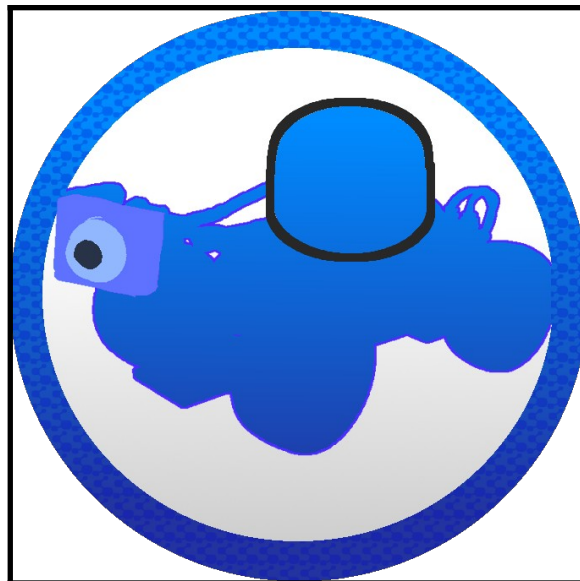


Project Report

Behaviour Tree PiCar-V



Name: Chi Huu Huynh

Student Number: C00261172

Submission Date: 19/04/2024

Lecturer Name: Oisin Cawley



Table of Contents

Introduction.....	.3
2. Project Description.....	.4
2.1 Application Frontend & Middleware.....	.4
2.3 Daemon Application.....	.5
3. Challenges.....	.6
3.1 Route Navigator.....	.6
3.2 Cartographer Library.....	.6
3.3 Missing Libraries for Lidar Scanner and TB6612 motor.....	.6
3.4 Connection with the Raspberry Pi.....	.7
3.5 Compilation of the Raspberry Pi executables.....	.7
3.6 Battery Issue.....	.7
3.7 Electron Lag Issue.....	.8
4. Learning Outcomes.....	.9
4.1 Technical.....	.9
4.1.1 XMake.....	.9
4.1.2 Doxygen and Github Actions.....	.9
4.1.3 Svelte.....	.9
4.1.4 Electron.....	.9
4.2 Personal.....	.10
5 Achieved & Did not Achieve.....	.11
6 What you would do differently if starting again.....	.11
7 System Architecture Diagram.....	.12
8 Screens.....	.13

8.1 Lidar 3d Projection.....	13
8.2 Behaviour Tree Codebox.....	14
8.3 Websocket Server.....	15
8.4 Raspberry Pi TUI.....	15
Conclusion.....	16
Acknowledgements.....	16
Plagiarism Declaration.....	17

Introduction

This project report introduces a project aimed at building an app to control a Sunfounder PiCar-V using Behaviour Trees. It focuses on integrating key components like the Camera and Lidar Scanner with the Raspberry Pi.

The app includes a backend on the Raspberry Pi, along with middleware and frontend parts in the Admin Panel.

This report explains how the app was designed and made, offering a clear understanding of its features and how it works.

2. Project Description

The aim of this project is to develop an application where you could control a Sunfounder PiCar-V using Behaviour Trees.

There are components such as the Camera and Lidar Scanner which were necessary to utilize on the Sunfounder PiCar-V's Raspberry Pi.

This full-stack application features a daemon backend on the Raspberry Pi, alongside middleware and frontend components housed within the Admin Panel.

2.1 Application Frontend & Middleware

The frontend user interface was built using Svelte on an Electron application. The Electron application served both as the Frontend and Middleware allowing it to communicate to the Daemon application via Websocket connections.

There are three screens which were on the Admin Panel:

- Home Screen – Consists of a Live Stream with the Camera and Lidar as it's source, Code box and a tab which shown the saved Behaviour Trees.
- Websocket Screen – Consists of a button to enable the Websocket Server, a list of connections, and a list of ip address that the raspberry pi may use.
- Behaviour Tree Validation Screen – Consists of a code box with a button to validate the Behaviour Tree.

In order for the admin to send Behaviour Trees to the Raspberry Pi, they have to write the Behaviour Tree XML on the home screen's code box.

2.3 Daemon Application

The main executable on the Raspberry Pi is a daemon controls the components of the Sunfounder PiCar-V, the Camera, and the Lidar Scanner.

