UPGRADING A MINIATURE AUTONOMOUS CAR TESTBED FOR USE AS AN EDUCATIONAL TOOL

Name	Rhys Kessey
Student ID	20925234
School	Electrical & Electronic
Supervisor(s)	Dr Ghulam Mubashar / Dr Tim French
Date Submitted	21/10/2018
Word Count	6921
Report Type	Design & Build

ABSTRACT

Autonomous cars are a hot topic right now and there is a large variety of research being undertaken into creating safe and robust methods of driving. Of these, an important discussion topic is the concept of simulating driving under non-ideal conditions. Any old simulation can tell you cars should drive straight, but what happens when the cars don't drive as precisely as expected? The miniature autonomous car testbed here at UWA allows us to explore the dark corner of the world that contains these non-ideal simulation conditions.

The students who built the testbed last semester gave me a very good starting point from which I launched my investigations. A Raspberry pi microcomputer detects and controls a number of small blue tooth cars as they attempt to drive around the white surface. A projector displays the image of a road on top of this white surface, giving the effect of cars driving around a map. The primary challenge presented with this particular setup is that we need to use a Raspberry Pi camera to track the tiny cars as they drive within a virtual environment.

My research is focused on upgrading this testbed to allow for greater usability which will allow it to be used as an educational tool for classrooms or on show at open day. In order to verify the effectiveness of the testbed, I have gathered information about some simple traffic routing algorithms and ported it across for use with the testbed. Using the results produced from these tests I have assessed how effectively the testbed now acts as an educational tool and what future work can be done to further enhance the simulations.

CONTENTS

1. INTRODUCTION

The Miniature Autonomous Car Testbed is a wooden structure that projects an image onto a white board so that small blue-tooth controlled cars can drive around a virtual track and demonstrate interesting autonomous car algorithms. It was created by Ray Barker and Aaron Hurst last year and has been picked up by myself and two other students (Zen Ly and Michael Finn) for further development this year. When we received the testbed, the Raspberry Pi controller was already set up to communicate with the cars that did not have any functionality other than simply driving around the projected track.

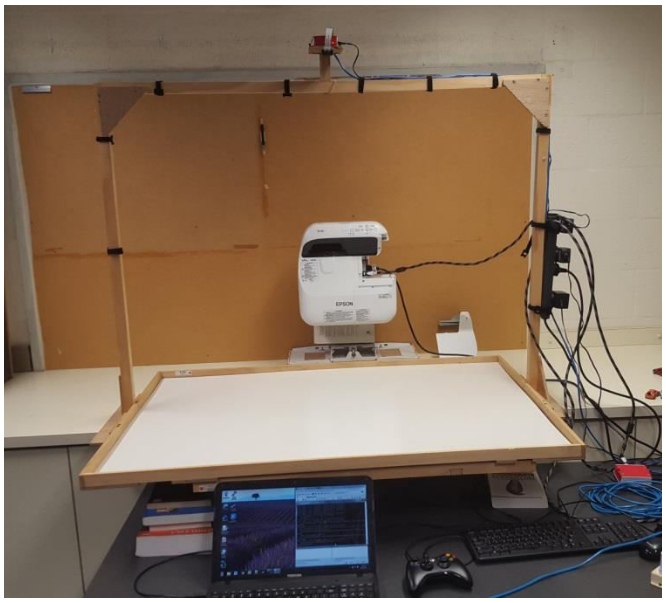


Image 1: Full view of the testbed.

One of the first issues I addressed was the clumped nature of the current code, solved by effectively rewriting all of the existing functionality that was used to control the cars. Along with Michael Finn, I converted the existing structure that integrated camera tracking and car functionality in the same script into a modular solution that split the car control algorithms into their individual files, making it far easier to swap and change to a different car control algorithm whenever required. This was the first step towards creating a streamlined user experience, but not what I am focusing my research project towards.

My goal has always been to upgrade the testbed to make it more 'user-friendly', to allow other students to come and implement their own autonomous driving algorithms using the testbed with minimal effort and maximum result.

This document will explain the process chronologically, first defining the **Scope** and **Motivations** of the project before moving into how I constructed the new **Graphical User Interface** which allows for an intuitive way to quickly launch a simulation. Following this will be an explanation of the **Application Programming Interface** that allows students to effectively port their autonomous routing algorithms for use with the testbed. Lastly there will be a discussion of the **Methodology** and **Results** of my algorithm tests, explaining how well the testbed completes its intended task of acting as an effective educational tool.

2. SCOPE

This section of the report will detail the purpose behind my project, its importance to the field and how I plan on assessing my outcomes.

2.1 PROBLEM IDENTIFICATION

There is a huge variety of software simulations in the world right now, ranging from professional suites to student projects. These offer pristine environments where algorithms can be tested for their functionality without any variable outcomes. In reality however, conditions are not so perfect so it is important to be able to run simulations in non-ideal environments.

Having experience programming car control agents to drive in non-ideal environments is essential moving forward as the autonomous car industry continues to grow and approach wide-scale deployment. This is because programming in the environment offered by this testbed makes you solve different problems than you would see elsewhere.

