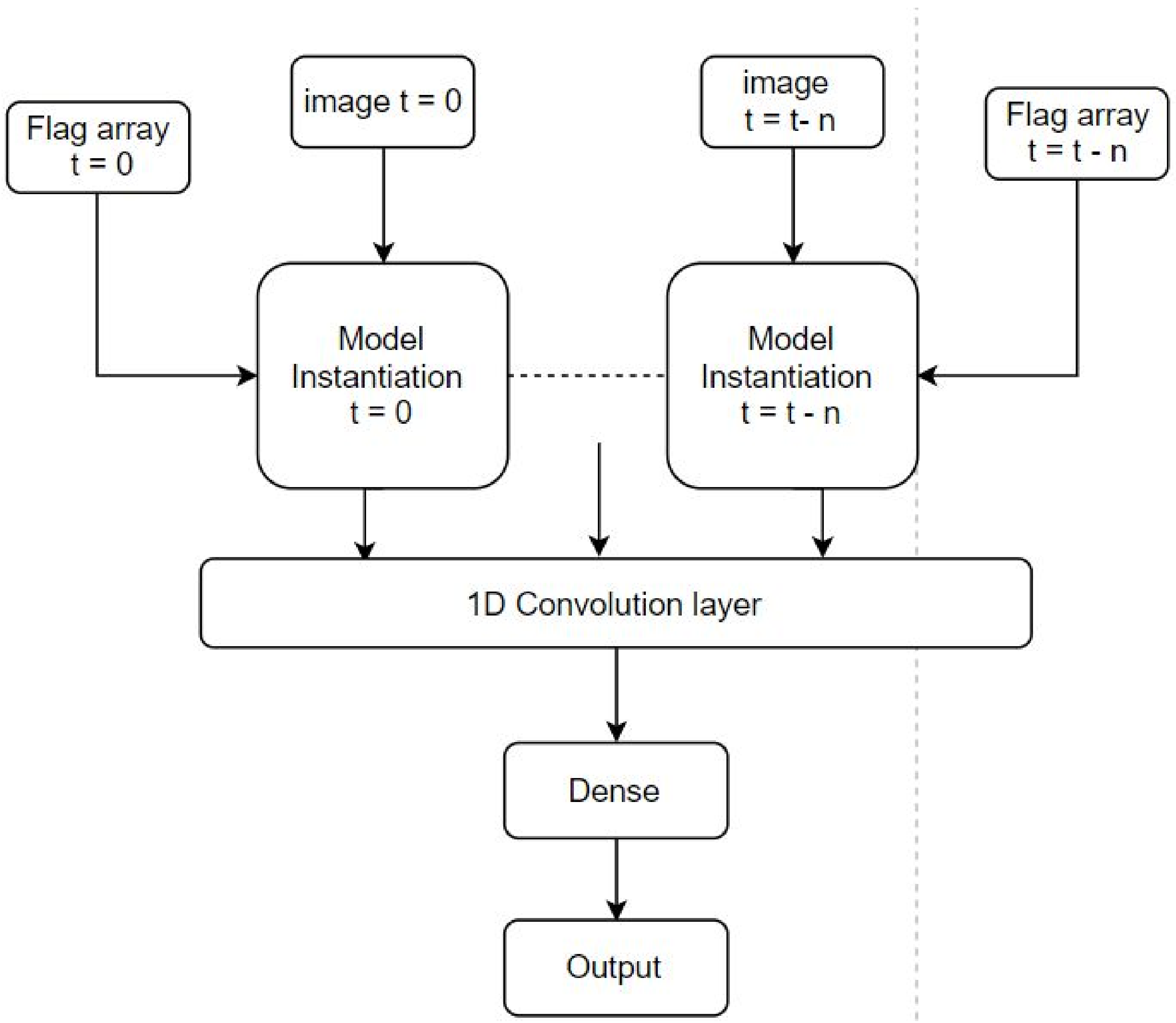


Fig 7.23: Idealized High-level architecture of the intended model

To solve this issue, the above design was suggested, as it uses a 1D Convolutional layer along with the parameters from the original model, instantiated n times for the previous n frames, to produce a result that accounts for the previous n frames as well. However, a theoretical flaw for this design is that the convolutional layer can only extract features from the parameters passed to it, it does not account for the order in which these frames arrived in.

It is to be noted, we did attempt to implement this design in TfLearn, which was unsuccessful due to some deprecated features in the API. This prompted us to shift our development onto keras, but the convolutional layer along with multiple instantiations could not be handled by the development PCs. We paused development on this module in favor of other modules of the system.

A paper from Princeton (mentioned in the related work section), solved this by appending a CNN with an LSTM model, which are time sensitive (i.e, account for the order of the inputs as well).

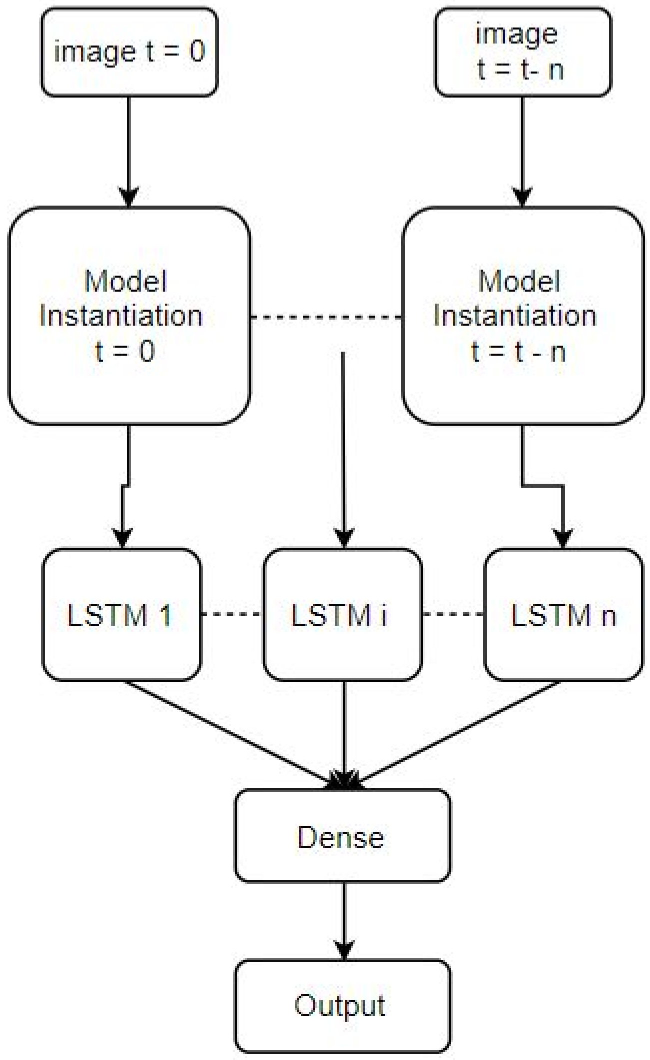


Fig 7.23: High-level architecture of model used in research paper

We would modify this design by adding the ability to interpret the flag arrays at each time t as well.

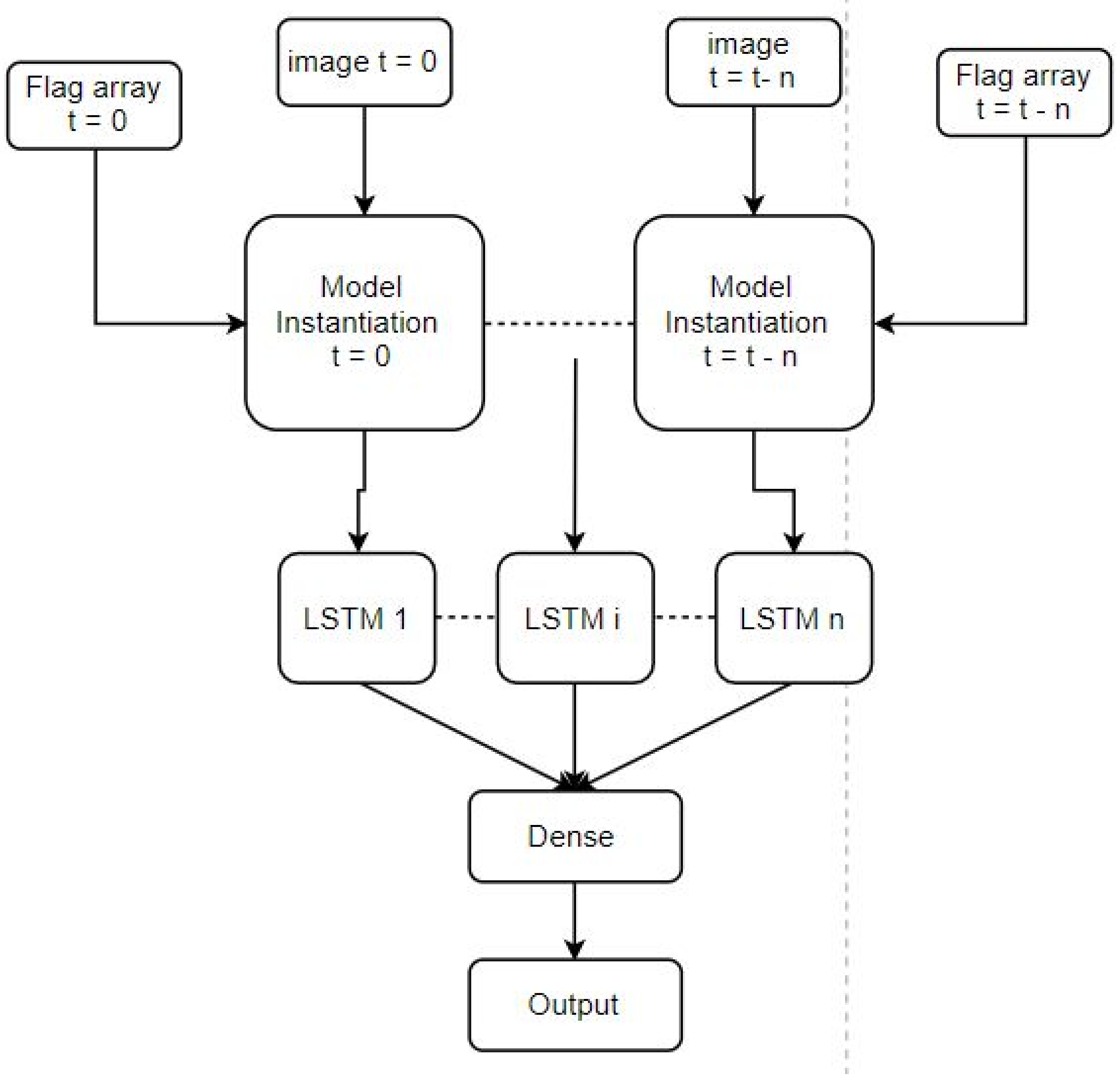


Fig 7.24: Suggested modifications to model in Fig 7.23 for implementation in the proposed system

7.1.7 System buffers

System buffers are used in multiple areas in the system to allow the system to have a sense of 'memory'. These n-buffers are implemented as python lists. System buffers are used to store:

1. The data collected from screen capture and data collection model
2. The transformed data from the data extraction module

3. Screen capture from the screen capture module and flag array from the decision-making module

We did not implement any buffers in our system, as we could not integrate all the modules into one system.