The daemon executable automatically starts upon powering on the Raspberry Pi and waits for the WebSocket Server to become available. This setup enables easy accessibility and relieves the user from the need to log in to the Raspberry Pi each time they run this application.

Daemon executables are background executables which can start at boot and are used in Linux.

There is also another executable which I made at the start of the project, it utilizes a library called FTXUI which I used to develop a Terminal User Interface (TUI) to control the Raspberry Pi. Towards the end of the project, I found out that daemon executables exist and thus concluding my main focus on the TUI application because of the explanations in the above paragraph.

3. Challenges

3.1 Route Navigator

Initially at the start of the project, I wanted to make application which was able to control the Raspberry Pi by tracing a path on a HTML canvas and to use Cartographer for the display.

The application would feature rooms where users could purchase the use of the Sunfounder PiCar-V and they would be able to chat to each other.

However in the end, I decided to make an application which controlled the Raspberry Pi via Behaviour Trees because of these reasons:

- Rooms: The idea is very generic and would consume valuable time developing the Raspberry Pi.
- Cartographer: There was limited documentation and examples which shown how to use it.
- Path: Looking at the previous year project with the Sunfounder PiCar-V, they had a project based on paths and the Route Navigator would be too similar to it.

3.2 Cartographer Library

The Cartographer Library is built using C++ and CMake, it takes in points from a Lidar Scanner and develops a map using Simultaneous Localization and Mapping (SLAM).

At the start of the project, I planned to use the Cartographer library with XMake. But to make it compatible, I had to fix the Protobuf-cpp implementation in XMake for Windows, which took up a lot of time.

I also had to create the package lua script which allowed the Cartographer library to be used in XMake.

3.3 Missing Libraries for Lidar Scanner and TB6612 motor

Since I was using C++, there weren't many libraries which supported the Lidar Scanner and the TB6612 motor compared to Python.

For this, I leveraged the use of ChatGPT to help me develop these libraries and in the end, I was successful in creating these libraries.

3.4 Connection with the Raspberry Pi

Connecting to the Raspberry Pi while on the SETU Wifi was a tricky problem to solve. At the start, I connected to the Raspberry Pi using a HDMI cable to display it's current IP address on the SETU Wifi but every so often the IP address of it changes. This would slow the development of the Project if I haven't solved this problem.

To start, I used a mobile hotspot to connect both the Raspberry Pi and the development device which worked but I would have to use my mobile data for this.

So in the end, I used the Windows hotspot feature to open a hotspot for the Raspberry Pi to connect to. This allowed the Raspberry Pi and the development device to both have a static IP address when communicating to each other.

3.5 Compilation of the Raspberry Pi executables

When using C++, you must compile the code before use, unlike Python, which utilizes an interpreter. Compilation takes an enormous amount of memory which was not available on the Sunfounder PiCar-V's Raspberry Pi as it only had 2GB of memory with 1.7 GB being free.

To solve this, I decided to use another Raspberry Pi to compile the executables, this Raspberry Pi had 8GB of memory.

Initially I wanted to set up a Virtual Machine (VM) which may compile the executables but I decided against as it would be complex, time consuming and there was no guarantee that it would actually work.

3.6 Battery Issue

Towards the end of the project, I found a critical problem relating to the batteries of the Sunfounder PiCar-V. This was critical problem as it would limit my Final Project Demonstration to only around 30 seconds when connected to the WebSocket Server.

To help aid and fix this problem I decided to add a frames per second (FPS) limiter to the camera of the Raspberry Pi to limit the amount of captures per second. Before I added the FPS limiter, the camera was capturing 200 images a second which definitely contributed to the battery issue.

Another fix to this was to charge the batteries for longer as I was using the Sunfounder PiCar-V more often at the end of the project.

3.7 Electron Lag Issue

When running the Admin Panel for a prolonged amount of time while connected to the Raspberry Pi, there was lag while using the Code Box and interface. This lag lead to the Admin Panel being unusable and made it difficult to control the Raspberry Pi. There was also a large amount of memory being used which indicated a memory leak.

Since this lag issue could happen during the Final Project Demonstration, I was forced to improve the performance of the Admin Panel.

Before tackling this lag problem, I knew that the Admin Panel didn't lag when it wasn't connected to the Raspberry Pi so I immediately investigated the Livestream code on the home screen and found that there was unnecessary loops to which I removed / reworked.