I can see this testbed helping to fill the gap in accessible non-ideal autonomous car environments by acting as an educational tool for students. Due to the hardware used in this project, there are physical limitations that mean the testbed can only be used as an introduction to unique autonomous environments and a different platform will be required for more intensive testing purposes but that is still a solid step away from pure software simulation.

2.2 OBJECTIVES

The testbed is a unique option for one of these non-ideal environments and so my goal is to bring it up to a level where it can be easily used by other students who are interested in porting their algorithms across.

After rewriting the control code that was left to us by the students from last year, I constructed three distinct goals that I used to structure my project of **"Upgrading the Testbed for use as an Educational Tool"**.

ID	Description	Achieved?
1	Cars Drive around Simple Track	
2	Streamline Process for Creating new Agent Strategies	
3	Streamline Process for Loading new Agent Strategies	

Table 1: Project Objectives

Objective 1 was set in place to ensure that the result of my upgrade procedure at least offers the same functionality as it did when I first started the project. If I made changes and it 'broke' the testbed, I would be required to label this endeavour as a substantial failure.

Objective 2 is more of a qualitative assessment of the user experience when programming new agent strategies. This is an assessment of the usability of the API that I have created and how easily a new student might come down and convert his/her own autonomous routing algorithm into a format usable by the testbed. This objective also requires that the API have the potential to create complex agent control strategies.

Objective 3 is a qualitative assessment of the user experience when loading their new agent strategies onto the testbed and running a simulation. This is an assessment of the GUI that I have created and how easily a new student might come down and launch a simulation using their desired options.

3. LITERATURE REVIEW

This section is provided to bring to our attention the history of work in this field and how effective it has been. The idea of autonomous cars has been a very popular one recently so there are some very important contextual factors that I need to investigate. The miniature autonomous car testbed at UWA is unique in the field so looking at other iterations of this design is not an option, instead I will search for similar structures and what they're intended uses are; this will give me a good insight into what factors I should consider when upgrading the testbed at UWA.

3.1 BACKGROUND

The push for the growth of Autonomous cars has been slowly gaining strength since the 1920s [1] when the 'Achen' motor company got one of their cars to drive itself and even honk its horn at pedestrians. The idea of self-driving vehicles has fascinated humanity for a very long time. Since then, there has been a large amount of research put in to making it possible both physically and ethically; as there are a number of these challenges that have stood in our way.

Rigorous physical testing is an effective way to beat these physical challenges but it is very expensive to run large scale tests [2] on cars in a lab or outside environment, making the use of sophisticated software tools seem like a very attractive option.

There are a number of large scale autonomous vehicle behaviour simulators available already. One of these is the Opal/Orchestra Simulation Suite [3], which promises a robust and flexible simulation platform that intends to remove the need for physical simulations. Opal is not the only mass-simulation program on the market and together they successfully allow for the development and testing of algorithms in a purely virtual space.

These programs have been fantastic in supporting the concept of autonomous vehicles for a long time now but their usefulness declines as we get closer to the mass deployment of personal autonomous cars [4] because there is a great need to run tests in less ideal environments.

Simulating inside a virtual space can give a good approximation but no matter how many variables are included in the process, a simulation will never be able to capture all of the necessary uncertainties. The transition from virtual testing to the real world must be taken carefully because massive amounts of testing should be done to ascertain the accuracy of these analytical techniques [5].

After reviewing the different simulation options on the market, it is easy to identify a lack of physical testbeds using non-ideal environments.

3.2 WORKING WITH UWA

In an attempt to combat this issue, the University of Western Australia has developed a miniature RC car platform that has already been successfully used in showcasing autonomous car behaviours to the students at open day [6]. It is expected that the platform will ignite a further interest into the development of autonomous car routing algorithms. With such a great platform already existing at the university, my aim is to increase the effectiveness of the platform as a study tool by upgrading its features and streamlining the agent creation and simulation launching processes.

The students who were working on the testbed last year suggested a few improvements that were vital to the success of the system moving forward [6]. Two of these suggested improvements (User interface & Percept Filtering) align with the research topic that I have chosen, giving me further confidence that my topic will greatly benefit the state of the art.

3.3 SIMILAR PROJECTS

While not extensive, there has been some work done along the same lines as the testbed that we will be working on.

Some work on machine learning was completed using a mini-car soccer testbed [7]. The cars were controlled by humans but the system contained a camera that took a series of images and tried to predict when a goal was going to be scored. This process of prediction by mixing computer vision with machine learning resulted in some interesting conclusions, but it showed that even with the most careful planning, unexpected events always occur. This strengthens the argument that we need to offer more than pure computer simulations.

There is also a scaled down testbed [8] that captures and models human behaviour as they use a remote to drive around a track. This project was successful in showing that human driving is a lot more complex than we think. Slowing down around bends, small hesitations before crossing pedestrian crossings are all things that humans do so easily but are difficult to correctly code into a machine. Once again this reinforces the need to simulate using non-ideal conditions – there's just no way of knowing how effective an algorithm is until it is put in an emergency.