7.1.8 Human input

Obviously, there are limitations to our design. One of them is that the system may be optimized to run in 1 set of conditions (such as sunny midday, with low traffic, city driving), but the developer might choose to test the system in multiple conditions. For this, in our idealized system, a developer would train/implement the modules to run in different modes for different conditions. When the developer chooses, he/she can then send a command to those modules to use the implementation of the modules for the specific conditions desired by the developer. An example of this would be training the navigator module to run in different conditions. A more complex example would be changing the object attributes of the objects in the map constructor module, to test the decision-making agent.

7.2 Execution environment

A high priority for this semester was to figure out a workaround for the lack of processing power available to us. The first workaround proposed was to do a distributed implementation of the learning models employed in the system [20]. However, after doing some research on possible implementations, we deemed this solution to be infeasible. Firstly, there are no extensive resources available to provide guidelines on how to parallelize the workload over a network. Secondly, the biggest bottleneck performance wise is the size of the GPU RAM available (2 GB). In a parallelized network, the RAM available is not cumulative across all GPU nodes, the effective RAM available would still be 2 GB, and the improvement would be seen in raw processing power only.

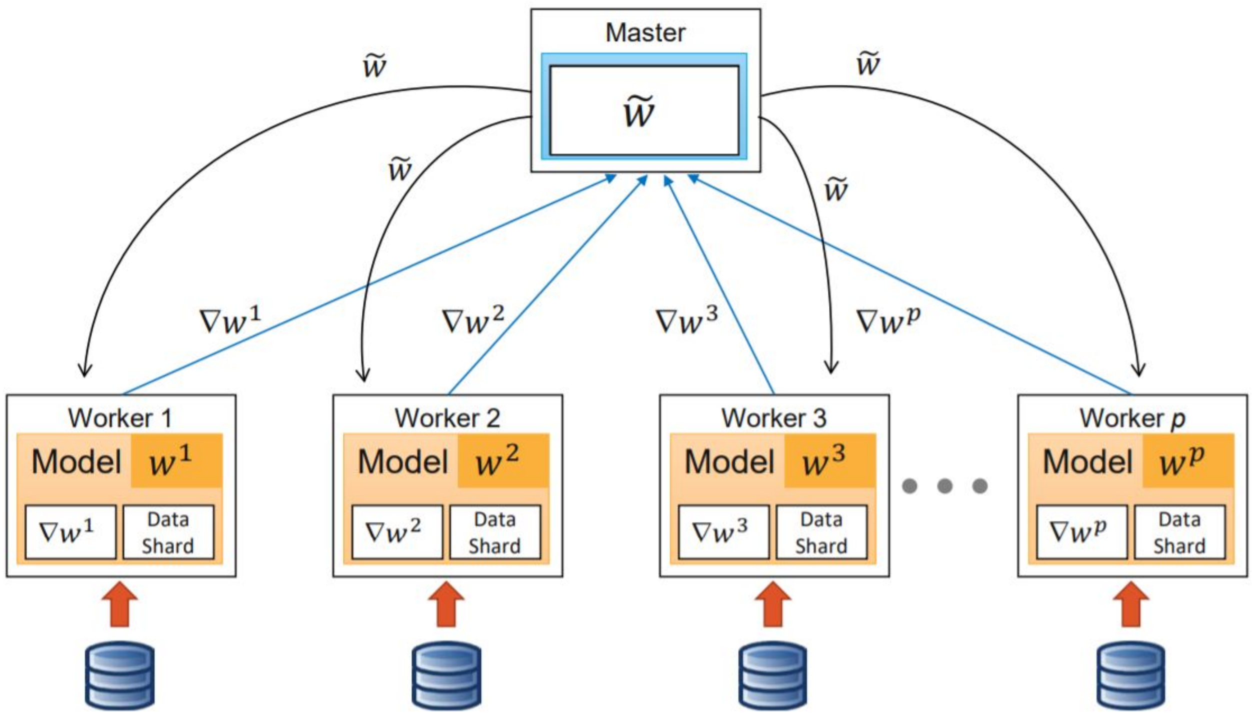


Fig 7.25: Parallelization of models over a LAN

The other approach was to distribute the modules in the system over a network. Each computing node in a network could run one or multiple modules, and they could communicate through a server-client model.

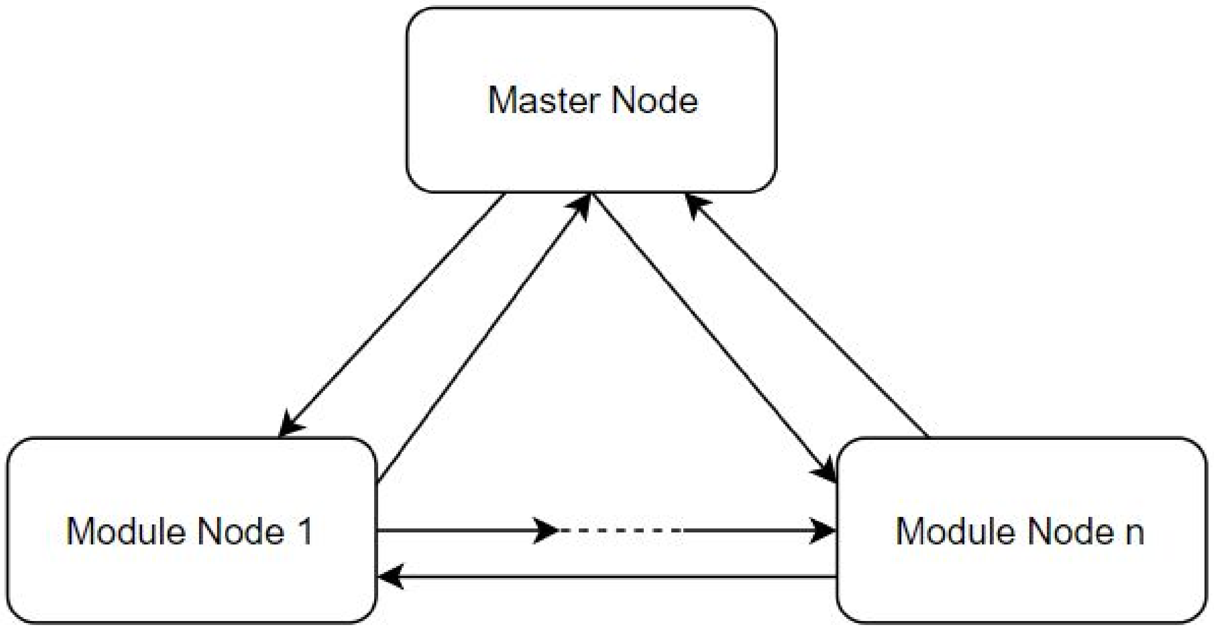


Fig 7.26: Distributed architecture of the system

This method of distributing the system works by having a master node which runs GTA V itself. It sends the initial frame and collected data to the relevant modules in the proper sequence. Each module is hosted on a separate computing node. The output from each module is passed to the next in accordance with the system architecture. The master node stays in contact with each module node to ensure proper synchronization across the whole system. This is done by assigning a sequence number to each module's output, and a module node sends that sequence number to the master node, which checks it against a predetermined sequence table. If the sequence is not in order, the current data frame is discarded, this can be changed depending on the protocol employed by the developer. The data passed between the nodes does not have to be unidirectional, as the system design does have some feedback loops built in (specifically for buffers).

We implemented this distributed architecture by deploying our navigator module on a single module node, and GTA V (along with control and screen capture module) on the master node. We could not deploy the data extraction module as any PC available to us could not handle all the models and scripts employed in that module. As the decision-making module relies on output from the data extraction module, this could not be implemented as well. Hence, this implementation was more of a proof of concept for our distributed architecture.

The master node sends a screen frame to the navigator. This screen frame is of maximum dimension 340x191x1 due to the limitations of a single UDP packet. A higher resolution image can be passed through by compressing the image or dividing the image into chunks at the server and recompiling it at the client. However, this would add complexity and latency to our system. Our navigator module requires an image of dimension 160x120x1, so the resolution limitation should not be an issue. More complex implementations of the navigator module might require an RGB image (triple channel) image, rather than the black and white image provided by the current implementation (single channel). This can be done by splitting an image into its RGB components (as demonstrated below). We use OpenCV library to perform image capture and processing. OpenCV stores images in the form of an array of arrays, where each pixel is contained in an array of size 3 with the 3 values contained being RGB values. Selecting the corresponding value from each pixel, and storing them in a new array. This contains 3 images of dimension 340x191x1, and these 3 images can be sent in order to the next module node and recompiled there. This method can be seen as preferable over using python compression libraries, as array operations (using the numpy library) can be parallelized on the GPU.