Afterwards, the Admin Panel was using less CPU usage but the problem still remained until I decided to make the Admin Panel ignore the data from the Raspberry Pi while the window was hidden and this prevented unneeded updates to the Livestream.

Although a slight problem still remained to which the Admin Panel was using a lot of memory. To fix this, I used `URL . revokeObjectURL` after calling `URL . createObjectURL` to free up memory from the image blobs.

4. Learning Outcomes

4.1 Technical

4.1.1 XMake

I decided to use XMake as my C++ build system instead of CMake as it used Lua instead of a Domain Specific Language which CMake used. I had prior experience with XMake and Lua which made the choice very simple.

It featured a package manager similar to NPM, NodeJS's package manager and it's syntax is very simple.

While there weren't many packages that suited my specific project, I created the ones I needed.

4.1.2 Doxygen and Github Actions

I learned how to use Doxygen with Github Actions to automatically create a documentation website every time when I commit new code to the Github Repository.

4.1.3 Svelte

I learned how to use Svelte which is a frontend application alternative to React. It is a simpler framework compared to React and allows developers to define components with Javascript and HTML.

4.1.4 Electron

I learned how to develop an Electron desktop application using Svelte. Electron is a framework which allows developers to create cross platform applications and is very simple to use. The programming language of it is Javascript.

4.2 Personal

The learning outcome for me was that I had to learn when to decrease the scope of the project and when to entirely change the plan of the project to make it reasonable.

If I had stuck onto the original idea of the Route Navigator, the project would have been a repeat of the previous Sunfounder PiCar-V Path Following project, without much improvement.

5 Achieved / Did not Achieve

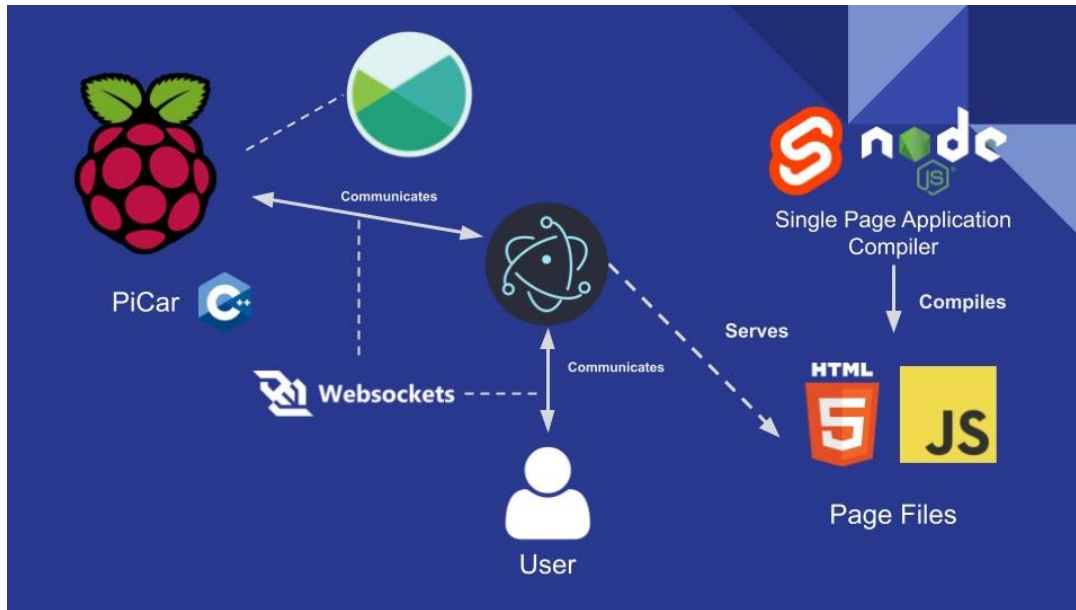
While I have not achieved what I have outlined in the initial documents, with the new functionality I believe that I have completed the Behaviour Tree PiCar-V application exceeding the initial proposed functionality.

6 What you would do differently if starting again

I would have listened to my supervisor's advice earlier which would have improved the Behaviour Tree PiCar-V functionality as I would have more time to work on it instead of working on the Route Navigator.

