

Final Year Project Report

Full Unit – Interim Report

SELF-DRIVING F1TENTH VEHICLE USING PLANNING & LOCALISATION ALGORITHMS WITHIN ROS– Interim Report

Kacper Buksa

A report submitted in part fulfilment of the degree of

**BSc (Hons) in Computer Science (Software
Engineering)**

Supervisor: Francisco Ferreira Ruiz



Department of Computer Science
Royal Holloway, University of London

Table of Contents

Chapter 1:	Aims.....	3
1.1	Testing.....	3
Chapter 2:	Objectives	4
Chapter 3:	Introduction.....	5
3.1	F1Tenth Simulator & Algorithms	5
3.2	Robotics Operating System (ROS)	7
3.3	F1Tenth Hardware	7
Chapter 4:	Installation and Configuration of ROS and the F1Tenth Simulator.....	8
4.1	Installation of Ubuntu Operating System.....	8
4.1.1	Installing Oracle VM VirtualBox	8
4.1.2	Ubuntu Installation – Version 18.04 LTS (Bionic Beaver)	8
4.1.3	Ubuntu Virtual Machine Set-up on VirtualBox	8
4.2	Installation & Configuration of ROS Environment	9
4.3	Installation of F1Tenth Simulator	11
Chapter 5:	F1Tenth car hardware, sensors, and communication.....	13
5.1	Introduction.....	13
5.2	F1Tenth Hardware	13
5.2.1	Traxxas Slash 4x4 Premium Chassis	13
5.2.2	High-capacity GTR Shocks (suspension)	14
5.2.3	Traxxas Velineon Power System	14
5.2.4	LiPo batteries & Charger	14
5.2.5	VESC 6 Mk VI	14
5.2.6	NVIDIA Jetson Xavier NX.....	15
5.3	Ranging Sensors – Lidar & radar.....	15
5.3.1	Hokuyo UST-10LX Scanning Laser Rangefinder	15
5.4	Hardware Communication with ROS	16
5.5	F1Tenth Car Comparisons to Simulator.....	17
Chapter 6:	Proof of Concept Programs.....	18
6.1	Message Passing.....	18
6.1.1	Nodes & Topics	18
6.1.2	Publisher	18
6.1.3	Subscriber	19
6.2	ROS Turtlesim automatic navigation.....	20
Chapter 7:	References & Citations	21
Chapter 8:	Diary	22

Chapter 1: Aims

This project requires installing specific software to create any proof-of-concept programs and show all work in action. Therefore, a thorough guide to show how to install, configure and use all the tools will have to be created. From installation of the simulator and ROS to showing how to use the ROS environment. This is in line with the deliverables:

- Early Deliverable 1 – “Familiarity with Robot Operating System (ROS) environment and F1tenth simulator. Demonstrate manual control of the simulated car”
- Early deliverable 3 – “Report on installation and configuration of ROS and the F1tenth simulator”

It is important to discuss how the physical car’s hardware works to understand how it may be shown within the simulator. The simulator should work in a similar manner to the actual car, with detection, driving capabilities and data processing. This is part of the deliverable:

- Early Deliverable 2 – “Report on F1tenth car hardware, sensors and communication.”

Localisation and planning algorithms such as Monte-Carlo Localisation, Bug Algorithm (BA) and Ackermann Steering Geometry are used by the robot car to attempt any form of automatic navigation and automatic steering across the map. Monte-Carlo’s allows the robot to identify its current location on the map and its current heading while BA would make it possible for the car to traverse an unknown map using a wall-following method to reach the desired objectives.

These algorithms are one of my main points of investigation within this project, with being stated within my deliverables:

- Early Deliverable 4 – “Implementation and evaluation of basic algorithms (PID control, wall-following, obstacle avoidance).”
- Final Deliverable 1 – “Implementation and evaluation of at least one localization algorithm of choice.”
- Final Deliverable 2 – “Implementation and evaluation of at least one planning algorithm of choice.”

1.1 Testing

Since producing this project will require implementing software engineering components, as my later deliverables will be larger programs, where class systems may potentially be implemented. Testing will also have to be implemented within these programs, as programs will need to be automated with reaching the desired goal, these tasks will need to be tested to see if the automations work correctly. Since I am working entirely in Python, I will create unit testing to test the functions by creating scenarios with goals that must be reached.

Chapter 2: Objectives

1. To set up F1Tenth simulator. Set up any software environments, install packages.
2. To study ROS documentation, learn how to use it, follow F1Tenth lecture tutorials.
3. To create a report on installation and configuration of ROS and the simulator.
4. To research and report on the hardware of F1Tenth, its sensors and other components. To Explain how simulator programming helps in working with actual car.
5. To show how ROS works with message writing, publishing messages and listening to such messages.
6. To create a simple car track with vehicle to drive through successfully, with user assistance
7. To research self-driving components on ROS. Report on algorithms implemented so far. Localisation algorithms researched and reported on how it can be used in project.
8. To further research on track learning technology, identify use of car sensor to identify track boundaries.
9. To implement self-driving capabilities, staying within the walls of the track
10. To create scenario where user assisted car drives through unknown track, experiment track learning technology, with car storing processed track data

Chapter 3: Introduction

In this project, I will be using the F1Tenth simulator to create a program that allows a driverless car to navigate through a virtual map without any human assistance. This allows me to simulate a scenario that could then further be developed for use in fast time trials and racing, since this is what the F1Tenth car was created for. By creating this concept, it helps prove that autonomous racing is a safer and more efficient form of racing. Alongside autonomous driving, I will also implement track learning, where the car has to navigate through an unknown terrain, like rally car racing, where the drivers tend to learn the track on the go. The simulator should be able to provide all the necessary frameworks and tools to accomplish this project.

Throughout the history of car racing, humans driving cars has been the only way of performing this sport. Accidents have always been a major problem, with over 90% of accidents caused by human error [1]. Some of these have been caused by drivers miscalculating their speed, poor assessment of the track, distractions, fatigue, health, and many other factors. If the driver is affected by external factors, it may potentially impact their performance, either making their track time longer, or cause them to create mistakes which can result in accidents.

Self-driving allows the potential elimination of human error since losing focus, fatigue and other 'human factors' do not affect machines. Self-driving also allows the car to be able to always drive through the track at its optimal performance, being able to calculate in the moment what would be the best approach for a certain part of the track in real time without any internal factors affecting its decision. If by any chance there was some sort of accident that occurred during driving, there will not be any driver present that could potentially suffer from any harm, creating a safer environment in the racing industry.