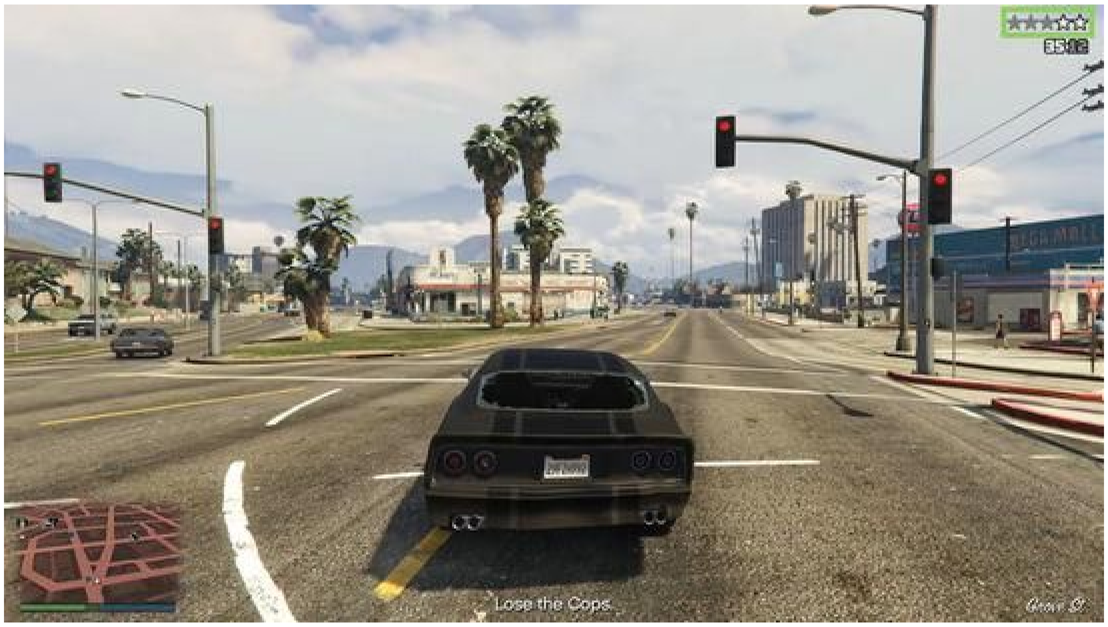


Fig 7.27: Original screen capture frame

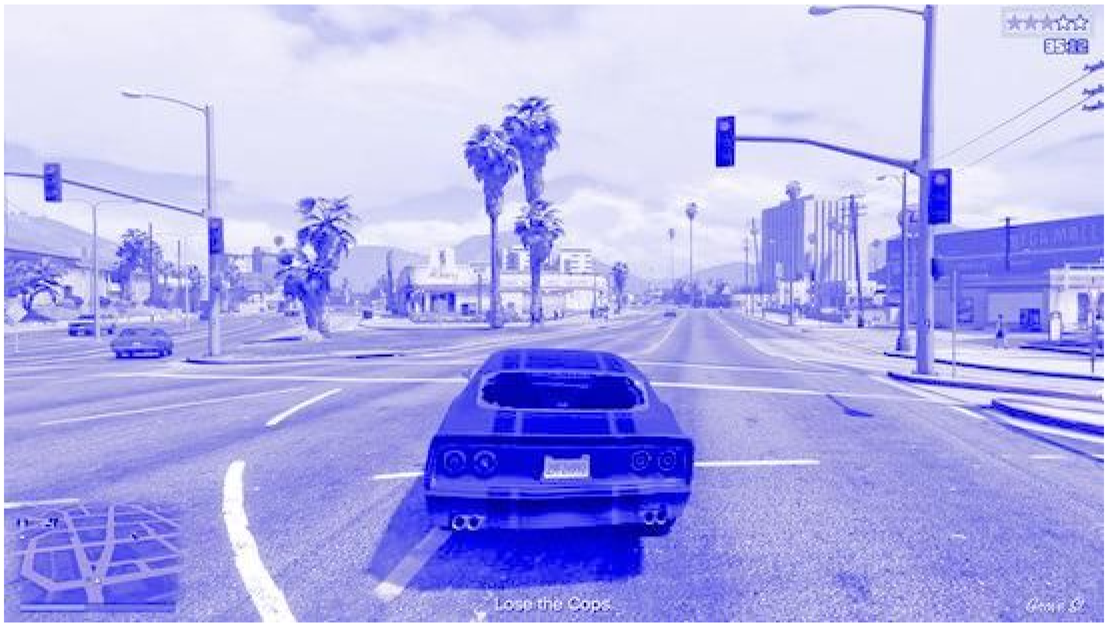


Fig 7.28: Image with only blue values

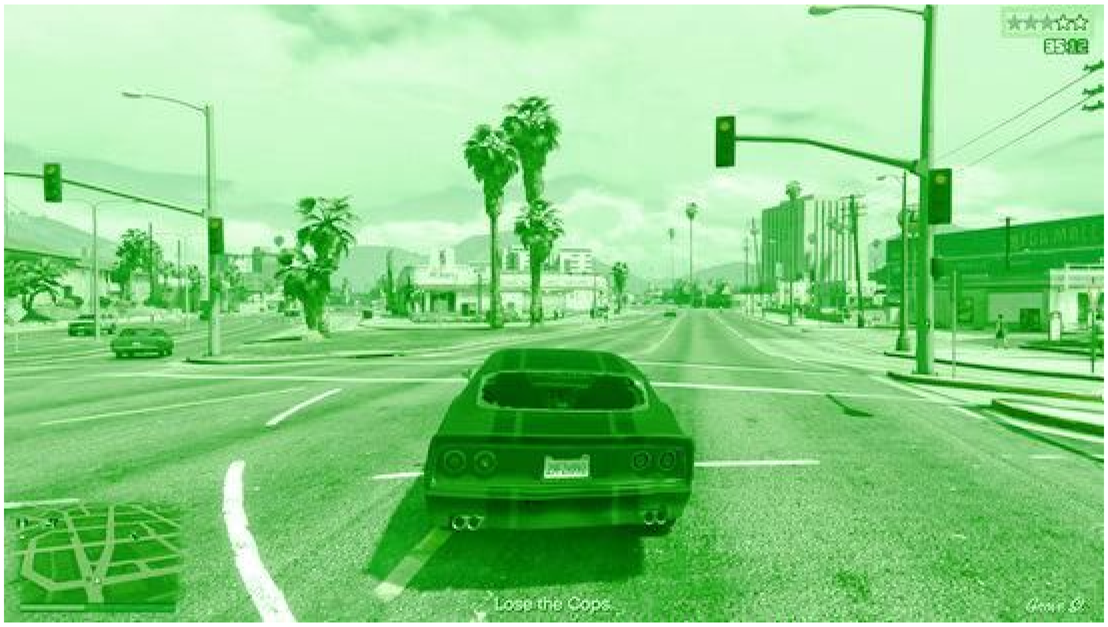


Fig 7.29: Image with only green values

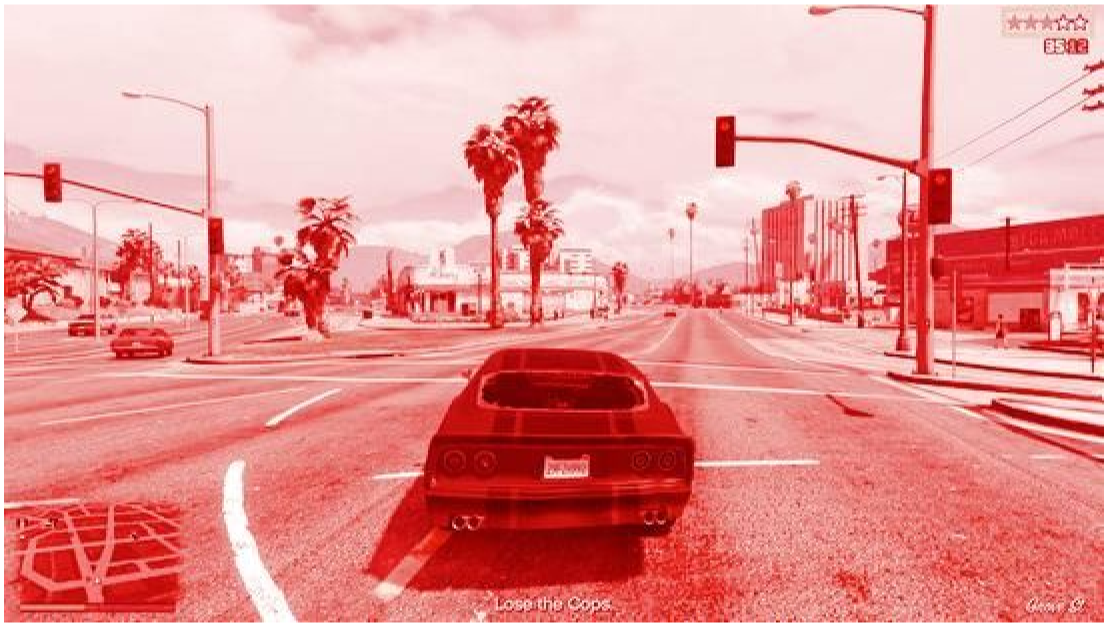


Fig 7.30: Image with only red values

The module node then sends the master node the output from the navigator module, the control module inside the master node interprets that output and in turn operates the actuators of the car inside GTA V.

A sequence table is not needed in this implementation as there is only one module that operates outside the master node.