During the weekly meetings, he mentioned that Cartographer was very difficult to use and I should have known after looking for examples which barely existed.

7 System Architecture Diagram



C++ – Icon below Raspberry Pi Icon: Programming Language for the PiCar

Electron – The Icon in the Center: Allows communication between the User and PiCar

Xmake – Icon above Electron: Build System for C++

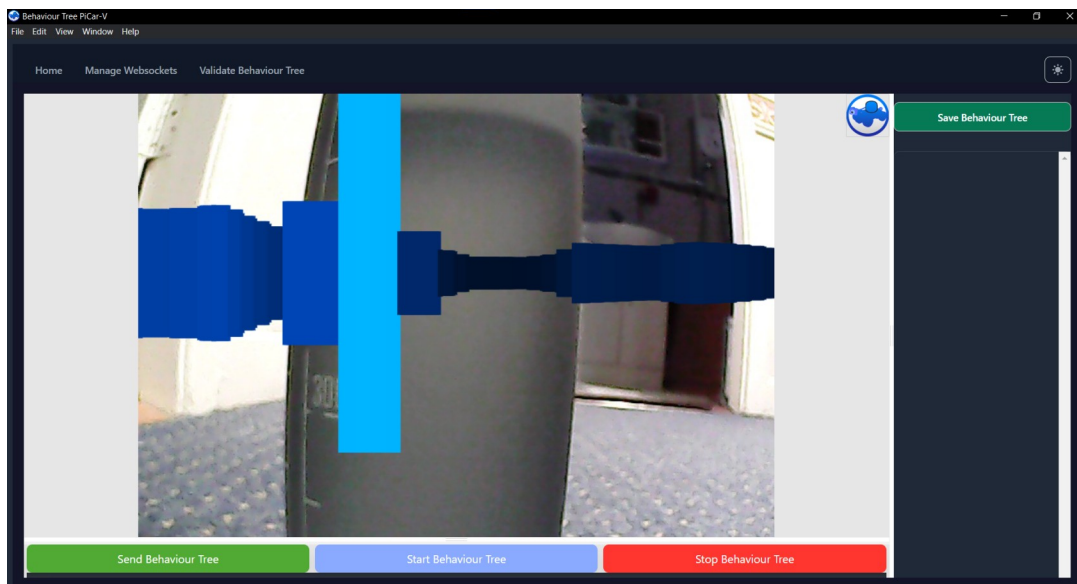
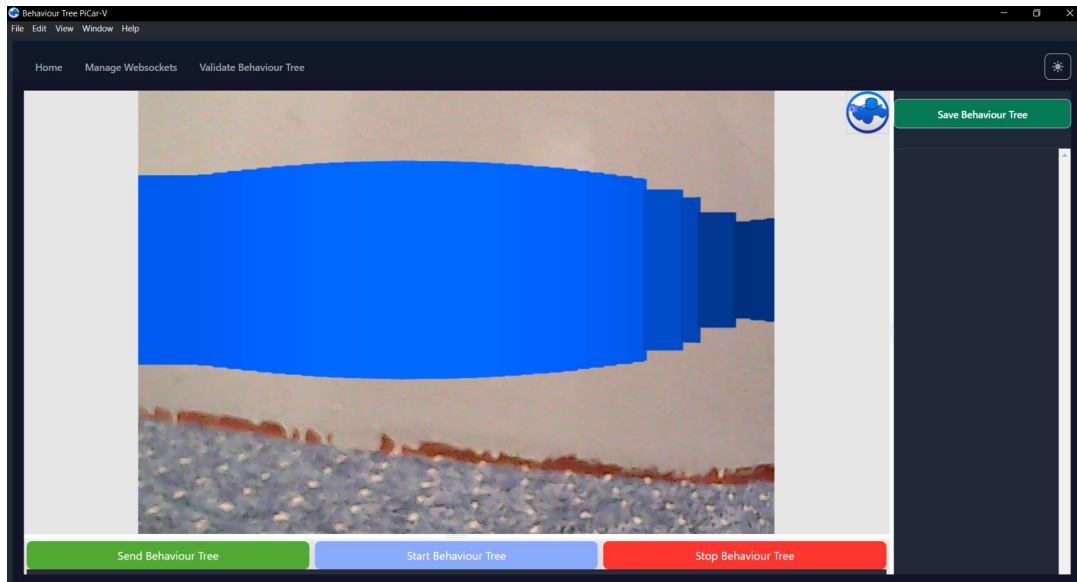
Javascript – Icon with JS: Programming Language for Electron's Backend code to communicate with the PiCar