The aspect of track learning would create the potential for the self-driving car to race in a condition where the track is currently unknown to the system. By studying the path, it 'senses' in front of itself using sensors to identify what its surroundings are, the car will be able to make real-time decisions on its speed and turns it needs to take. With this technology, the car can then be taken into the real world, where these skills can be transferred to traversing unknown paths and tracks with some ease.

3.1 F1Tenth Simulator & Algorithms

The F1Tenth Simulator is one possible way of testing such ideas. This software allows the testing of wall-following capabilities by programming the car to sense walls and drive within their boundaries. By doing so, the car will not need human interaction to drive it, since it can create its own path using the environment it is in. By using odometry and camera, the car can view what is in front of it, being able to track the current environment it is at. Once it tracks the current path, it can store it and overtime, once it ends driving, the car would be able to recreate the whole path in its storage, even if it has never seen it beforehand [2].

ROS F1/10 Simulator technology has a variety of frameworks that give the capabilities of creating such system that the car can drive automatically and learn the track using its sensor libraries. You can program a track that the car attempts to stay within and drive to specific locations through different localisation and planning algorithms.

There are many localisation algorithms that can be used as a solution for the robot. One algorithm could be the Monte-Carlo localisation algorithm, which is specific for robots to calculate their

coordinates in the environment, with taking the robot's odometry (robot's current location and its current heading direction), the map's environment (known landmarks on the map and the robot's distance to them) and sensor data, which is used to gather the distances, [3] pg. 162.

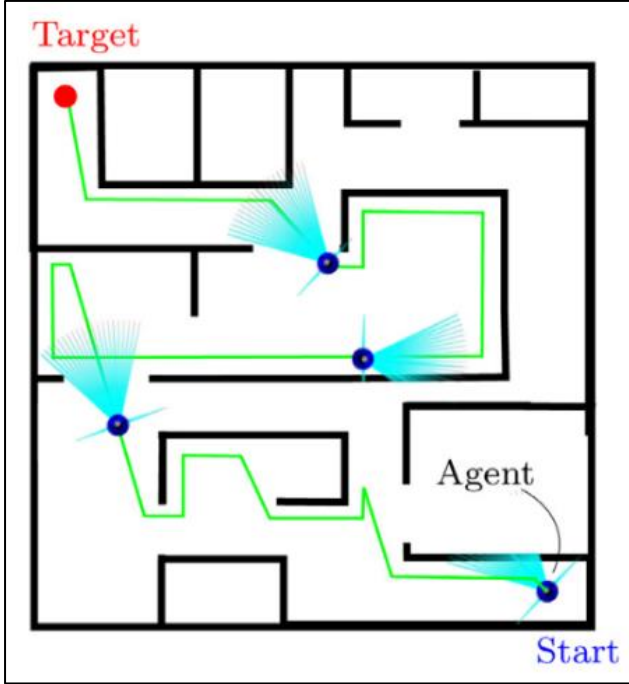


Figure 1 - Bug algorithm behaviour performed, [4] pg. 1

The simulator also requires a use of planning algorithms to be used for the robot to reach its destination from its starting point. One of these algorithms is called the Bug Algorithm. This algorithm can be implemented on a map that is unknown to the robot, where it uses its starting point and compares it to the target point. The car will then travel in the target's general direction, while following any walls on the way (Figure 1). This way the car learns the path while also reaching its destination without knowing the whole map's layout.

Another planning algorithm that plays a part in the robot's steering is the Ackermann Steering Geometry, which is used in ROS as a series of dependencies which allows robots with wheels to turn realistically. In the simulator, the car's wheels would be connected via trapezoid linkage, labelled ψ (ABCD in Figure 2). The wheels would turn

right once all wheels are faced in the direction of centre O , with left wheel turned θ_1 and right wheel turned θ_2 . The equation for BC (l_{20}) when turning angle is zero can be seen in Equation 1.

$$l_{20} = l_4 - 2l_1 \cos \psi$$

Equation 1 - Length of link BC when turning angle equals zero

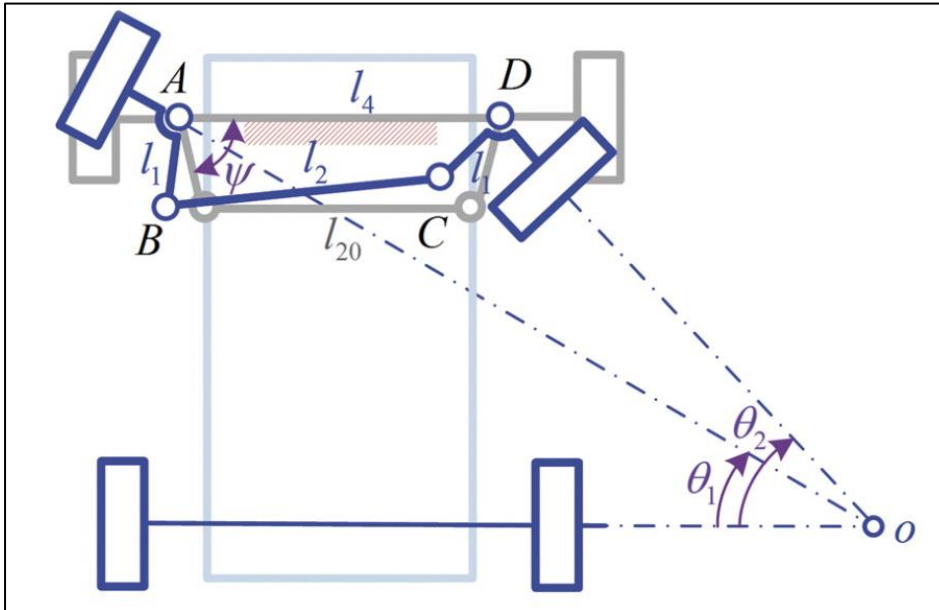


Figure 2 - Ideal Ackermann turning geometry with 4 bar linkages [7]

3.2 Robotics Operating System (ROS)

Since ROS is a new concept to me, there are a wide variety of documentations and tutorials online to swiftly help me understand the technology to be used within this project. The F1Tenth website [2] supports set up guides, software optimisations and lecture tutorials for understanding ROS. By using these resources that are tailored to the F1Tenth simulator, the learning curve will become much more manageable.