Education has also been effected by this movement towards cars and testbeds. A study was done on a new way to attempt to teach children road safety rules using a large testbed [9] consisting of a track, some obstacles and a few toy cars. The aim was to present the children with a situation and teach them when the correct time to cross the road would be. The results were not very conclusive but it opens up a new possible use of the testbed we have available to us. We could start a simulation and have cars driving themselves around the track then ask the students to try to cross the road safely – it would be more effective than toy cars that we move with our hands.

Taking a look at similar projects being undertaken reinforced exactly how unique the testbed truly is. While it will never be as computationally powerful as a full software suite, I can see a great use for it as an educational tool that will be extended further every year with new ideas.

4. METHODOLOGY

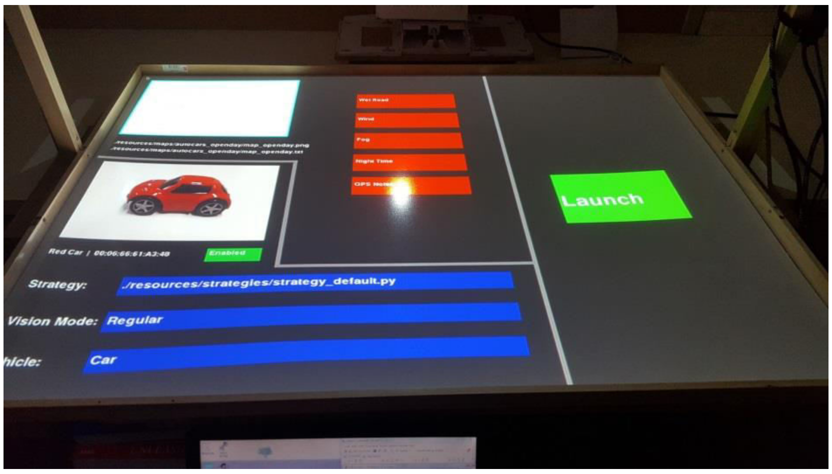
This section of the report will contain the reasoning behind the design decisions that I have made during this project. This will include decisions when creating the user-friendly interface (the GUI and the API), and the experimental decisions made when running specific traffic routines for use with the testbed.

4.1 DESIGNING THE GUI

The GUI is something that was not present when we received the testbed and it was the first task that I jumped into after rebuilding the structure of the control code with Michael Finn.

The main purpose behind the GUI was to allow users to simply walk into the testing room with their autonomous car algorithm loaded on their USB, plug it into the Raspberry Pi and begin testing right away. This was an attractive upgrade because the previous solution required the user to edit multiple configuration files to change the options of the simulation – not user friendly at all.

I went through 4 different iterations of the GUI before getting it to a state where everyone who was using it was satisfied with the flow and presentation of the data.



./resources/maps/autocars_openday/map_openday.png
./resources/maps/autocars_openday/map_openday.txt

Strategy: ./resources/strategies/strategy_default.py

Vision Mode: Regular

hicle: Car

Image 2: The Simulation Launcher GUI

Written in Python, the purpose of the GUI is to let the user set up the simulation by modifying the options available to them. Navigation of the interface is performed using an Xbox remote which was selected because it is a familiar device that contains more than enough input elements to navigate the blocky interface in multiple directions.

Each of the blocks in the interface contains data that will be sent to the control program once it is launched. These blocks can be 'toggled' either forwards or backwards to scroll between options. Note that there are 5 blocks located around the centre of the interface; these are legacy options that had planned functionality but due to a change in scope they were made redundant. They have been left in the interface as a possible future extension but at this point in time they do not serve any purpose.

The controls available for the Xbox remote in the current iteration of the GUI are as follows:

Xbox Button	Function
A	Navigate Down if possible
B	Navigate Right if possible
X	Navigate Left if possible
Y	Navigate Up if possible
Right Bumper	'Toggle' Forwards
Left Bumper	'Toggle' Backwards

Table 2: Xbox control for the GUI

Excluding the redundant blocks mentioned earlier, there are 7 interface blocks that offer functionality to the launcher.

Block Name	Location in Image 2	Function
Map Select	Top Left, shows a white rectangle.	Toggle this block to choose which map you would like to simulate on. The launcher auto-detects all

		maps located in the resources/maps folder on the pi.
Car Select	Middle Left, shows a red car.	Toggle this block to select a different car. All of the options below this block are car specific so each car can be modified individually for the simulation.
Car Enable	Just below Car Select, shows a green rectangle.	Determines whether or not the currently selected car is to be used in the simulation or not. Toggles between green (ON) and red (OFF).
Strategy	Bottom left, top of the blue rectangles.	Chooses which control code is applied to the currently selected car. The launcher auto-detects all control files located in the resources/strategies folder on the pi and toggling this block allows the user to choose between them.
Vision Mode	Bottom left, middle of the blue rectangles.	This is an additional simulation option that is intended to allow a simulation to limit the amount of information that an autonomous agent has available to it, but it has not been fully implemented at this time.
Vehicle	Bottom left, bottom of the blue rectangles.	Toggling this option will allow the user to choose between multiple vehicle types. Selecting a different vehicle type will have an impact on the top speeds and accelerations of the selected vehicle.
Launch	Right side, Green rectangle.	Toggling this option will launch the simulation with the selected map and car options.