Svelte – Icon beside NodeJS: UI Framework

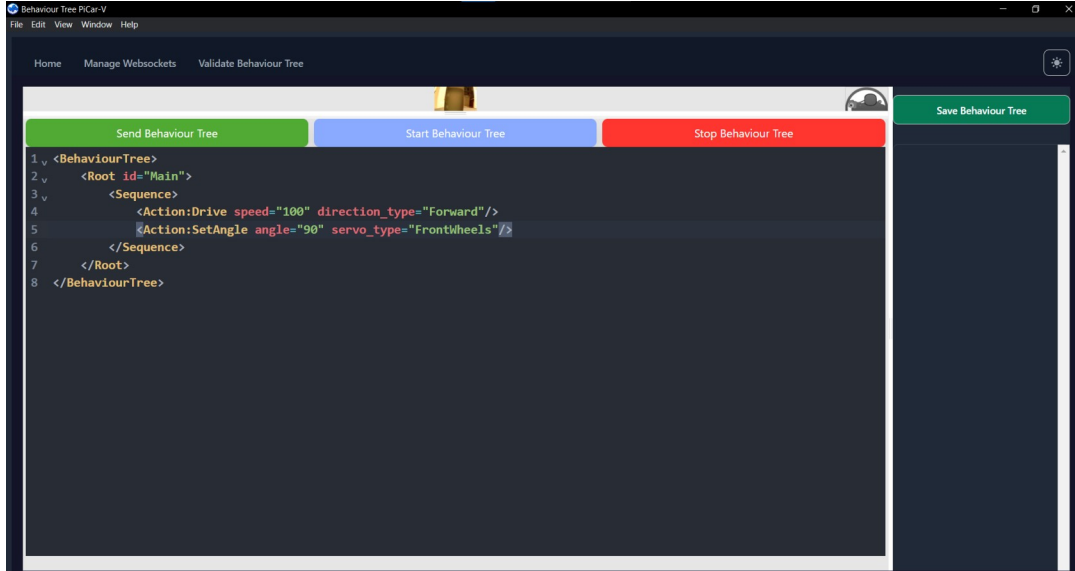
NodeJS – NodeJS Icon: Compiles the Svelte Application