3.3 F1Tenth Hardware

Although this project mainly focuses on the simulator, the F1Tenth hardware can be used to test the car's capabilities without any risk of damaging the physical car, instead, creating a virtual environment first. After creating this scenario, these components can then be transferred to the physical machine itself to perform the tasks in the physical environment, to see how it performs in a real-life scenario. Within this project's early deliverables, a section is present, explaining how the simulator's programming can be transferrable to F1Tenth's hardware, how sensors and its other components will be used from the simulator's programming.

Chapter 4: Installation and Configuration of ROS and the F1Tenth Simulator

This section discusses the installation and configuration process to be able to use the ROS environment and implement it within the F1Tenth simulator that are used within this project. From installation of a virtual machine, use of Linux operating system, to installing the necessary packages and software to use the F1Tenth simulator.

4.1 Installation of Ubuntu Operating System

To use the ROS environment and F1Tenth simulator, the most stable way to use it is within the Ubuntu OS. This is because the simulator is only supported in Ubuntu at this moment in time, while ROS can only run natively in Linux [2]. Before installing the F1Tenth simulator, ROS is one of the prerequisites. Since the work on this report will be done locally, installing Ubuntu on a virtual machine would be most suitable.

4.1.1 Installing Oracle VM VirtualBox

The Virtual Machine used is the Oracle VM VirtualBox (version used for this project is 6.1.38), which can easily be installed from their official website, selecting either the current most stable version (as of 22/11/2022, version 7.0.x is the most recent version), or selecting older version of the software from their website¹. The installation type depends on the host OS. Having VirtualBox set-up is a pre-requisite for Windows hosts, with the installation software having easy to follow steps.

4.1.2 Ubuntu Installation – Version 18.04 LTS (Bionic Beaver)

To be able to emulate Ubuntu on the VM, it is required to install an ISO image file of the wanted OS as a pre-requisite, in our case Desktop image of Ubuntu Version 18.04 LTS (Bionic Beaver)², the use for this specific version is explained in the “Installation & Configuration of ROS Environment” section. This downloads all the required installation files necessary for the Ubuntu OS that will be used in the next section.

4.1.3 Ubuntu Virtual Machine Set-up on VirtualBox

Configuration of this program should be followed from the Oracle VirtualBox official documentation [5], with in-depth information regarding set-up of virtual machines after downloading the program. Once in the program, Press the ‘new’ button to create a virtual machine:

1. This will take you to a screen (Figure 3) where you enter the name of your VM, Type of OS and the OS version. In ‘Type’ select ‘Linux’ and ‘Version’ select ‘Ubuntu (64-bit)’, [5] section 1.8.1.
2. Next, select the amount of RAM memory you wish to use for the VM and press next, [5] section 1.8.3.
3. Afterwards, you need to create a new virtual hard disk for storage in the VM, select ‘Create a new hard disk now and select create.
4. In the next screen, select hard disk type ‘VDI (VirtualBox Disk Image)’.

¹ <https://www.virtualbox.org/wiki/Downloads> - VirtualBox download section.

² https://releases.ubuntu.com/18.04/?_ga=2.28871484.1441637138.1669127045-503508063.1662037585 – Ubuntu version 18.04 LTS Bionic Beaver ISO image file.

5. For storage on physical hard disk section, allow storage to be 'Dynamically Allocated'
6. Allow file location to remain default and select hard disk size (recommended 20GB), [5] section 1.8.4.
7. Once created, go into settings:
 - a. System → Processor, select Processor amount (4 Cores recommended)
 - b. Display → Screen, select Video Memory to max
8. The virtual machine is now launchable, select Start for further set up of the VM
9. VM opened (Figure 4), select icon circled and find the ISO Image File saved on PC and press start
10. The final step is to follow the installation of Ubuntu so it can now be used within the virtual machine

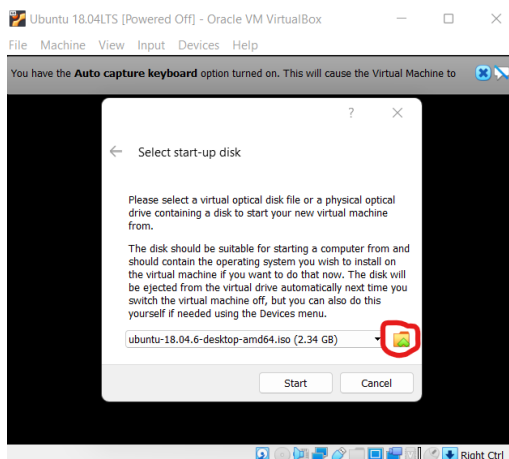


Figure 4 - New virtual machine start-up containing Ubuntu OS

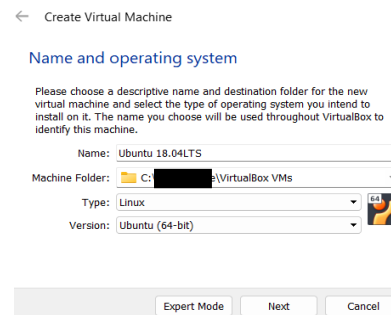


Figure 3 - Creation of new Virtual Machine

After following these steps, you now can set up the ROS environment and the F1Tenth simulator that will use ROS.

4.2 Installation & Configuration of ROS Environment

This section will be fully referenced from the ROS Melodic installation documentation [6].

After the successful launch of the virtual machine, with full support of Ubuntu OS, the next step is to download all the required ROS packages. The ROS packages have been built on the Debian package format system, which is supported on Linux. To complete this installation, it required to use the terminal built in to download and configure all necessary files.

Within the F1Tenth build webpage, it is stated that the version of ROS required is ROS Melodic, which is most stable within the Ubuntu 18.04 LTS Operating System, hence why this version was chosen [2].

The first step in set-up is to configure the Ubuntu system repositories to allow “restricted”, “universe” and “multiverse”. This can be done by opening Ubuntu Software & Updates and selecting all the required boxes (Figure 5), [6] section 1.1.