Table 3: Interface Button Functions

I built the GUI alongside the reconstruction of the control code so as it became clear we needed another simulation option selectable by the user, I could integrate it into the GUI straight away. This led to the iterative process that was used to bring the launcher to the state it is now.

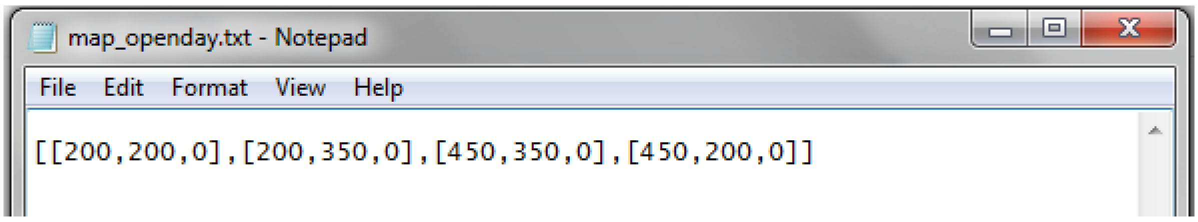
4.2 DESIGNING THE API

While the GUI allows a user to easily begin a simulation with the desired conditions, an arguably more important element of this project is the state of the API that a programmer will have available to them when creating their own autonomous car control algorithms.

While we rewrote most of the core control code for the simulation (explained earlier), we decided to use a similar waypoint functionality that was used in the project last year. That is – the car will attempt to drive around the map following a series of waypoints that are defined in advance by the user. This means that currently for a student to construct their own simulation, they must provide the testbed with:

- The agent strategy file(s) that they intend to test. Thanks to the work that Michael and I did on modularising the control structure of the testbed, these agent strategy files are each going to be independent Python scripts that utilise the API detailed in the later sections of this chapter of the report.
- A map file – this is a simple image of the map that they want to drive around. It will be projected onto the driving surface. Recommended format is a .png image with 640x480 dimensions.

- A map waypoint file – this is a text file containing an array of waypoints where each waypoint is defined by 3 values: [x location, y location, waypoint type]. The waypoint type was rarely used in my project however its inclusion allows for a greater complexity of simulation to be performed. The image below shows a default waypoint file that designates a rectangle for the cars to drive around.



```
[[200,200,0],[200,350,0],[450,350,0],[450,200,0]]
```

Image 3: Sample Waypoint File

When defining the API for writing the agent control scripts, I started at the lowest level possible and worked my way up to the moderate level that is available at the time of writing this document.

Control of the cars is restricted by the physical nature of the cars as well as the communication protocol that was established by ZenWheels. The work done by the team last year allows us to begin communications to the cars straight away, as we have not modified the bluetooth communications at all.

The raw commands that these cars accept are simple values for motor speed and turning angle. The cars also offer control of their horn, headlights and indicator lights. This leads to the natural lowest level API commands:

Command	Description
Vehicle.set_speed(speed)	Sends a raw value for the speed that ZenWheels car will attempt to drive at. Accepted values are between -64 and 63 but we have noticed that the speed of the cars caps at a level of around 30. The physical limitations of the cars do not let them exceed this. Negative values will make the car drive in reverse.
Vehicle.set_angle(angle)	Sends a raw value for the angle that the ZenWheels car will attempt to turn to. Accepted values are between -64 and 63 where an angle of 0 means driving straight ahead, negative angles mean turning left and positive angles mean turning right.
Vehicle.stop( )	Shorthand function for Vehicle.set_speed(0)
Vehicle.horn_on( )	Gives power to the vehicle's horn. A constant buzzing sound will eminate until Vehicle.horn_off( ) is called.
Vehicle.horn_off( )	Stops power to the vehicle's horn.
Vehicle.headlights_on( )	Gives power to the vehicle's lights. A constant light will shine from the front of the vehicle until Vehicle.headlights_off( ) is called.
Vehicle.headlights_off( )	Stops power to the vehicle's lights.
Vehicle.left_signal_on( )	Starts the left hand indicator flashing.
Vehicle.left_signal_off( )	Stops the left hand indicator flashing.
Vehicle.right_signal_on( )	Starts the right hand indicator flashing.
Vehicle.right_signal_off( )	Stops the right hand indicator flashing.

Table 4: Low level vehicle commands

Note that all of these commands simply involve sending a bluetooth packet to the car itself and letting it perform the function. There is no control performed on our end of these interactions. The horn and light functions are also purely cosmetic, while they were useful in producing a demonstration for the public on Open Day, they do not add any functional merit to the simulations.