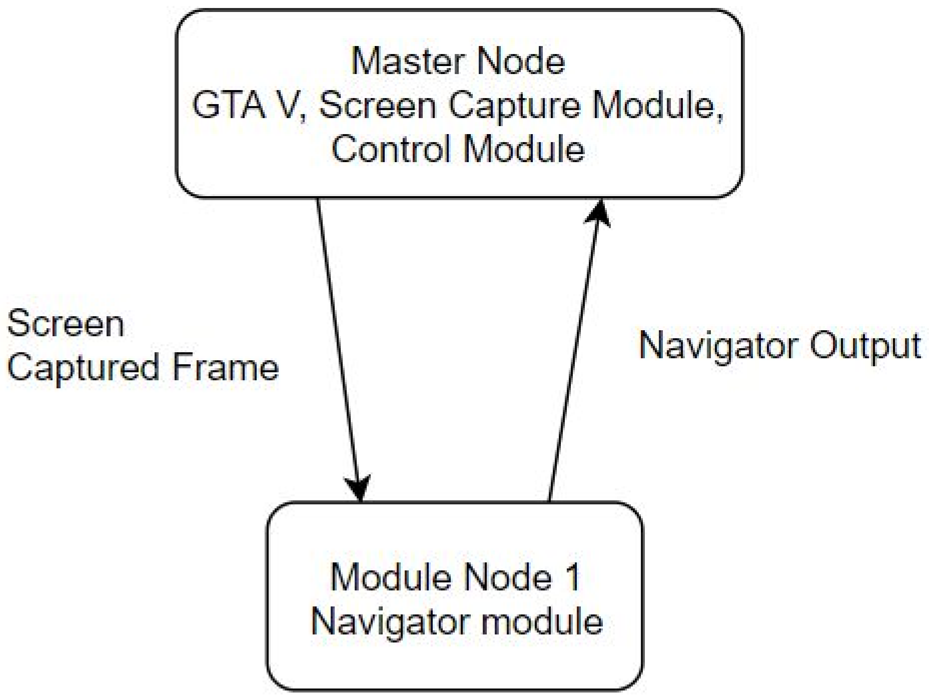


Fig 7.31: Implementation of distributed architecture

7.3 Additional scripts

Some additional scripts were used in the development of this project. They are detailed below:

1. Testing scripts:
   These scripts are detailed further in the testing section of the report
2. Training data collection script.
   This script was used to collect data for the navigator module. This script collects user input for controlling the car, as well as the screen frame at each instance.
3. Training data analysis script.
   This script was used to perform basic exploratory data analysis on the training data collected.
4. Training data modification script
   This script was used to modify the data, to change the captured screen frames dimensions, as well as adding decision flags to the data.
5. Navigator module training script
   This script was used to train the model in the navigator module using the data collected for training

6. Networking scripts for distributed nodes
   These scripts enable the master node and module node to communicate with each other.

8. GTA V

In this section we will discuss how GTA V was used in our project, the modifications made to it, its feasibility and drawbacks.

8.1 Modifications

8.1.2 DeepGTAV

Once the game starts, DeepGTAV will wait for TCP clients to connect on port 8000. Clients connected to DeepGTAV are allowed to send messages to GTAV to start and configure the research environment (*Start* and *Config* messages), send driving commands to control the vehicle (*Commands* message) and to stop the environment to fall back to normal gameplay (*Stop* message).
When the environment has been started with the *Start* message, DeepGTAV will start sending the data gathered from the game back to the client in JSON format, so the client can use it to store a dataset, run it through a self-driving agent.

The data sent back to the client and initial conditions will depend on the parameters sent by the client with the *Start* or *Config* messages. The parameters that can be controlled for instance are the rate of transmission, frame width and height, weather, time, type of vehicle, driving style, whether to get surrounding vehicles or pedestrians, type of reward function, etc.

This mod has seen less and less support, and it was not usable after the latest update to GTA V. We suspect the lack of support is because most projects have shifted over to Microsoft's AirSim (which will be discussed later).

We relied heavily on this mode to provide is most of the data from the game, losing the use of this mod setback a lot of our progress.

8.1.3 Other mods used in the project

All the mods used to manipulate and control GTA's environment are listed in the following table:

Mod Name & version	Mod Details
Community Script Hook V .NET 2.10.9	This mod allows the use of GTA V script native functions in custom *.asi plugins written in any .NET language in game
Enhanced Native Trainer(ENT) Update 40	This mod allows environmental control of the game. Such as controlling the weather, traffic, time and spawning various vehicles
NativeUI 1.9.1	This mod is responsible for building and displaying other game mods in game.
NFS gauge - RPM Gear Speedometer & Timer 2.63.2	This mod is responsible for measuring the speed of the car being driven in game.
Map Editor 2.13	This mod allows for the creation of custom roads, buildings, infrastructure, and spawning NPCs

8.2 Feasibility and issues

GTA V is a product of Rockstar Games. They are a game development company that is motivated by profit. The decisions they make and the updates they push for GTA V is to keep the game enjoyable and engaging for the players. Any modifications added onto the game by us are 3rd party software products. These software products are often open source and maintained by the community. When an official update is published for GTA V, often times, it causes these modifications to break and not function properly. This is the issue we faced when we were using the DeepGTAV mod for extensive data collection from the game. We could not get DeepGTAV to work with the most recent update to the game, and we had to alter most of our implementations to work around this issue. Additionally, Rockstar could the use of mods on their games at any time. They did ban mods before, but they reversed their decision.

Another issue with using GTA V to develop and test self-driving AIs is that it is limited in its customization. Though we can create custom tracks and spawn in objects to imitate certain scenarios, we are still limited by the entities developed in the game. For example, we can spawn in the house to customize a track, but we cannot make our custom house. The same is true for cars, traffic lights, barriers, etc.

Finally, as mentioned above, the mods added to the game are 3rd party software. They are freely available and maintained by the community. This helps with accessibility and availability of the mods, but it also means that support and updates for these software programs are not guaranteed.

However, GTA V is a massively open world game which gives unprecedented freedom to the user. Therefore, it has been used for the development of self-driving systems by other institutes as well (as mentioned in our related work section).

The use of GTA V for similar projects is dying down due to other more suitable environments such as AirSim.

9. Project management and estimation

9.1 Tasks and Sprints Backlog

9.1.1 Sprint 1

Daniyal and Mahmoud

Task	Task Duration(hours)
Conducting a literature review for research paper	75
Writing the research paper	80
Meeting Okan hoca	9
Revising the paper	35
Preparing a test plan	15
Revising test plan	8
Executing tests	30
Preparing test report	8

9.1.2 Sprint 2

Mahmoud

Task	Task Duration(hours)

Searching for new modding tools	4
Installing and setting up new modding tools for GTA V	2
meeting with the supervisor	9
Troubleshooting unstable modding tools	10
Literature review	15
Researching for a map building modding tool	7
Designing required circuit inside GTA V	25
Carrying out testing on GTA V and all modding tools	4
Collecting relevant data using the designed circuit	10
Revising collected data	3

Daniyal

Task	Task Duration(hours)
Restructuring CID Architecture	22
Designing and implementing a data extraction module	40
Training models in data extraction modules	15
Distributing modules over a network to different computing nodes	20
Modifying existing mods for GTA V	15
Modifying data mod data collection script	10
Designing the decision-making module	20
Implementing an abstracted environment in decision-making module	40
Validating changes to CID modules and architecture	60

9.2 Gantt Charts

9.2.1 Sprint 1

20 February – 15 April

Task Number	Task description	Start Date	Days to complete
1	Conducting literature review for research paper	20 Feb	30
2	Meeting Okan hoca	26 Feb	1
3	Writing the research paper	1 March	30
4	Meeting Okan hoca	5 March	1
5	Revising the paper	15 March	30
6	Meeting Okan hoca	12 March	1
7	Preparing test plan	18 March	7
8	Meeting Okan hoca	19 March	1
9	Revising test plan	21 March	7
10	Executing tests	22 March	9
11	Meeting Okan hoca	26 March	1
12	Preparing test report	30 March	7

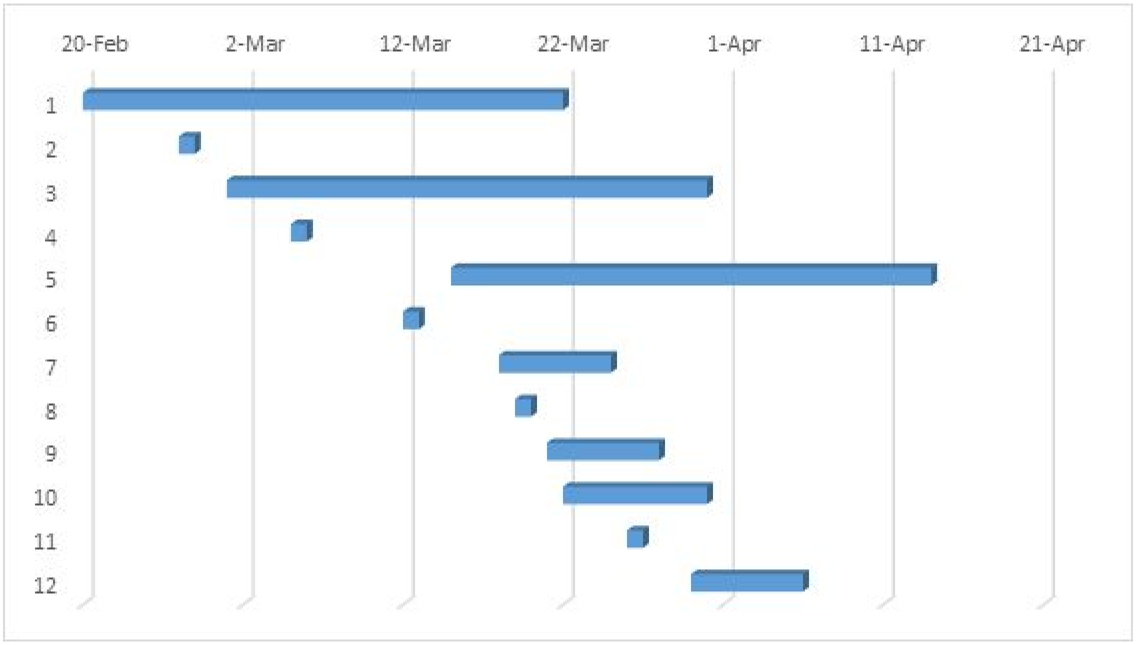


Fig 9.1: Sprint 1 Gantt chart

9.2.2 Sprint 2

15 April – 2 June

Task Number	Task description	Start Date	Days to complete
1	Installing and setting up new modding tools for GTA V	15 April	2
2	meeting with the supervisor	16 April	1
3	Troubleshooting unstable modding tools	16 April	3
4	Literature review	17 April	30
5	Designing required circuit inside GTA V	19 April	7