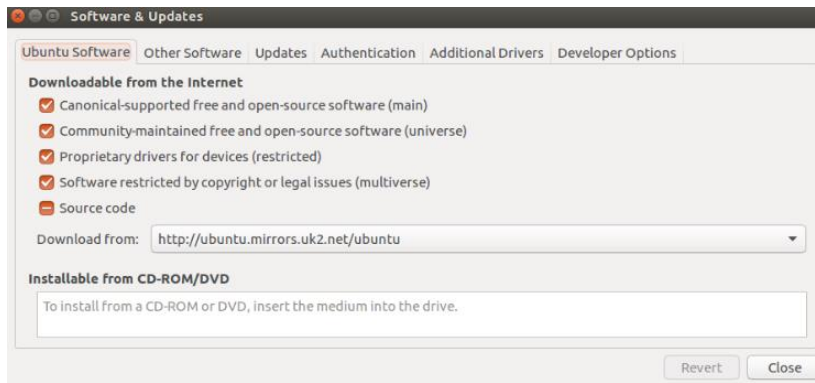


Figure 5 - Ubuntu repository configuration screen

The next step is to set-up the system to accept software from “packages.ros.org”. This sets up your system to allow these “.deb” packages to work on the machine, shown in [6] section 1.2. To do this, you must enter this command within the terminal:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Next, key set-up is needed and installation of curl – a tool for transferring data from server to host machine and vice versa, which is used to transfer the GitHub content for ROS and use it to set up as a key, [6] section 1.3.

```
sudo apt install curl # if you have not already installed curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc |
sudo apt-key add -
```

Now for installing ROS packages, you must first check that the package index of Debian is up to date by entering:

```
sudo apt update
```

Once the package index is up to date, the installation process can now begin. The recommended installation will be downloading the full desktop, which offers many different ROS tools and libraries, such as 2D/3D simulators and perception (which is one of the needed libraries for the F1Tenth simulator), shown in [6] section 1.4. This can be done by entering:

```
sudo apt install ros-melodic-desktop-full
```

Once installation is successful, you can check to find all the packages – in case any are missed out – installed by typing:

```
apt search ros-melodic
```

After installation, setting up so that the ROS environment variables are automatically added to the terminal when a new shell is launched, for easy use of ROS tools and software, [6] section 1.5. This is done by entering:

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

If not necessary, you can change the environment of the current shell each time by typing:

```
source /opt/ros/melodic/setup.bash
```

It is later recommended to install additional dependencies for managing ROS workspaces, any necessary tools and requirements that are separate from the installation of ROS Melodic, [6] section 1.6. This is done by:

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator  
python-wstool build-essential
```

Finally, to use many of ROS' tools, 'rosdep' needs to be initialised:

```
sudo apt install python-rosdep
```

To initialise rosdep:

```
sudo rosdep init  
rosdep update
```

With ROS installed, the final section is to install the simulator itself.

4.3 Installation of F1Tenth Simulator

With all the pre-requisites installed, the simulator will also be downloaded using terminal commands. Firstly, there are ROS dependencies that are required to run the simulator, all of which are from the ROS environment. This can be done through this command:

```
sudo apt-get install ros-melodic-tf2-geometry-msgs ros-melodic-ackermann-  
msgs ros-melodic-joy ros-melodic-map-server
```

Once completed, the next step is to install the simulator package. this is done by cloning the GitHub repository for this simulator onto a catkin workspace folder. A catkin folder is used to modify, build and install catkin packages, which is used within ROS, this folder needs to be created, then have the repository cloned into it.

```
mkdir ~/catkin_ws  
mkdir ~/catkin_ws/src  
cd ~/catkin_ws/src  
git clone https://github.com/fltenth/fltenth\_simulator.git
```

Then catkin_make command needs to be run to build the simulator:

```
cd ~/catkin_ws  
catkin_make  
source devel/setup.bash
```

The simulator is now ready to run through the command:

```
roslaunch fltenth_simulator simulator.launch
```

This command launches all the required tools needed for the simulation: roscore, the simulator, a preselected map, a model of the race car, and the joystick server [2]. Figure 6 shows the full simulator launched

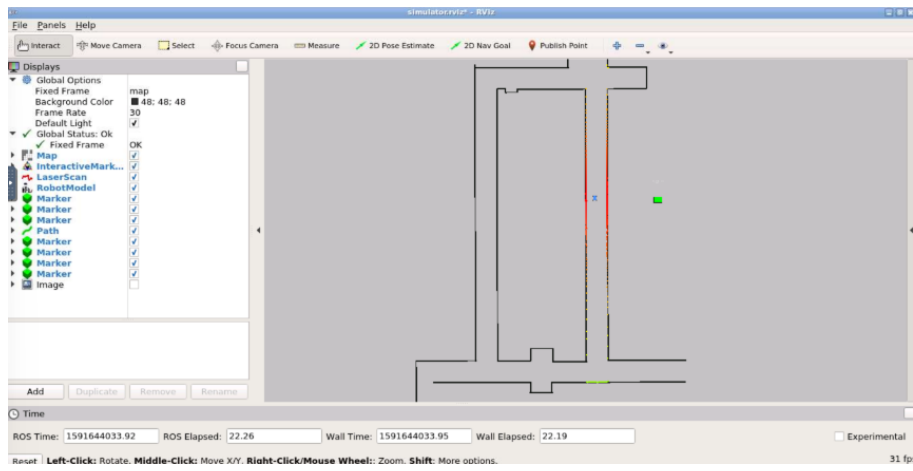


Figure 6 - F1Tenth simulator launched

Chapter 5: F1Tenth car hardware, sensors, and communication

5.1 Introduction

In this chapter, I will be discussing the hardware components of the F1Tenth, mainly their importance of how they work together to create an autonomous vehicle. It covers the basis of the robot's chassis, describing the different features that create the smooth manoeuvrability on different indoor surfaces. Afterwards, a discussion how the hardware's aspects may appear similar within the F1Tenth simulator, seeing how similar their performances can be transferred from testing in the virtual environment and the live one.

5.2 F1Tenth Hardware

Scroll right to see links	Qty	Cost per Unit
Chassis		
Traxxas Slash 4x4 Platinum Edition, 1/10 Scale Brushless Pro 4WD	1	\$269.95
2 Lipos and Charger Combo	1	\$199.95
Antenna Mount (3D print)	1	\$10.00
Platform Deck (laser-cut)	1	\$20.00
Fasteners		
M3 Socket Head Kit	1	\$17.98
M5 Socket Head Kit	1	\$8.99
6mm hex 14mm M3 standoffs	2	\$3.12
6mm hex 25mm M3 standoffs	4	\$4.56
6mm hex 45mm M3 standoffs	6	\$3.69
M2.5 x 0.45mm, 6mm long	4	\$6.87
M2.5 x 0.45mm, 10mm long FF Standoffs	4	\$0.62
Compute module		
Jetson Xavier NX	1	\$599.99
Micro SD Card 32 GB	1	\$7.50
NVMe SSD Card 250 Gb	1	\$75.00
Sensors		
Hokuyo 10LX	1	\$1,600.00
Electronics		
VESC 6 MkV	1	\$249.00
Joystick Opt 1: Logitech	1	\$39.00
Joystick Opt 2: DualShock 4 for PS4	1	\$69.99
LiPo safety bag like the Aketec Silver Large Size Lipo Battery Guard Sleeve/Bag for Charge & Storage.	2	\$9.99
Barrel Jack to Pigtail	1	\$4.33
12V 5A Power Adapter (optional)	1	\$13.99
Antenna	1	\$7.99
Intel RealSense D345i (optional)	1	\$179.00
Miscellaneous		
TRX to XT90 Adapter	1	\$11.99
Traxxas id charge lead adapter	1	\$15.99
Header Pins	1	\$7.99
VESC ppm cable	1	\$6.95
HDMI emulator	1	\$8.50
short (~1 ft) A USB-to-microUSB cable	1	\$7.98
Bullet Adapter 4mm Male 3.5mm Female	1	\$5.99
Total:		\$3,478.75

Figure 8 - Bill of Contents showing F1Tenth components. Main components highlighted [2]

The car has multiple components that are combined to create the final product, I will discuss the main ones that have a major impact on its performance (as highlighted in Figure 8).

5.2.1 Traxxas Slash 4x4 Premium Chassis

The car chassis is based off the Traxxas Slash 4x4 Premium Chassis (Figure 7), where all its components are attached onto its body [8]. This chassis was created to have a low centre of gravity by having the battery and electronics to be held low to position the weight of car low in the chassis. Because of the low centre of gravity, it is possible that the car's stability and ability to take on corners at an increased speed is improved. The chassis itself is ultra-smooth, which is documented by the Traxxas saying it reduces drag and improves ability to take on many types of terrain.

Included with the chassis, you have the GTR Shocks suspension system, a power system, wheels, and the body itself to hold all these components [8].

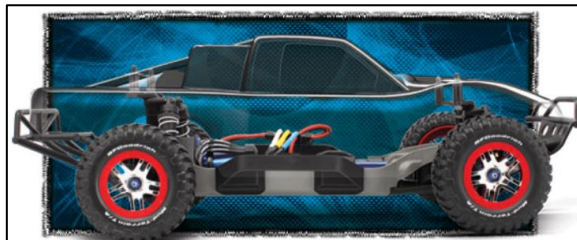


Figure 7 - Traxxas 4x4 Premium Chassis used for F1Tenth car – Traxxas [8]

5.2.2 High-capacity GTR Shocks (suspension)

[8] The shock system has a near frictionless piston travel due to its “PTFE-coating”, allowing for smooth suspension work when driving through ridged terrain. This also allows for smooth and comfortable suspension action for extended periods of time before potential replacing. The shocks have an option to also change spring reload and ride height of the car, which can be done simply by turning threaded spring collars. This allows for fast and easy adjustment to the suspension, making the F1Tenth versatile.

5.2.3 Traxxas Velineon Power System

The power system has been optimised for high-speed performance while maintaining smooth driving, to preserve control over the vehicle [8].

5.2.4 LiPo batteries & Charger

³These rechargeable Traxxas lithium polymer (LiPo) batteries (Figure 10) are the main source of power for the car to work, along with a charger for these batteries. Due to them being rechargeable, it allows the car to be a long-term investment, saving up on money as you do not have to restock on additional batteries. With the two ‘5000mAh’ batteries, the car has a possibility of reaching speeds of over 60mph (Figure 9).



Figure 10 - Traxxas EZ-Peak 3S "Completer Pack" Dual Multi-Chemistry Battery Charger w/Two Power Cell Batteries (5000mAh) - Traxxas

Slash 4X4 lets you choose the performance you want!
Select the right Traxxas iD battery to fit your needs from the options below