After the raw vehicle commands were defined, it allowed me to move onto the more complex agent commands which required some extra processing to exhibit their intended behaviour.

The first step I made towards higher level commands was integrating the vehicle characteristics into the movement commands. This led me to the following new functions.

Command	Description
Agent.aim_speed(speed)	Gives the Agent a speed set point that it will approach over time. The time taken to reach this set point will depend on the Vehicle's acceleration value defined by the vehicle type chosen in the launcher. This function uses feedback from the camera tracker to determine how close it is to the real speed.
Agent.aim_angle(angle)	Gives the Agent an angle set point that it will approach over time. Currently there is no variability between the defined turning circles of vehicles but the angle supplied to this function is an absolute angle with 0 being North, 90 being East and so on…
Agent.get_waypoint_index( )	Returns the current waypoint that the agent is attempting to drive towards.
Agent.set_waypoint_index(wp)	Sets the current waypoint that the agent is attempting to drive towards. The user should be careful not to attempt to index a waypoint that was not defined in the waypoint file.
Agent.get_waypoint_type( )	Returns the type of the current waypoint that the agent is attempting to drive towards. This is used to change car behaviour as you approach certain locations around the track.

Table 5: Low-medium level agent commands

The next commands were produced because I was testing cars attempting to chase other cars instead of focusing on waypoint navigation.

Command	Description
Agent.get_vector_between_points(x1,y1,x2,y2)	Performs some simple trigonometry to determine the absolute distance and angle from (x1,y1) to (x2,y2). Will return the vector as an array [dist, theta] where theta is an angle degrees clockwise from north.
Agent.get_vector_to_waypoint( )	A shorthand method that uses Agent.get_vector_between_points but automatically uses the car's current location as (x1,y1) and the next waypoint's location as (x2,y2).

Table 6: Medium level agent commands

These are the highest level commands that are currently installed in the software but due to the modular nature of the control system, I expect more will be added in future projects involving the testbed.

4.3 PORTING EXISTING TRAFFIC ALGORITHMS FOR USE WITH THE TESTBED

With the GUI and the API completely defined, I decided to test the effectiveness of the testbed by taking some standard traffic routines and attempting to simulate them. Keeping in mind that my true goal is to upgrade the testbed for use as an educational tool, I chose three distinct situations that would each test a different element of simulation, resulting in a good overall assessment.

4.3.1 SITUATION: STANDARD DRIVING

One of the most important factors to the success of the testbed is for the cars to have the ability to successfully drive around a track. I used a track in the form of a simple loop as seen in the image below.

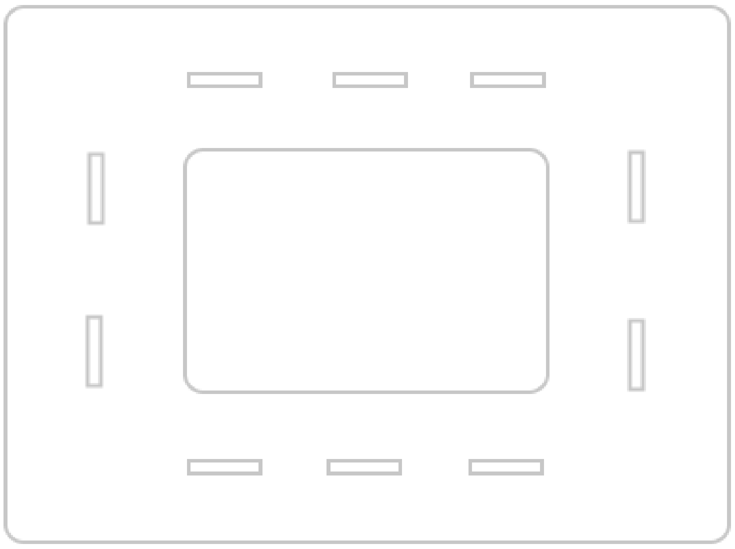


Image 4: Simple loop track, map_openday.png

I have had trouble with some of the cars throughout the course of this project so using this track as a baseline for a car's ability to make it all the way around will allow me to formally select the best cars for the simulations moving forward. Of course it is in the best interests of the university to ensure that they replace the cars that break so that future students have the maximum number of vehicles to test with.

The cars will be tested driving around this track with the same agent script controlling them so that the only variability lies with the cars themselves. The logic behind this default agent script is as follows:

- Drive towards next waypoint, using feedback to attempt to stay on target.
- If within 150 units of the waypoint, slow down.
- If within 50 units of the waypoint, mark it as visited and move on towards the next one.

While the cars are being tested, I tested them one at a time to ensure that there were no unfair interactions and each car was judged purely on its own merit. I will let each car attempt to drive around this track and record whether they were successful or not.

4.3.2 SITUATION: 2-WAY INTERSECTION

After the cars have passed the baseline situation, the two-way intersection map is their next challenge. This situation requires a higher level of control than the baseline loop because there is an area where a collision can potentially occur. This is seen in the image of the map below.