6	Carrying out testing on GTA V and all modding tools	21 April	2
7	Collecting relevant data using the designed circuit	25 April	7
8	Revising collected data	2 May	2
9	Searching for new modding tools	28 April	4
10	meeting with the supervisor	23 April	1
11	Installing and setting up new modding tools for GTA V	30 April	2
12	Restructuring CID Architecture	1 May	7
13	Designing and implementing data extraction module	8 May	10
14	Training models in data extraction modules	14 May	5
15	meeting with the supervisor	30 April	1
16	Distributing modules over network to different computing nodes	16 May	3
17	Modifying data mod data collection script	17 May	4
18	Designing decision-making module	18 May	6
19	meeting with the supervisor	14 May	1
20	Implementing an abstracted environment in decision-making module	19 May	8

21	Validating changes to CID modules and architecture	20 May	10
22	meeting with the supervisor	21 May	2

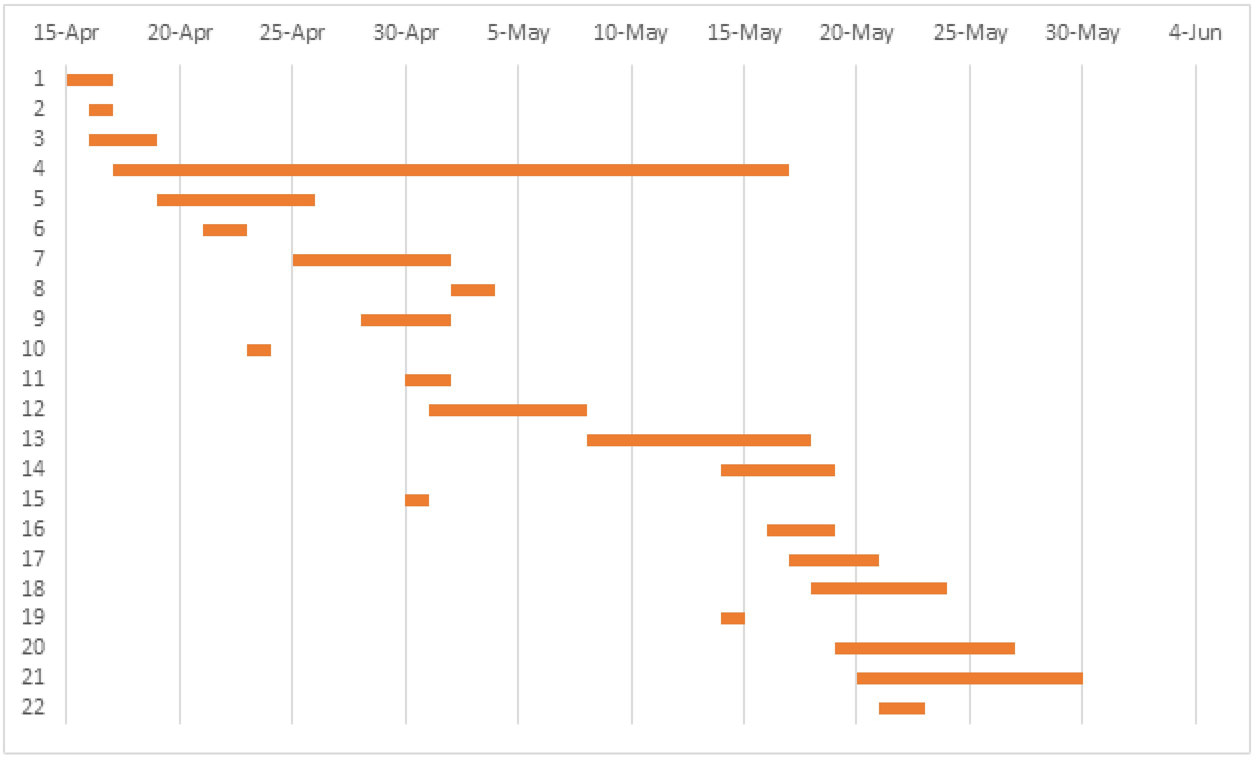


Fig 9.2: Sprint 2 Gantt chart

9.3 Sprint Review

In order to fully reorient our project based on feedback from CNG 491, a large portion of the time spent this semester was on doing literature review, looking up related work done and writing the research paper (as mentioned in the sections above).

After that, we modified all existing work done on the project to align with our new goals. The architecture of CID itself was modified to perform all the tasks mentioned above, and all the pre-existing modules were altered accordingly. Some of the work (including modules and sub-modules) had to be started from scratch, while others needed further updating and change after multiple setbacks due to events that were out of our control.

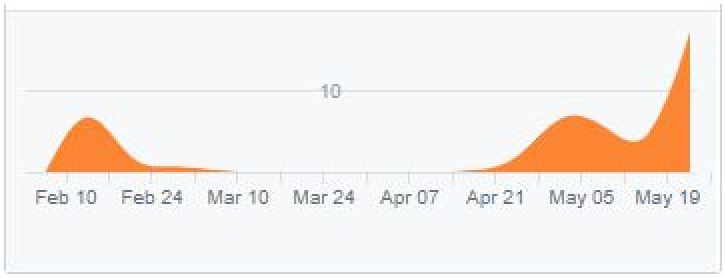


Fig 9.3: Commits on the project repository

The development of CID had to be accelerated, and the project itself has been in a constant state of change, even up till the final days of the semester. Thorough testing of CID has been performed on parts of the projects (modules and submodules).

9.4 Sprint Retrospective

Since this was our second semester and a couple of sprints later, we were more efficient at communicating which resulted in solid coordination between the 2 team members. Towards the end, the frequency of the meeting grew to address the new emerging problems like the previous semester. Informal meetings between the 2 members were the most productive, while formal meetings with Professor Okan served their purpose in giving us some guidance. The biggest hurdle for us was again the lack of resources for our project. In addition, we were set back by 2 weeks due to an accident that flooded the lab and consequently wiped our data. Overall, we are satisfied with how the team performed, however, there is still room improvement that could have been achieved with better resources.

Midterms and other course projects have derailed our progress somewhat but we managed to cope with our best ability. The help and understanding of the course supervisor and project supervisor have been crucial in keeping our project on track, even though we have missed some report and milestone deadlines.

9.5 LOC Size Metric

We are using Keras over the TensorFlow platform in Python.

LOC per module/subsystem = 720

$$KLOC(KDSI) = 10000 \textit{ lines of code}$$

Our Basic COCOMO model is obviously semi-detached.

$$Therefore, a = 3.0, b = 1.12, c = 0.35$$

$$MM(Man\ Month) = a * KDSI^{b} = 39.54$$

$$since\ it\ is\ semi\ detached, TDEV(total\ development\ time) = 2.5 * E^{0.35} = 2.5 * 39.54^{0.35}$$
$$= 9.1\ calender\ months$$

9.6 Project Milestones

Sprint 1 milestone:

- Finalizing the research paper
- Submitting the paper to Okan Hoca for application to conferences

Sprint 2 milestone:

- Finalizing circuits design
- Redesigning system based on literature review
- Building scripts needed for the project
- Building at least a majority of the modules for demonstration
- Testing developed modules
- Integrating submodules into modules

9.7 Testing

Due to the nature of the project, testing was split into 2 parts: 1. Testing system modules and scripts 2. Testing GTA V mods Custom scripts were developed to test system modules and scripts. For the testing of GTA V mods, we did not develop and scripts, as it did not apply. Instead, we tested the mods in the game, and manually performed the tests. The 3 types of testing performed are:

1. Unit testing

2. Integration testing

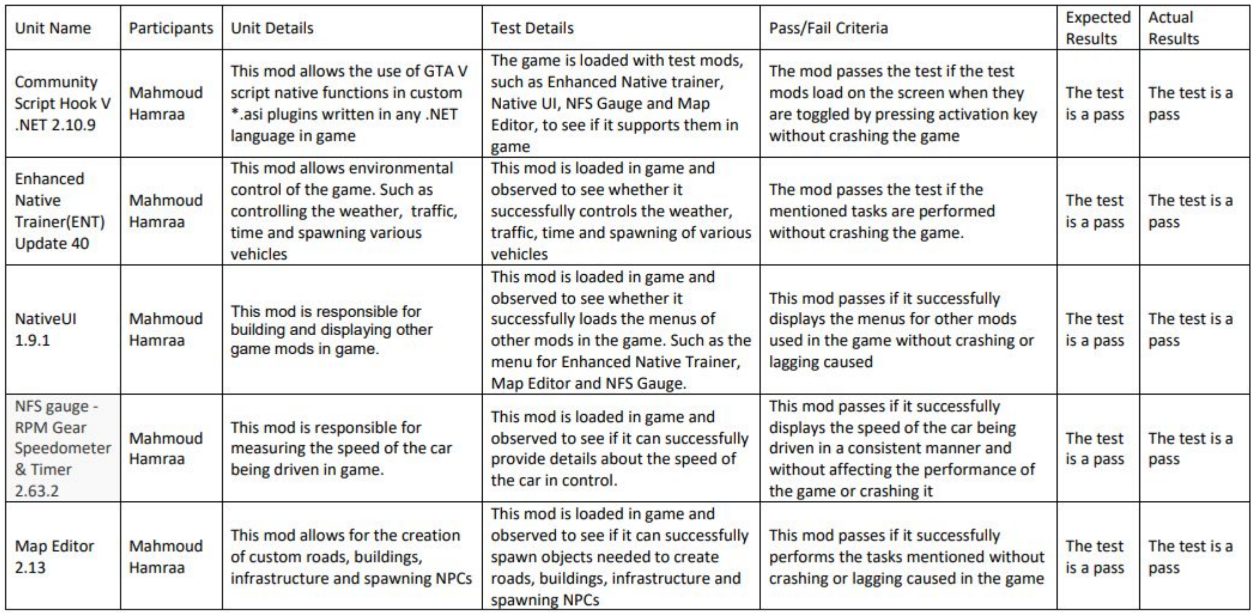
3. Performance testing

For unit testing, the units selected were either singular modules and scripts that are designed to perform one task or a collection of smaller sub-modules and scripts, that collectively work together to produce the desired output. For integration testing, we took a big bang approach. If the testing produced undesired results and fails, then we will proceed with a bottom-up approach to integration testing to isolate and detect the problem areas. For performance testing, we only tested a subset of the finalized parts of our project. This is because we are not concerned with the overall performance of the project, but we still want to identify any possible bottlenecks in the system that can affect the overall functionality of the system. We also identify the expected result and the actual result for each test perform. If we feel that the expected result will not be a pass for the test, then we will detail why we think the test will fail. If the actual result is a fail, then we will detail the identified cause of the failure. We do not perform security testing as it is not applicable to our project. We do not perform Alpha/Beta testing as our project is a research-oriented project and we do not intend to release for general users, such testing is not applicable to our project