Speed	35+mph	40+mph*	60+mph*
Battery	2923X (1) 3000mAh NiMH	2843X (1) 5800mAh 2S LiPo	2872X (1) 5000mAh 3S LiPo
Pinion/Spur	13-T / 54-T	19-T / 54-T	19-T / 54-T
Skill Level	1	4	5

Larger pinion gear/smaller spur gear combinations are for high-speed running on hard, smooth surfaces only.
*With included optional gearing

Figure 9 - Traxxas EZ-Peak 3S "Completer Pack" Dual Multi-Chemistry Battery Charger w/Two Power Cell Batteries (5000mAh) – Traxxas

5.2.5 VESC 6 Mk VI

The TRAMPA VESC (Figure 11) allows you to control and adjust the speed and steering of the car. This is caused by a command (from the Jetson Xavier NX as discussed in next section) fed to the

VESC to increase or lower the voltage of the motor as necessary, which changes the propeller speed. The MK VI contains three phase shunts and an adjustable current/voltage filter, which allows a more precise and reliable detection of attached motors. This also allows motors to run smoothly and react linear to throttle input commands.

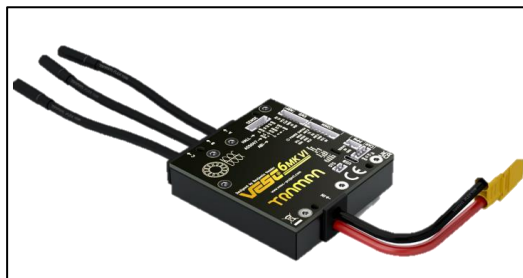


Figure 11 - VESC 6 Mk VI – TRAMPA.

³ <https://www.amainhobbies.com/traxxas-ezpeak-3s-completer-pack-dual-multichemistry-battery-charger-tra2990/p474595>

5.2.6 NVIDIA Jetson Xavier NX

The NVIDIA Jetson Xavier NX Developer Kit [9] (will be simplified to JXNX) is be the primary component of the F1Tenth's system, due to it having control over the VESC, which is a controller that oversees controlling and regulating the speed of the car's motor [2]. The JXNX will have the possibility to give out commands on the steering and speed control of the car due to having control over VESC, feeding it these commands. The JXNX also can receive information from the LIDAR sensor (Figure 13), which is one of the primary sources of information that dictate the regulation of speed.

The way that JXNX receives these commands is through the host computer that will be used to code and control the software of the F1Tenth, connecting remotely through SSH [2].

5.3 Ranging Sensors – Lidar & radar

Lidar sensors in the robotics field are used for many different reasons to help the robot in completing its set tasks. It can, for example, assist in navigation and obstacle avoidance. It is an active remote sensor, which judges an object's (or wall's) distance by measuring the time taken for the light beam to return to the sensor [8].

5.3.1 Hokuyo UST-10LX Scanning Laser Rangefinder

This Lidar light sensor⁴ (Figure 13) is used for the physical car, which can detect obstacles at a long distance of maximum being 30 metres, allowing the F1Tenth to conduct localisation of the car, with the scan being at a 270-degree angle (Figure 12). The sensor uses a light source as a way of obstacle detection, which in effect allows for a near immediate detection time, with scan speed measured to

Model Number	URG-04LX-UG01	URG-04LX	URG-04LX-F01	NEW UST-10LX	NEW UST-20LX
Appearance					
Light source	Semiconductor laser diode (FDA approval, Laser safety class 1)				
Scanning range	0.02 to 5.6m 240-degree		0.02 to 10m 270-degree		0.02 to 20m 270-degree
Measuring Accuracy	±/-30mm		±/-10mm		±/-40mm
Angular resolution	0.352 degrees (360/1,024)		0.25 degrees (360/1,400)		
Scanning frequency	10Hz (600rpm)		36Hz (2160rpm)		40Hz (2,400rpm)
Multi Echo Function	N/A		N/A		
Communication Protocol	SCIP 2.0	SCIP 1.1/2.0	SCIP 2.0	SCIP 2.2	
Communication and I/O Interface	USB1.1/2.0	USB1.1/2.0, RS-232C		Ethernet 100BASE-TX	
Power source	USB bus power	5VDC	12VDC	12V or 24VDC	
Power Consumption	0.5A or less		0.4A or less	0.15A or less (on 24VDC)	
Ambient Illuminance	—		10,000 lx or less		
Protective Structure	—		IP40		IP65
Weight	160g		185g		130g
Size (W x D x H) mm	50 x 50 x 70		60 x 75 x 60		50 x 50 x 70
Reference price					
Note	Inventoried item	Inventoried item	MTO item	It will be scheduled to go on sale in January 2014. Specifications are subject to change without notice.	

Figure 13 - Lidar UST-10LX Scanning Laser Rangefinder Specifications – [UST-10LX]

Figure 12 - Hokuyo UST-10LX Scanning Laser Rangefinder – UST-10LX.



⁴ <https://www.robotshop.com/en/hokuyo-ust-10lx-scanning-laser-rangefinder.html> - Light sensor details

5.4 Hardware Communication with ROS

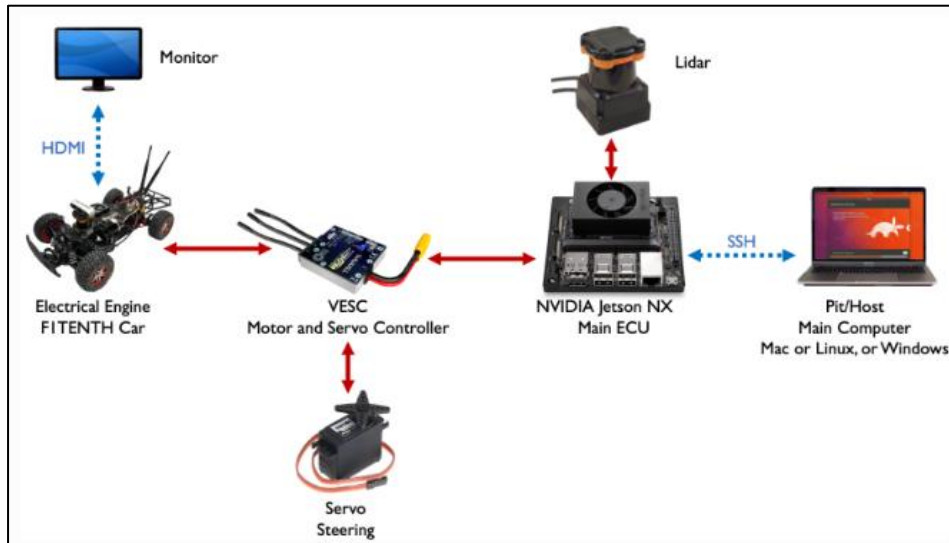


Figure 14 - Depiction of Data Flow of the F1Tenth Car [2]

The NVIDIA Jetson NX is the primary way that programs from the F1Tenth Autonomous Vehicle System are run for the car to operate and can communicate with the vehicle (as seen in Figure 14)⁵.

The Host laptop is the main computer that access the car and its data, also to start its software. This is the main way of connecting with the NVIDIA Jetson NX via SSH in a remote way (Figure 15), due to the Jetson having WiFi capabilities. This allows the user to create a way to use the car for different purposes. Figure 8 & 9 both show that a monitor, mouse, and keyboard can be used to connect to the car, but only if its stationary. With WiFi capabilities, the car can be connected to at any time without any wired connections needed. This creates a possibility to access the car while driving.