8 Screens

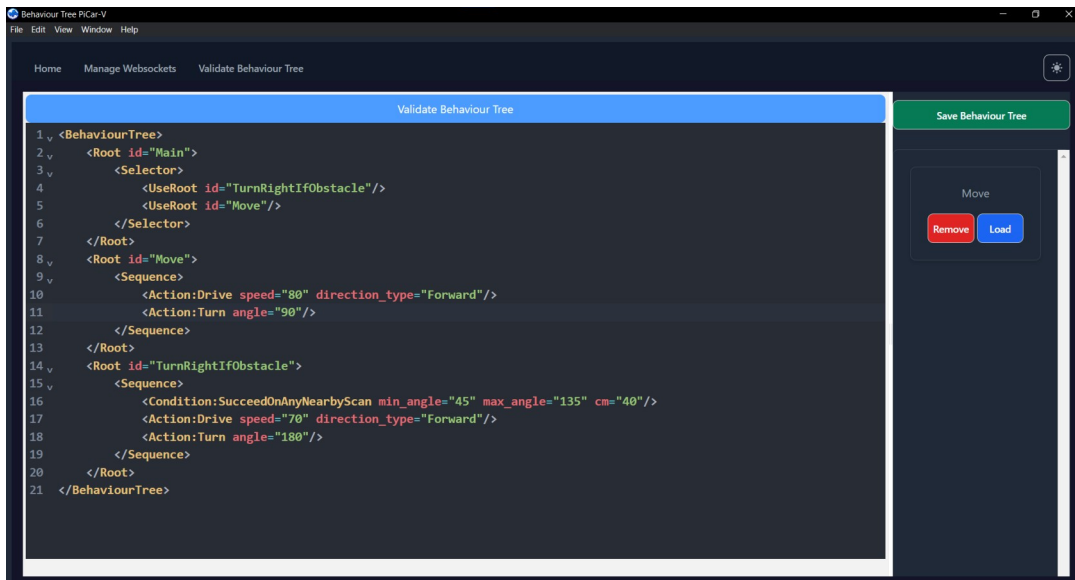
8.1 Lidar 3d Projection



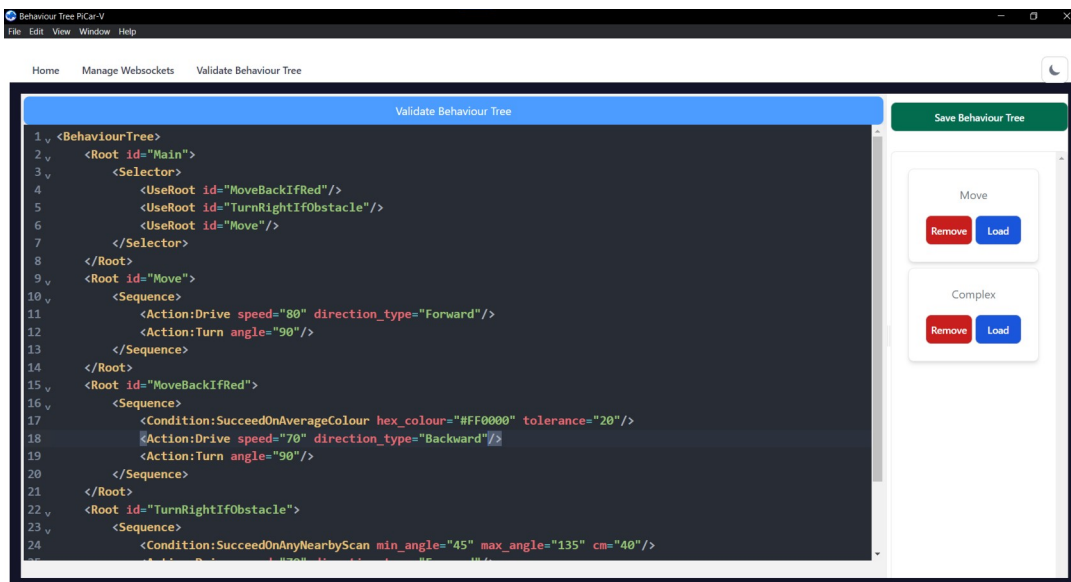
8.2 Behaviour Tree Codebox



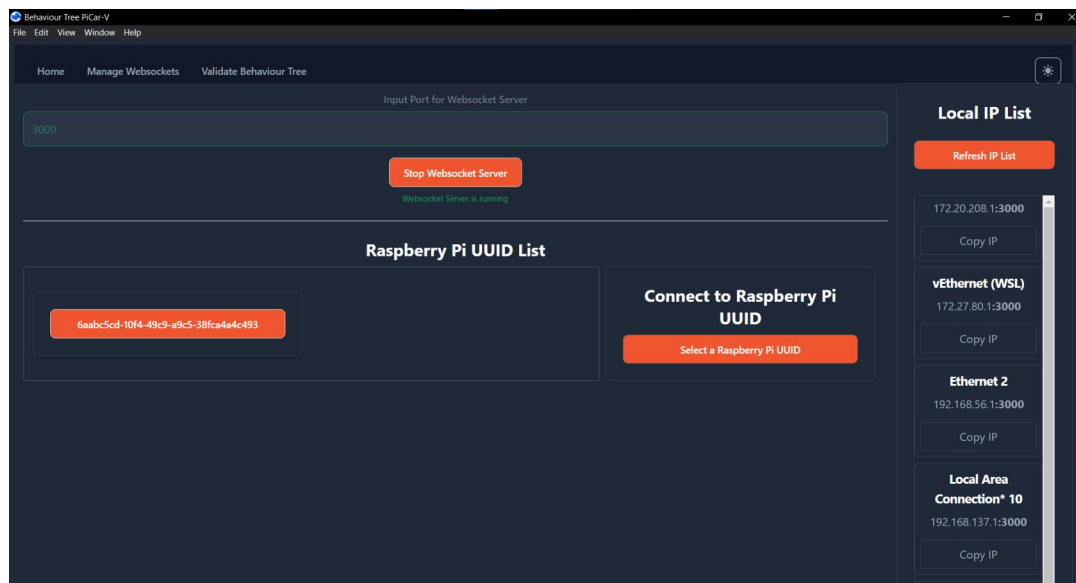
```
1 <BehaviourTree>
2 <Root id="Main">
3 <Sequence>
4 <Action:Drive speed="100" direction_type="Forward"/>
5 <Action:SetAngle angle="90" servo_type="FrontWheels"/>
6 </Sequence>
7 </Root>
8 </BehaviourTree>
```