9.7.1 Unit Testing

Unit Name	Participants	Unit Details	Test Details	Pass/Fail Criteria	Expected Results	Actual Results
Network Communication script	Daniyal Selani	This script is responsible for sending and receiving data between modules that are located on 2 different computing nodes on the same network. The module is also responsible for encoding and decoding the messages.	The script on the master pc is given a test data frame to send to the slave pc. The slave pc returns a test data array.	The script on the master pc encodes and sends a data frame. The unit on the slave pc receives, decodes and displays the data frame. The slave pc encodes a response, sends it to the master pc, and the master pc decodes it. If no errors are raised, the test is a pass	The test is a pass	The test is a pass.
Data capture module	Daniyal Selani	This module is responsible for capturing and transforming frames from the game GTA V to the proper dimensions that are needed as input to the different modules of the system.	The module will be required to capture a certain region of the screen, which is of dimension (1280, 720, 3). The module will display the captured region in the following dimensions: 1. (480, 270, 3) 2. (340, 191, 1) 3. (160, 120, 1)	The module passes the test if it captures and displays screen-captured frames correctly without any distortion and/or corruption.	The test is a pass	The test is a pass.
Gameplay recording script	Daniyal Selani	This script records gameplay of GTA V for training models. It records screen capture and the corresponding action taken by user. The script records gameplay and stores it as a ".npy" file.	The script will be required to record GTA V gameplay (frame capture of the frame and corresponding key input from the user). The captured gameplay will be played back and inspected.	The script passes the test if the recorded gameplay can be played back successfully without any distortion or corruption and no saved files are overwritten.	The test is a pass	The test is a pass.
Data extraction module	Daniyal Selani	This module is responsible for extracting relevant information from the input frames from GTA V.	The module is composed of multiple smaller models that perform different computer vision and processing tasks on the input frame. Since these models work together to extract information from the data to describe the state of the environment, it shall be tested it as a single unit. The module will be fed prerecorded gameplay screen capture, and the module will output extracted information of each frame in a json file.	The module passes the test if the module produces the correct output in the form of a json file and it contains the following information: 1.Number of objects in a frame 2. Object classes 3. Confidence level of object prediction 4. Object location In frame 5. Area of bounding box of object 5. Lane/Path of the car in the frame 6. Location of lane/path 6. Segmented area of the lane/path 7. If any object is present in the area of the lane/path	The test is a pass	The test is a pass.
Control module	Daniyal Selani	This module takes output from the system as input, and acts as an actuator to control the car in the game based on the input.	The module will be provided a prerecorded set of control outputs as inputs. The module will be required to translate those inputs into in game actions.	The module passes the test if it can successfully read the inputs and translate them into in game movements.	The test is a pass	The test is a pass.

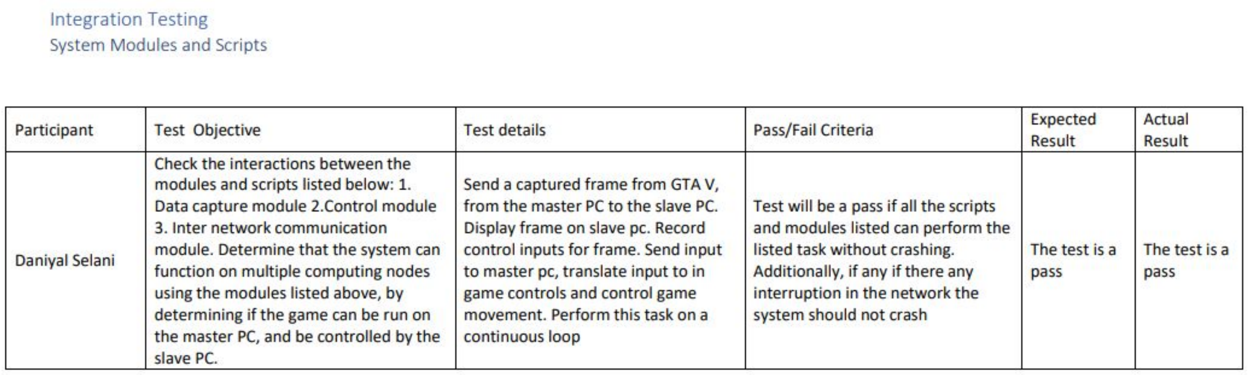
Fig 9.4 Unit Testing for System Modules and Scripts



Unit Name	Participants	Unit Details	Test Details	Pass/Fail Criteria	Expected Results	Actual Results
Community Script Hook V .NET 2.10.9	Mahmoud Hamraa	This mod allows the use of GTA V script native functions in custom *.asi plugins written in any .NET language in game	The game is loaded with test mods, such as Enhanced Native trainer, Native UI, NFS Gauge and Map Editor, to see if it supports them in game	The mod passes the test if the test mods load on the screen when they are toggled by pressing activation key without crashing the game	The test is a pass	The test is a pass
Enhanced Native Trainer(ENT) Update 40	Mahmoud Hamraa	This mod allows environmental control of the game. Such as controlling the weather, traffic, time and spawning various vehicles	This mod is loaded in game and observed to see whether it successfully controls the weather, traffic, time and spawning of various vehicles	The mod passes the test if the mentioned tasks are performed without crashing the game.	The test is a pass	The test is a pass
NativeUI 1.9.1	Mahmoud Hamraa	This mod is responsible for building and displaying other game mods in game.	This mod is loaded in game and observed to see whether it successfully loads the menus of other mods in the game. Such as the menu for Enhanced Native Trainer, Map Editor and NFS Gauge.	This mod passes if it successfully displays the menus for other mods used in the game without crashing or lagging caused	The test is a pass	The test is a pass
NFS gauge - RPM Gear Speedometer & Timer 2.63.2	Mahmoud Hamraa	This mod is responsible for measuring the speed of the car being driven in game.	This mod is loaded in game and observed to see if it can successfully provide details about the speed of the car in control.	This mod passes if it successfully displays the speed of the car being driven in a consistent manner and without affecting the performance of the game or crashing it	The test is a pass	The test is a pass
Map Editor 2.13	Mahmoud Hamraa	This mod allows for the creation of custom roads, buildings, infrastructure and spawning NPCs	This mod is loaded in game and observed to see if it can successfully spawn objects needed to create roads, buildings, infrastructure and spawning NPCs	This mod passes if it successfully performs the tasks mentioned without crashing or lagging caused in the game	The test is a pass	The test is a pass

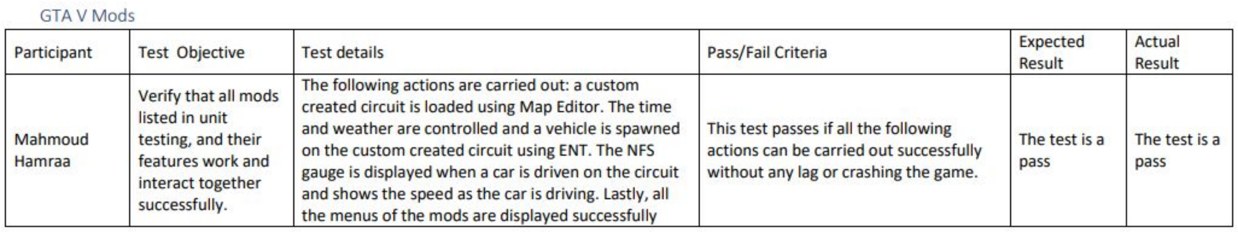
Fig 9.5 Unit Testing for GTA V mods

9.7.2 Integration Testing



Integration Testing
System Modules and Scripts

Participant	Test Objective	Test details	Pass/Fail Criteria	Expected Result	Actual Result
Daniyal Selani	Check the interactions between the modules and scripts listed below: 1. Data capture module 2.Control module 3. Inter network communication module. Determine that the system can function on multiple computing nodes using the modules listed above, by determining if the game can be run on the master PC, and be controlled by the slave PC.	Send a captured frame from GTA V, from the master PC to the slave PC. Display frame on slave pc. Record control inputs for frame. Send input to master pc, translate input to in game controls and control game movement. Perform this task on a continuous loop	Test will be a pass if all the scripts and modules listed can perform the listed task without crashing. Additionally, if any if there any interruption in the network the system should not crash	The test is a pass	The test is a pass