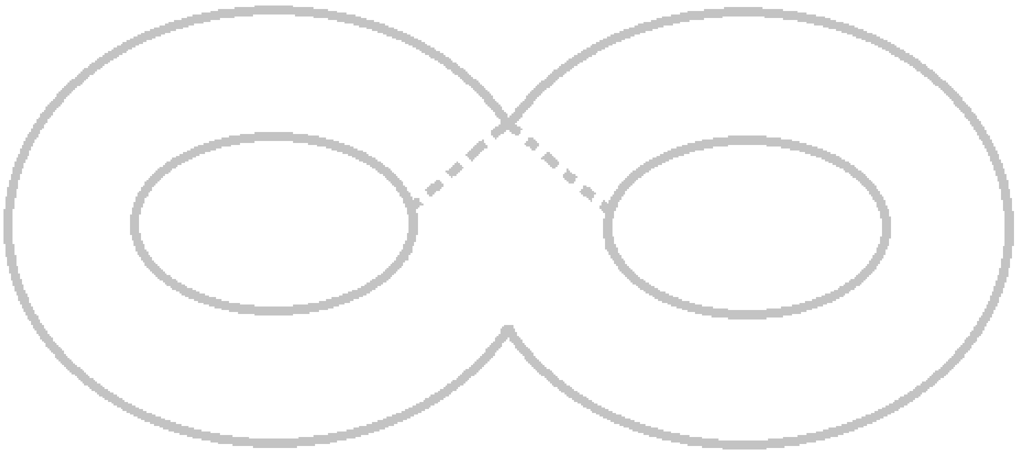


Image 5: A more complex track involving a two-way intersection. map_fig8_bare.png

This map was designed for multiple cars to navigate it at the same time, but at different speeds. This means that there were times when one car needed to wait for the other car to pass through the middle before it continued on.

This functionality was achieved by using the waypoint type variable made available by the format of the waypoint file. All of the waypoints were given type '0' apart from the one waypoint before the middle intersection, this was given a waypoint type '1'. When a car reached this waypoint with a type of '1', it would check the distance and angle to the other car, if it was too close, the car would stop and wait until the distance to the other car is large enough so that it is safe to move on.

I will test two cars driving around this track using different threshold values for the acceptable distance and angle that a car requires before it takes off through the intersection. I expect that the variability of the cars and the current tracking routine will require large thresholds to avoid collision.

4.3.3 SITUATION: PULLING AWAY FROM TRAFFIC LIGHTS

The final test was less of an examination of the potential of the API but more of an assessment of the current limitations of the system. I used one straight lane (seen in the image below) where the car would start from rest and attempt to drive straight to the end of the lane. This is the sort of behaviour seen by cars as they pull away from traffic lights.

Image 6: Single lane track, map_straight.png

I was worried about the robustness of the testbed and so this test was put in place to assess how well one of the cars can stick to its path. I will run two timers, one tracking the overall duration of the drive (until the car reaches the end of the road) and the other time I will start and stop, tracking the time the car remains in the lane. This will give me a percentage time that the cars spend in their own lane, which I will use as an effectiveness metric for the stability of the cars' steering.

5. DISCUSSION OF RESULTS

After spending two semesters building and iterating on the upgrades I have made to this testbed, it is finally time to sit back and assess my results. In this section of the report I will present any qualitative or quantitative findings I have made as well as suggest future work that can be performed to further enhance the testbed.

5.1 REVIEWING THE GUI

The GUI was built quite early and has not been modified recently at all. This has allowed everyone who uses the testbed to have a go at navigating the interface and give me feedback on their experience.

Overall, there were no major problems in starting a simulation with the desired options; however there was some recurring feedback which I have thought up solutions for, but have not implemented.

Feedback	Future Work
It feels weird to use ABXY for positional navigation.	Modify the GUI to accept directional arrow commands as well as ABXY for selecting different blocks in the interface.
I don't know which button to press?	Create a region in the GUI where it explains the controls.
I'm not sure what toggling this block does?	Create a region in the GUI that gives a detailed explanation of the purpose of the currently selected block.
How to I start the Launcher?	Force the Launcher to auto-start on booting up the Raspberry Pi.
Needing to copy my scripts to the Pi to use them is quite annoying.	Enable the launcher to also search through any USB drives for maps and agent scripts.

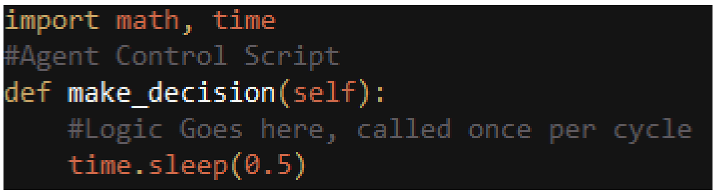
Table 7: Feedback on the GUI.

The successful implementation of the GUI satisfies **Objective 3** of my scope as discussed in section 2.2 of this document.

5.2 REVIEWING THE API

Unlike the GUI where I had lots of people come through and play with the Xbox controller to navigate the interface, there were no external people who came in to write up an agent control script.