As discussed in the Jetson NX section, this component takes data input from the LIDAR sensor to assess the current environment it is driving in and use it to regulate speed by overseeing the VESC. This is where this data is used, regulating the motor speed and steering control.

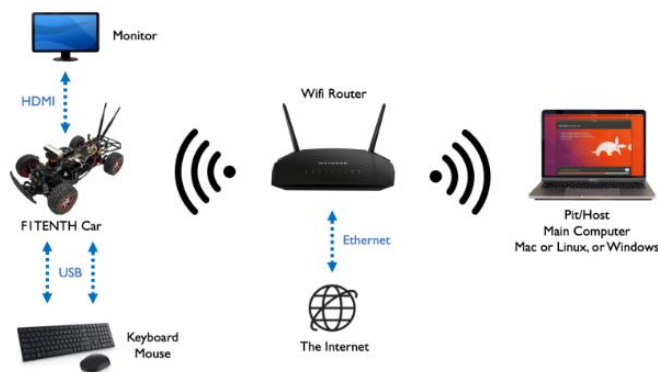


Figure 15 - Diagram showing possibility to connect to Jetson via WiFi [2]

⁵ https://f1tenth.readthedocs.io/en/stable/getting_started/software_setup/index.html - F1Tenth Configuration Site

5.5 F1Tenth Car Comparisons to Simulator

One similarity that the F1Tenth car has to the simulator is the way that obstacle detection works. With the car itself, the Lidar sensor (Figure 6) is used to detect obstacles via light sensor readings. This allows a precise detection of up to 30 metres from the sensor and near instant data collection.

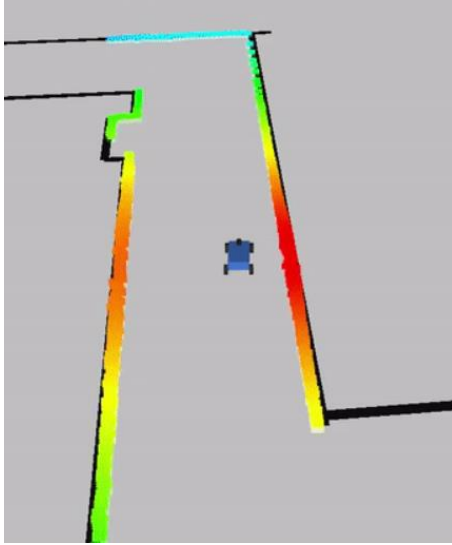


Figure 16 - F1Tenth Simulator heatmap of obstacles [2]

Meanwhile, the simulator uses a 'sensor' of its own, by tracking in a 270-degree cone (just like the actual Lidar sensor) all obstacles to a specific distance. The simulator produces constant updates of obstacle distance from the car every frame. To display the sensor distance, a heat map is used to show the distance of the object detected (Figure 16), the closer the obstacle is, the hotter the colour (e.g., red for very close, blue/purple for far).

PID control is an important aspect of the simulator, where the car's speed and steering are regulated by the constant PID loop. For steering, the car may be programmed to follow a wall at a specific distance, in this case PID function will calculate the distance error and correct it by getting closer or further from the wall. Overtime, the car will be able to navigate its way to be at that specific distance from the wall (Figure 17) with each cycle being a smaller error margin until there is no error in distance from the wall. The physical car

works the same as the simulator, with the process being same as Figure 17, with instead the VESC (Figure 11) overseeing this steering and speed adjustment of the car.

The difference in the simulator and real-life car is that there is no noticeable obstacle detection delay in the real car, since speed of light travel is near instant within the workspace the car will be tested in (usually corridors). Meanwhile, processing instructions within the simulator will be slower, albeit also a miniscule delay, since there will be multiple instruction lines that need to be processed about an obstacle. The number of instructions will be larger the further away the obstacle is. Moreover, for each obstacle, a line needs to be traced from the car to the obstacle, which will be updated constantly,

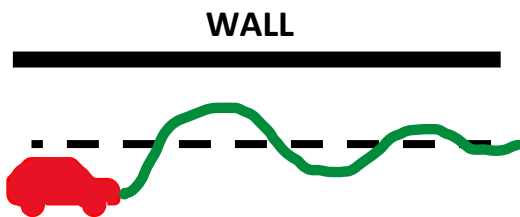


Figure 17 - Example of PID control and car navigating to follow wall

requiring even more instructions in need of processing. The simulator may be further delayed from program execution time if computer specifications are weak.

Chapter 6: Proof of Concept Programs

6.1 Message Passing

6.1.1 Nodes & Topics

[10] The primary way that you program within ROS is using nodes. These are processes that will perform defined basic tasks and computations within the systems, such as sending and receiving messages. Alongside these sets of nodes, there is a *Master Node* that will hold all the data about the other nodes within the ROS network.

Nodes will communicate by sending messages on topics, which they can become a part of. Each node which is part of some topic can receive all messages sent from other nodes that are part of this topic. These messages consist of a simple data structure, which can be an *int*, *string* or any other type.

The most common form of communication is by using a publisher-subscriber model. A topic *'foo'* can exist, where a publisher node will send a message to, and where a subscriber node can receive messages from *'foo'*, this is because of this topic channel being shared between these two nodes. All information that should be shared with other nodes of that topic will be stored in that message.

6.1.2 Publisher

Figure 18 shows an example of a publisher node called *'publisher_int16'*, which will send a series random numbers generated as messages to the *'num_output'* topic. This program will output *"Number is: [random number]"* message into the terminal in an endless loop until the program is terminated, as seen in Figure 19.

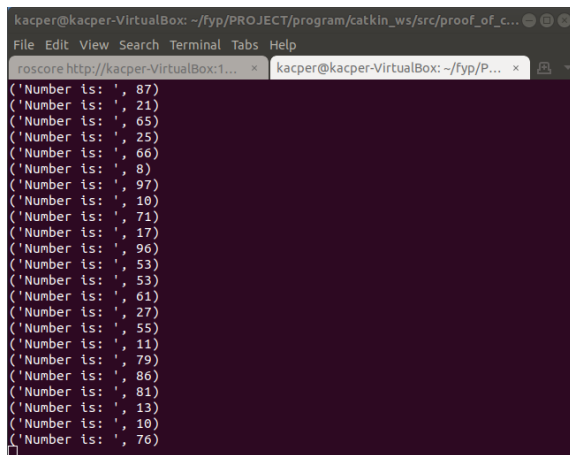
```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import Int16
import random