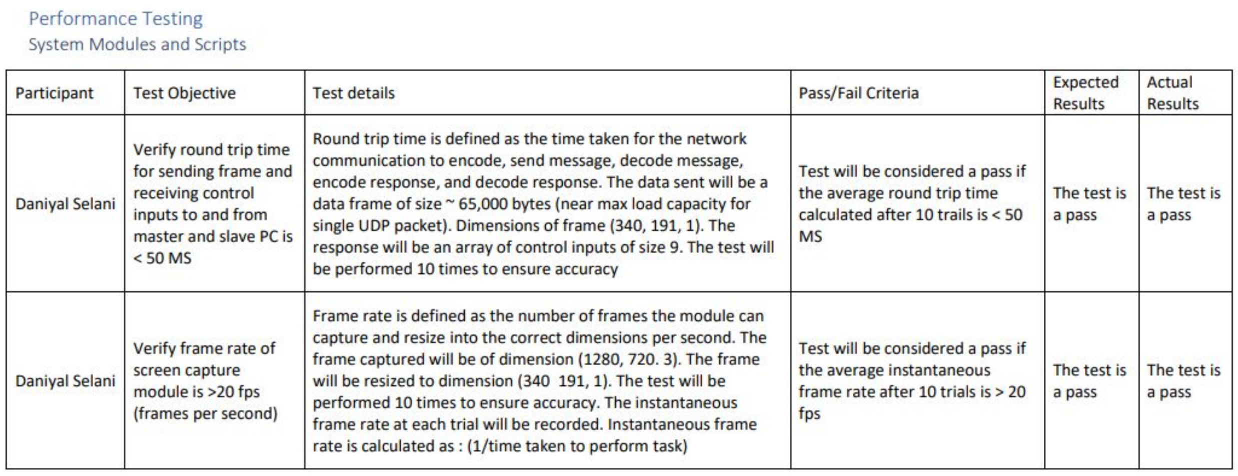


GTA V Mods

Participant	Test Objective	Test details	Pass/Fail Criteria	Expected Result	Actual Result
Mahmoud Hamraa	Verify that all mods listed in unit testing, and their features work and interact together successfully.	The following actions are carried out: a custom created circuit is loaded using Map Editor. The time and weather are controlled and a vehicle is spawned on the custom created circuit using ENT. The NFS gauge is displayed when a car is driven on the circuit and shows the speed as the car is driving. Lastly, all the menus of the mods are displayed successfully	This test passes if all the following actions can be carried out successfully without any lag or crashing the game.	The test is a pass	The test is a pass

Fig 9.6 Integration Testing for System modules and scripts

9.7.3 Performance Testing



Performance Testing
System Modules and Scripts

Participant	Test Objective	Test details	Pass/Fail Criteria	Expected Results	Actual Results
Daniyal Selani	Verify round trip time for sending frame and receiving control inputs to and from master and slave PC is < 50 MS	Round trip time is defined as the time taken for the network communication to encode, send message, decode message, encode response, and decode response. The data sent will be a data frame of size ~ 65,000 bytes (near max load capacity for single UDP packet). Dimensions of frame (340, 191, 1). The response will be an array of control inputs of size 9. The test will be performed 10 times to ensure accuracy	Test will be considered a pass if the average round trip time calculated after 10 trails is < 50 MS	The test is a pass	The test is a pass
Daniyal Selani	Verify frame rate of screen capture module is >20 fps (frames per second)	Frame rate is defined as the number of frames the module can capture and resize into the correct dimensions per second. The frame captured will be of dimension (1280, 720. 3). The frame will be resized to dimension (340 191, 1). The test will be performed 10 times to ensure accuracy. The instantaneous frame rate at each trial will be recorded. Instantaneous frame rate is calculated as : (1/time taken to perform task)	Test will be considered a pass if the average instantaneous frame rate after 10 trials is > 20 fps	The test is a pass	The test is a pass

Fig 9.7 Performance Testing for System modules and scripts

10. Self-assessment of system

10.1 Objectives

In this section, we will assess ourselves of how close we came to the fulfillment of the 3 objectives that we defined before. These were:

1. Is modular in design for easy upgradability, interpretability and debugging:
   The design of our system was from the beginning centered around modularity. The output from each module is well defined, allowing easy debugging of different parts of the system, as well as easy interpretability of what each part of the system is doing. For this semester, we altered the modules to perform at least 1 task required for self-driving cars, compartmentalizing tasks allows further interpretability of the operation of the system.
   An example of how useful the modularity of our system is how we handled the loss of one of our crucial mods. We were relying on DeepGTAV for most of our in-game data collection. However, after an update, this mod broke and we could not seem to find a fix for it, or fix it ourselves (since this is not in the realm of our expertise). We decided to rely more on computer vision for data extraction. This meant we had to change the data extraction module extensively, but the rest of the system did not need any alterations.

2. Can be easily deployed on consumer grade hardware for testing and validation:
   Availability of hardware resources has been the biggest bottleneck in our project. After doing an extensive literature survey, we decided that our modules should be able to work a common

consumer-grade hardware (as defined in the problem definition and scope section). All of the neural network models, GTA V and all the mods, employed for our project should be able to run on consumer grade hardware, using the distributed architecture detailed above.

3. Outlines an approach for debugging the behavior of the autonomous system that is developed. This objective ties in with the interpretability of the system. Our biggest criticism of existing research on creating self-driving cars in GTA V is that they only employ neural network models that can imitate user driving behavior. We used these models to establish baseline autonomy and then modify their behaviors using agents whose decisions can be easily logged and interpreted.

We are aware that we could not implement every module, and even those modules that we did implement, they have plenty of room for improvement. However, given the restrictions and setbacks that we faced (as mentioned above), we believe that we can be, at the very least, satisfied with the progress made on this project.

10.2 Simulation Environment

It is to be noted, that the use of GTA V is no longer the most ideal simulation environment for this project. Microsoft recently released an open source project called AirSim [https://github.com/microsoft/AirSim]. Which uses the Unity engine to create an open source simulation, specifically designed for the development of a self-driving system. It has inbuilt data collection and environment manipulation tools that make it extremely ideal for our project. AirSim is still in beta development but it is open to the community for anyone to use. If we could start this project again, this would be the first major change made to the project.

10.3 Scaling Autonomy of System

The 5 levels of autonomy defined by NHTSA [21]

- Level 0: No Automation
- Level 1: Driver Assistance

- Level 2: Partial Automation
- Level 3: Conditional Automation
- Level 4: High Automation
- Level 5: Full Automation

Let's asses how our system can be used to scale to the different levels of automation. It is important that our design is flexible enough to scale to the level of autonomy a developer may want to target. We shall look at this in detail for level 1, level 3, and level 5

10.2.1 Level 1: Driver Assistance

At this level, the vehicle can assist with some functions, but the driver still handles all accelerating, braking, and monitoring of the surrounding environment. An example would be a system that notifies the driver of unsafe lane changes, or if the driver is going to fast to brake in time to not hit the car in front.

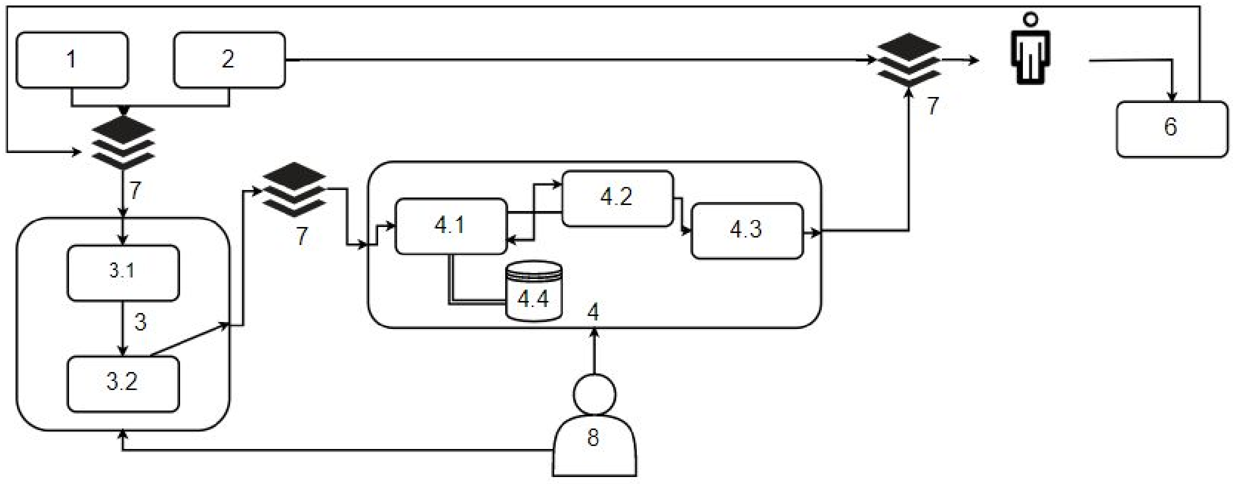


Fig 10.1: System modified for level 1

To achieve this level of autonomy, the navigator module can be replaced with a human driver. The decision from the decision-making module simply informs of the driver of impending disasters or possible actions. All the other modules function the same.

10.2.2 Level 3: Conditional Automation

This level is marked by both the execution of steering and acceleration/deceleration and the monitoring of the driving environment. In levels zero through two, the driver does all the monitoring. At level three, the driver is still required, but the automobile can perform all aspects of the driving task under some circumstances. The driver must hand off the control of the car to the system, but must be able to regain control whenever he/she wants to.

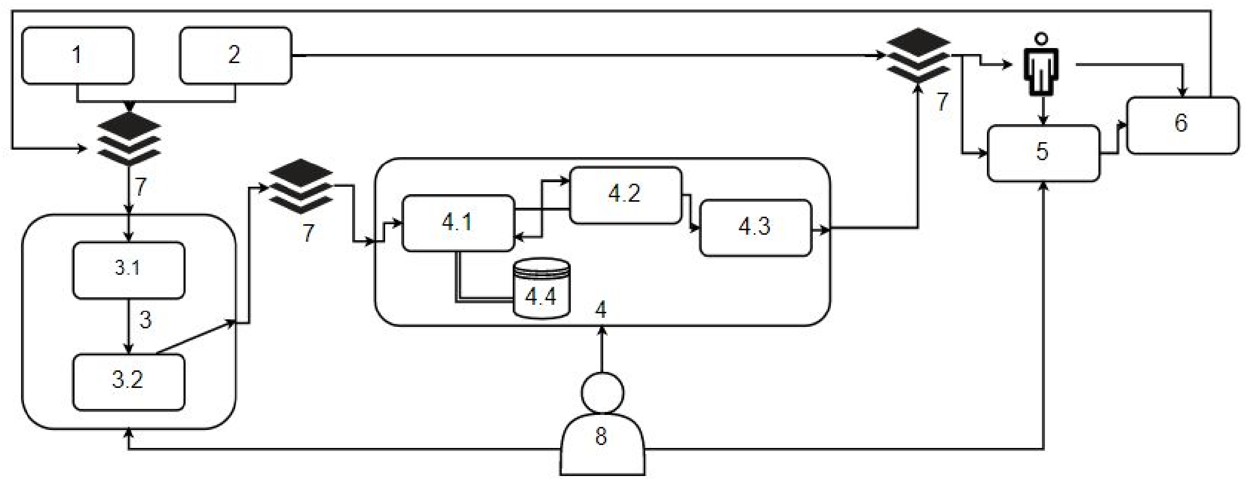


Fig 10.2: System modified for level 2

To achieve this level of autonomy, the system still notifies the user of the decision made by it, but the user can choose to hand of control to the system completely.