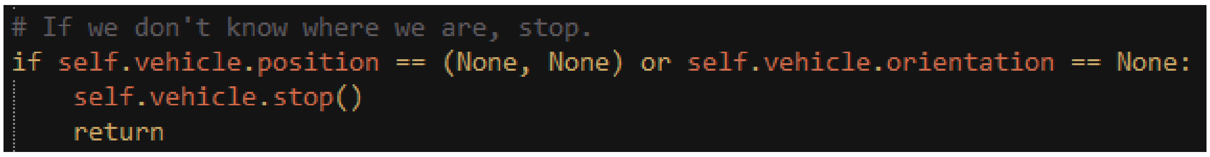
These control scripts are single python files, which need to contain a function called 'make_decision'. I found that the math and time libraries were especially useful so would recommend that they are imported into any future scripts. Limiting the calling of this threaded function with a 0.5s sleep also proved very useful because the current tracking algorithm uses the differential between current position and last position to determine orientation and if the function is called too quickly, the orientation calculation will not act as expected. Below is a sample skeleton for an agent control script.



```python
import math, time
#Agent Control Script
def make_decision(self):
    #Logic Goes here, called once per cycle
    time.sleep(0.5)
```

Image 7: Agent Script Skeleton.

Since there is only a mild level of error checking in the main control code located on the Pi, it is currently up to the agent to ensure that the data it receives is valid. Below is an important error check that should be included in every script.



```python
# If we don't know where we are, stop.
if self.vehicle.position == (None, None) or self.vehicle.orientation == None:
    self.vehicle.stop()
    return
```

Image 8: Checking against invalid location values.

I could have built more and more functionality into the API but time constraints required me to limit my work to the aim of this research project which was the upgrade of the testbed for use as an educational tool and with the get_vector function I have managed to implement some quite complex car behaviour, so I decided not to go any further.

Another constraint was the camera software that I am currently using; it is a simple hue detector that does not do much past locating blobs of the right colour. A lot of the deviation in car behaviour arises from the inconsistency of this tracker and so more functionality will be possible when Michael Finn finishes his research on superior tracking solutions and integrates it into the testbed.

The successful implementation of the API satisfies **Objective 2** of my scope as discussed in section 2.2 of this document.

5.3 REVIEWING THE EFFECTIVENESS OF THE PORTED ALGORITHMS

As described earlier, there are three distinct situations that I decided to simulate, each offering a different insight into the effectiveness of the testbed.

5.3.1 SITUATION: STANDARD DRIVING

The first task I had the cars complete was a simple rectangular loop of the driving surface. This is a simulation of regular driving conditions where there are no extra factors to consider. The techniques explored here are used as the base for further investigation because of the simplicity of the test.

I found that the cars were generally not very reliable. Only 3 of them successfully completed an autonomous track. Two of them had battery issues while the other two had tracking issues. The results of the baseline laps around the rectangular track are shown below.

Car Code	Car Colour	Loop 1	Loop 2	Loop 3	Average
48	Red	36	36	35	36
CD	Blue	36	35	35	35
0A	Pink	41	56	48	48

Table 8: Successful car loop times (Map_Openday).

Car Code	Car Colour	Notes
F7	Yellow	Tracking Failure
A5	Black	Tracking Failure
2D	Orange	Battery Failure
3D	Green	Battery Failure

Table 9: Unsuccessful cars.

From the experiment I can see that the tracker is the primary source of variability. The pink car finished the loop but took a few detours because the tracker bugged out in small areas. When the tracker works effectively (while tracking the red and blue cars), you can see the times behave quite consistently.

The yellow car was being tracked intermittently. It was able to drive half way around the loop, sometimes almost completing a full cycle but it was far too unreliable for me to continue using it in further experiments. This could be due to the lights in the room but the simple hue tracker that I am running was always going to be a problem.

The black car was not being tracked at all, the tracker couldn't locate it, and once again this is due to the simplicity of the tracker.

The orange and green cars won't even turn on, so I cannot quantify their effectiveness driving around the track.

I considered spending more time tweaking my basic hue tracker but since Michael Finn is working on a far more robust tracking solution for this testbed I didn't think it was necessary as 3 cars is plenty for running my future simulations.

Identification of the cars making their way around the track satisfies **Objective 1** of my scope as discussed in section 2.2 of this document.

5.3.2 SITUATION: 2-WAY INTERSECTION

A 2-way intersection is a common occurrence on regular roads and it is something that humans deal with quite easily.

Using the 'figure 8' track that was shown in section 4.3.2, I ran the red and the blue cars around the track. I ran this multiple times so that the cars would be forced to wait at the intersection as the other car passes through (as instructed by the strategy file).

As explained in the methodology section, I was interested in the different thresholds that would be required to allow the cars in this testbed to safely navigate the traffic hazard.