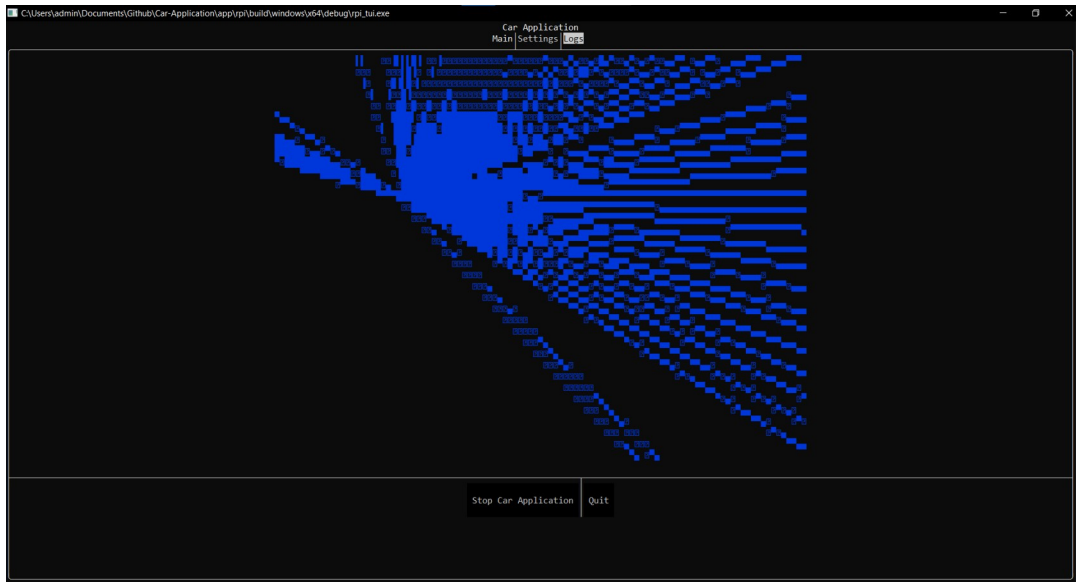
```
1 <BehaviourTree>
2 <Root id="Main">
3 <Selector>
4 <UseRoot id="TurnRightIfObstacle"/>
5 <UseRoot id="Move"/>
6 </Selector>
7 </Root>
8 <Root id="Move">
9 <Sequence>
10 <Action:Drive speed="80" direction_type="Forward"/>
11 <Action:Turn angle="90"/>
12 </Sequence>
13 </Root>
14 <Root id="TurnRightIfObstacle">
15 <Sequence>
16 <Condition:SucceedOnAnyNearbyScan min_angle="45" max_angle="135" cm="40"/>
17 <Action:Drive speed="70" direction_type="Forward"/>
18 <Action:Turn angle="180"/>
19 </Sequence>
20 </Root>
21 </BehaviourTree>
```



8.3 Websocket Server



8.4 Raspberry Pi TUI



Conclusion

In conclusion, I am thoroughly satisfied with my performance on this project. Successfully developing a fully functional Behaviour Tree application which controlled the Sunfounder PiCar-V's Raspberry Pi.

I have learned many new technologies including, Svelte, and Electron. And also further improved my skill of programming in C++.

Although there were a lot of issues when developing this project, I managed to overcome them.

Acknowledgements

I would like to thank Oisin Cawley for his support throughout the duration of the project. Our weekly meetings were useful, during which he offered insightful feedback and guidance. His dedication greatly contributed to the project's success, and I am very thankful for his commitment.

Finally, I want to thank all the SETU lecturers I've met who have taught me during the course.

Plagiarism Declaration



- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the university's regulations governing plagiarism constitutes a serious offence.

Student Name (Printed): Chi Huu Huynh

Student Number: C00261172

Student Name (Signed):