10.2.3 Level 5: Full Autonomy

This level of autonomous driving requires absolutely no human attention. There is no need for pedals, brakes, or a steering wheel, as the autonomous vehicle system controls all critical tasks, monitoring of the environment and identification of unique driving conditions like traffic jams. The original system architecture is design to reach this level of autonomy.

11. Installation and Deployment

All of the code, including the additional scripts, are uploaded to a Github repository. The current iteration of our system is called CIDv0.03. The code for this iteration is in a folder labeled CIDv0.03. All the code for the modules is in individual folders. Some modules are grouped together in a single folder for easier implementation. The modules are implemented as different libraries. Everything can be accessed through the following link:

https://github.com/daniyal9538/CID

11.1 Data Collection Module:

This module contains a script which communicates with the DeepGTAV mod. This mod is no longer functional. The script needs to be connected to the same LAN as the PC running GTA V with DeepGTAV running on it. By default, it will look for the mod on the local IP address, but the IP address of any PC on the LAN can be passed to it. The output of this script is a python dictionary with the relevant attributes. This data can be dumped into a JSON file for easy viewing.

11.2 Data Extraction Module:

This module contains a jupyter notebook and a secondary script.

11.2.1 motion.py

The script labeled motion.py provides pixel calculation for motion tracking. It contains functions that return the delta value between the current image and the previous n images.

11.2.1 Jupyter Notebook

The jupyter notebook contains a pipeline for data extraction. The pipeline is called as a single function, the input to it is an image of dimension 480x270x3. The main output of the pipeline is a python dictionary containing all extracted data. This data can be dumped into a JSON file for easy viewing.

Please note that the pipeline requires a pre-trained camera matrix, that is currently hosted on our personal google cloud storage bucket.

11.3 Decision Making Module:

This module contains 4 scripts.

11.3.1 helper.py

This script contains helper functions for the other scripts

11.3.2 constructor.py

This script is an internal map constructor sub-module. It takes in the JSON file produced by the data extraction module and uses it to build the internal map for the system.

11.3.3 sampler.py

This is a basic demo implementation of the decision sampler sub-module. It contains a function which takes in relevant data and outputs a binary flag array. The decision being sampled is to either turn, left or right, or do nothing.

11.3.4 decision_making.py

This script implements a dummy agent (controlled by the user) in the internal map. It output a JSON file of the decision made and sampled, along with other relevant data for interpretability and debugging.

11.4 Navigator, Screen Capture, Control Modules

This folder contains the scripts needed for the operation of the above 3 modules. All the modules are integrated into the script called navigator.py, executing this script instantiates all the modules..

The control module is distributed in the scripts labeled keys.py, getkey.py, directkeys.py The screen capture module is implemented in the grabscreen.py script The model that the navigator uses is defined in the script models.py. It is to be noted, that the navigator script requires a .h5 file which contains all the pre-trained weights for the navigator module's model.

11.5 Dependencies

The project was developed on python 3.6.7. The following libraries and packages were used(these do not include libraries that come preinstalled with the python package):

- Tensorflow
- Keras
- Pygame
- IPython
- Image.AI
- OpenCV
- Numpy
- Matplotlib
- PXYInput
- Cuda package
- CuDNN DLL
- PyWin32
- Win32Gui

Note: For all the libraries mentioned above, the latest stable build was used.

12. References

1. Tilley, A. (2017, October 04). Grand Theft Auto V: The Rise And Fall Of The DIY Self-Driving Car Lab. Retrieved from https://www.forbes.com/sites/aarontilley/2017/10/04/grand-theft-auto-v-the-rise-and-fall-of-the-diy-self-driving-car-lab/#18a5c5007d7a
2. Aitorzip. (2017, November 06). Aitorzip/DeepGTAV. Retrieved from https://github.com/aitorzip/DeepGTAV
3. Microsoft. (2019, May 17). Microsoft/AirSim. Retrieved from https://github.com/microsoft/AirSim
4. The 5 Autonomous Driving Levels Explained. (2018, August 05). Retrieved from https://www.iotforall.com/5-autonomous-driving-levels-explained/
5. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812332
6. April 2019. (n.d.). Retrieved from https://store.steampowered.com/hwsurvey/videocard/
7. Lanier, L., & Lanier, L. (2019, April 30). Steam Now Has One Billion Accounts (And 90 Million Active Users). Retrieved from https://variety.com/2019/gaming/news/steam-one-billion-accounts-1203201159/
8. Balaban, M. (2019, February 19). Titan RTX Deep Learning Benchmarks. Retrieved from https://lambdalabs.com/blog/titan-rtx-tensorflow-benchmarks/
9. Self-Driving Cars Explained. (n.d.). Retrieved from https://www.ucsusa.org/clean-vehicles/how-self-driving-cars-work

10. Lambert, F., Lambert, F., Fred, Fred, & Electrek. (2017, June 14). Tesla has opened the floodgates of Autopilot data gathering. Retrieved from https://electrek.co/2017/06/14/tesla-autopilot-data-floodgates/

11. Geometric Transformations of Images. (n.d.). Retrieved from https://docs.opencv.org/3.1.0/da/d6e/tutorial_py_geometric_transformations.html

12. Shelhamer, Evan, Jonathan, Darrell, & Trevor. (2016, May 20). Fully Convolutional Networks for Semantic Segmentation. Retrieved from https://arxiv.org/abs/1605.06211

13. Zuccolo, R., & Zuccolo, R. (2017, April 03). Self-driving Cars - Advanced computer vision with OpenCV, finding lane lines. Retrieved from https://chatbotslife.com/self-driving-cars-advanced-computer-vision-with-opencv-finding-lane-lines-488a411b2c3d?gi=b9d7062c89d4

14. Redmon, J. (n.d.). Retrieved from https://pjreddie.com/darknet/yolo/

15. Common Objects in Context. (n.d.). Retrieved from http://cocodataset.org/#home

16. OpenAI. (n.d.). A toolkit for developing and comparing reinforcement learning algorithms. Retrieved from https://gym.openai.com/

17. News New here? (n.d.). Retrieved from https://www.pygame.org/

18. Bayangan1991. (n.d.). Bayangan1991/PYXInput. Retrieved from https://github.com/bayangan1991/PYXInput

19. Krizhevsky, Sutskever, Hinton. "ImageNet Classification with Deep Convolutional Neural Networks ". Retrieved from https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

20. Zahid, Riegler, Skeie. "Distributed Deep Learning." Retrieved from https://www.uis.no/getfile.php/13458394/Forskning/Vedlegg/10%20IKT/deep_learning_intro.pdf

21. Hendrickson, J. (2019, January 25). What Are the Different Self-Driving Car "Levels" of Autonomy? Retrieved from https://www.howtogeek.com/401759/what-are-the-different-self-driving-car-levels-of-autonomy/

22. "Self-Driving Cars Explained." *Union of Concerned Scientists*, www.ucsusa.org/clean-vehicles/how-self-driving-cars-work.

23. Marshall, A. (2018, February 01). Self-Driving Cars Have Hit Peak Hype-Now They Face the Trough of Disillusionment. Retrieved from https://www.wired.com/story/self-driving-cars-challenges/

24. Dass, R. (2018, September 14). 5 Key Challenges faced by Self-driving cars. Retrieved from https://medium.com/@ritidass29/5-key-challenges-faced-by-self-driving-cars-ed04e969301e

25. Ai, C. (2017, June 26). How does openpilot work? Retrieved from https://medium.com/@comma_ai/how-does-openpilot-work-c7076d4407b3

26. Laukkonen, J. (2018, August 17). Take a Ride in Waymo's Self-Driving Car. Retrieved from https://www.lifewire.com/waymo-self-driving-cars-4171314

27. Lee, T. B. (2018, December 07). Even self-driving leader Waymo is struggling to reach full autonomy. Retrieved from https://arstechnica.com/cars/2018/12/waymos-lame-public-driverless-launch-not-driverless-and-barely-public/

28. Hawkins, A. J. (2018, July 13). George Hotz is on a hacker crusade against the 'scam' of self-driving cars. Retrieved from https://www.theverge.com/2018/7/13/17561484/george-hotz-comma-ai-self-driving-car-scam-diy-kit

29. Psyber. (n.d.). Retrieved from https://psyb

30. Filipowicz, A. (2018, May 08). Learning to Recognize Distance to Stop Signs Using the Virtual World ... Retrieved from https://www.slideshare.net/ArturFilipowicz/learning-to-recognize-distance-to-stop-signs-using-the-virtual-world-of-grand-theft-auto-5

31. Martinez, M., Sitawarin, C., Finch, K., Meincke, L., Yablonski, A., & Kornhauser, A. L. (1970, January 01). Beyond Grand Theft Auto V for Training, Testing and Enhancing Deep Learning in Self Driving Cars - Semantic Scholar. Retrieved from https://www.semanticscholar.org/paper/Beyond-Grand-Theft-Auto-V-for-Training,-Testing-and-Martinez-Sitawarin/a5a00a5612adbb5c1e01707f4c62109c77492240