Parameters	Angle: 180	Angle: 120	Angle: 90
Distance: 150	NO CRASH	NO CRASH	CRASH
Distance: 100	NO CRASH	NO CRASH	CRASH
Distance: 50	CRASH	CRASH	CRASH

Table 10: Effect of different thresholds on car safety, map_fig8_bare.png

There are some details to note with these results, I ran multiple simulations (5-10 for each threshold) and if even a single one resulted in a crash then that's how I marked it. This is of particular effect in the 90 degree angle column, where there were multiple instances of clean runs but due to the inconsistencies in the tracker, the results were quite variable and occasionally resulted in a crash.

This means that using the current tracking software, in order to accurately avoid moving obstacles, a very large field of vision is required. This is quite a disappointing result as it massively limits the potential of the current system to perform complex manoeuvres however it further confirms the importance of integrating a more robust tracking solution.

5.3.3 SITUATION: PULLING AWAY FROM TRAFFIC LIGHTS

The final test was a test of the physical limits of the cars and in particular how straight they can drive. I recorded time spent in lane and calculated it as a percentage of total time in the test and the results were far lower than expected from a simulation testbed.

Car Code	Car Colour	% Time in lane
48	Red	61
CD	Blue	44

Table 11: Percentage time in lane for two cars. map_straight.png

These low results are a result of how wobbly the cars are as they navigate around the waypoints. I found lowering the speed of the cars allowed them to stay in lane for much longer but I am more interested in the functional performance of the testbed so that involves running the tests at the usual speeds.

The poor percentage of time spent in their lane can once again be attributed to the lacklustre tracking solution running in the system. With a more accurate tracker, there will be less incorrect compensation and the cars should be able to perform well at a higher speed.

CONCLUSION

I have extended the testbed and streamlined the process that a user must go through in order to turn an idea for an autonomous routing algorithm into a real simulation with this system.

I created an API that allows users to create functioning scripts to control the ZenWheels cars as they drive around a track. This API covers all of the low level functions available to us through direct interaction with the cars and then starts to abstract these functions using some extra control code into higher level functions. As stated in the Methodology section of the report, there is still large room for growing this API even further and empowering the future developers.

I created a GUI that allows users to easily start a simulation with their desired options. This GUI is navigated with an Xbox remote and has proven successful in its intended role as the launcher for the simulation. There was some constructive criticism (detailed in the discussion of results section) that opens the door for future enhancement of this interface as well.

I put the testbed through some hoops by implementing three basic traffic routing algorithms and observing how the testbed handled them with its limited functionality. I found that if I kept things slow then the performance was adequate but ramping up the speed drastically reduced its effectiveness.

ID	Description	Achieved?
1	Cars Drive around Simple Track	YES
2	Streamline Process for Creating new Agent Strategies	YES
3	Streamline Process for Loading new Agent Strategies	YES

Table 12: Achieved Project Objectives

I observed that the biggest factor limiting the testbed was the tracking solution supplied by the primitive camera routine I was using. This is only a temporary measure so I am looking forward to seeing the effect of implementing the result of Michael Finn's research project that involves selecting superior camera software.

ACKNOWLEDGEMENTS

I would like to thank both of my Supervisors for their continual support during this year. Both Tim and Mubashar were always available to answer any questions that came up during my investigations and offered so much insight and advice on how to move forward with my project.

Michael Finn and Zen Ly are fellow students both doing different projects but we are all using the testbed as a base for our research, it has been great working with them and I look forward to seeing what the combined result of all of our progress will look like.

REFERENCES

[1] Google News Archive, "Phantom Auto' will tour city", *The Milwaukee Sentinel*. 8 December 1926. Accessed on 10/4/2018.

[2] Martinez, M, "Detroit's autonomous car testing site nears opening"; *Automotive News, suppl. TRAVERSE CITY DAILY TUESDAY,* Detroit vol.24. iss.1. 2017.

[3] Opal + Orchestra Autonomous Vehicle Simulation Package; accessed on 6/4/2018 at: https://www.opal-rt.com/autonomous-vehicle/

[4] Rizzatti, L, "When to use Simulation, when to use Emulation"; *Electronic Products, 1 September 2014*. vol.59. no.9.

[5] Pettersson, I & Karlsson, "Setting the stage for autonomous cars: a pilot study of future autonomous driving experiences"; *IET Intelligent Transport Systems.* vol.9. no 7. pp 694-701. 2015.

[6] Barker, Ray, "Interactive Tabletop Simulation System for Testing and Visualisation of Autonomous Vehicle Algorithms"; *MPE Final Report (UWA).* pp 5, 32. 2017.

[7] Yoshinaga, M, Yukihiro N & Suzuki E, "Mini-Car-Soccer as a Testbed for Granular Computing", *Electrical and Computer Engineering, Yokohama National University, Japan.*

[8] Yam, Y & Tong, K, "A Scaled-Down Testbed for Human Driver Modelling and Evaluation"; *Proceedings of the 2003 IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*. pp.376-181.

[9] Fyhri, A, Bjørnskau, T & Ullebern, P, "Traffic Education for children with a tabletop model"; *Transportation Research Part F - Institute of Transport Economics, Norway*. vol.7. no.4. pp 197-207, 2004.