#Program inspired by ros.org tutorial http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

#Simple function to publish number to be read by subscriber
def num_publish():
    #Declaring node publishing to 'num_output' topic with message type of 16-bit int
    #queue size limits number of queued msgs if publishing/subscriber is slow
    publish_node = rospy.Publisher('num_output', Int16, queue_size=10)
    #Name of node
    rospy.init_node('publisher_int16')
    rate = rospy.Rate(10) # Loops msgs at 10hz p/s
    #Runs program until ctrl+C to shutdown command
    while not rospy.is_shutdown():
        #Creates a message outputting random number
        msg_int = random.randint(0,99)
        print("Number is: ", msg_int)
        publish_node.publish(msg_int)
        rate.sleep()

if __name__ == '__main__':
    try:
        num_publish()
    except rospy.ROSInterruptException:
        pass
```

Figure 18 - Publisher.py



```

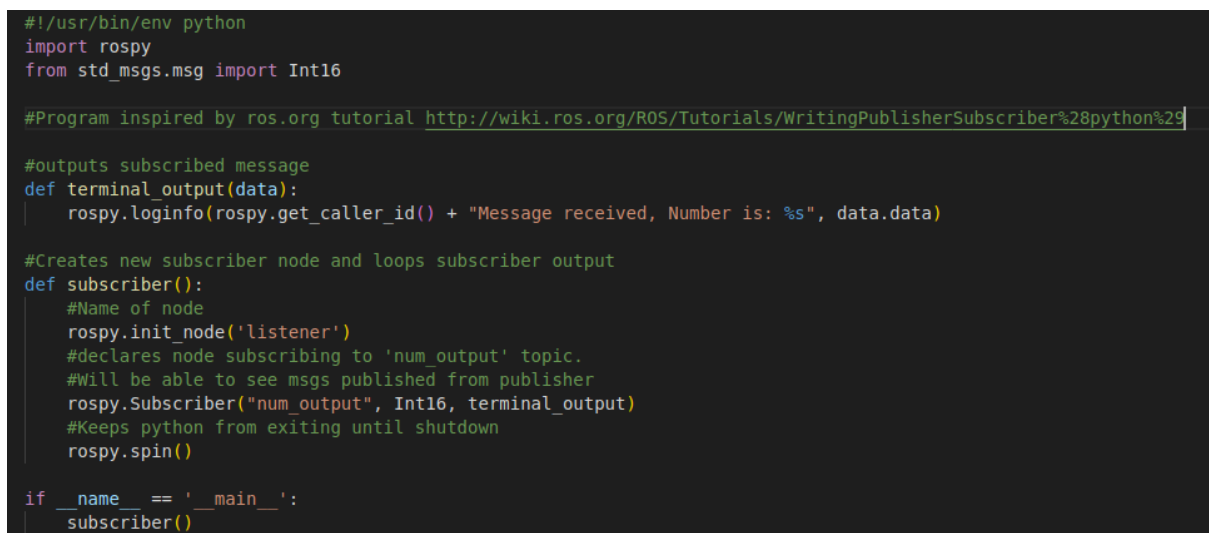
kacper@kacper-VirtualBox: ~/fyp/PROJECT/program/catkin_ws/src/proof_of_c...
File Edit View Search Terminal Tabs Help
foscure http://kacper-VirtualBox:1... x kacper@kacper-VirtualBox: ~/fyp/P... x
('Number is: ', 87)
('Number is: ', 21)
('Number is: ', 65)
('Number is: ', 25)
('Number is: ', 66)
('Number is: ', 8)
('Number is: ', 97)
('Number is: ', 10)
('Number is: ', 74)
('Number is: ', 17)
('Number is: ', 96)
('Number is: ', 53)
('Number is: ', 53)
('Number is: ', 61)
('Number is: ', 27)
('Number is: ', 55)
('Number is: ', 11)
('Number is: ', 79)
('Number is: ', 86)
('Number is: ', 81)
('Number is: ', 13)
('Number is: ', 10)
('Number is: ', 76)

```

Figure 19 - Publisher.py program running on terminal, sending message in topic

6.1.3 Subscriber

Figure 20 shows an example of a subscriber node called 'listener', which will receive a series random numbers generated as messages in the 'num_output' topic. This program will output the local node ID alongside "Message received, Number is: [random number]" message into the terminal in an endless loop until the program is terminated or no more messages are listed in the topic, as seen in Figure 19.



```

#!/usr/bin/env python
import rospy
from std_msgs.msg import Int16

#Program inspired by ros.org tutorial http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

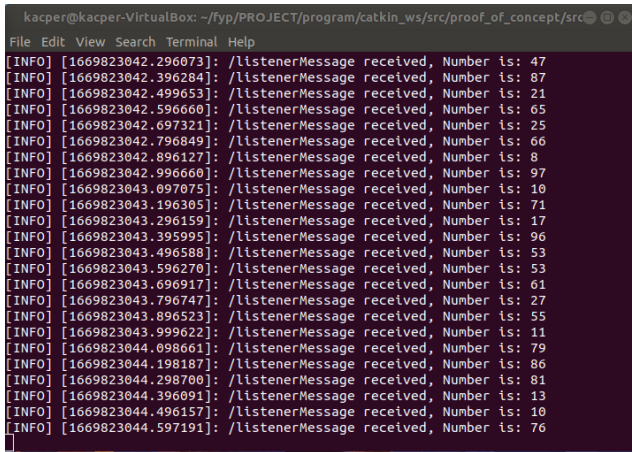
#outputs subscribed message
def terminal_output(data):
    rospy.loginfo(rospy.get_caller_id() + "Message received, Number is: %s", data.data)

#Creates new subscriber node and loops subscriber output
def subscriber():
    #Name of node
    rospy.init_node('listener')
    #declares node subscribing to 'num_output' topic.
    #Will be able to see msgs published from publisher
    rospy.Subscriber("num_output", Int16, terminal_output)
    #Keeps python from exiting until shutdown
    rospy.spin()

if __name__ == '__main__':
    subscriber()

```

Figure 20 - Subscriber.py



```

kacper@kacper-VirtualBox: ~/fyp/PROJECT/program/catkin_ws/src/proof_of_concept/src
File Edit View Search Terminal Help
[INFO] [1669823042.296073]: /listenerMessage received, Number is: 47
[INFO] [1669823042.396284]: /listenerMessage received, Number is: 87
[INFO] [1669823042.499653]: /listenerMessage received, Number is: 21
[INFO] [1669823042.596660]: /listenerMessage received, Number is: 65
[INFO] [1669823042.697321]: /listenerMessage received, Number is: 25
[INFO] [1669823042.796849]: /listenerMessage received, Number is: 66
[INFO] [1669823042.896127]: /listenerMessage received, Number is: 8
[INFO] [1669823042.996660]: /listenerMessage received, Number is: 97
[INFO] [1669823043.097075]: /listenerMessage received, Number is: 10
[INFO] [1669823043.196305]: /listenerMessage received, Number is: 71
[INFO] [1669823043.296159]: /listenerMessage received, Number is: 17
[INFO] [1669823043.395995]: /listenerMessage received, Number is: 96
[INFO] [1669823043.496588]: /listenerMessage received, Number is: 53
[INFO] [1669823043.596270]: /listenerMessage received, Number is: 53
[INFO] [1669823043.696917]: /listenerMessage received, Number is: 61
[INFO] [1669823043.796747]: /listenerMessage received, Number is: 27
[INFO] [1669823043.896523]: /listenerMessage received, Number is: 55
[INFO] [1669823043.999622]: /listenerMessage received, Number is: 11
[INFO] [1669823044.098661]: /listenerMessage received, Number is: 79
[INFO] [1669823044.198187]: /listenerMessage received, Number is: 86
[INFO] [1669823044.298700]: /listenerMessage received, Number is: 81
[INFO] [1669823044.396091]: /listenerMessage received, Number is: 13
[INFO] [1669823044.496157]: /listenerMessage received, Number is: 10
[INFO] [1669823044.597191]: /listenerMessage received, Number is: 76

```

Figure 21 - Subscriber.py program running on terminal, receiving message sent in topic

6.2 ROS Turtlesim automatic navigation

To learn more about the concepts of the ROS environment, there is an extension tool called ‘*turtlesim*’ within ROS for teaching how to use ROS and its packages. With this tool, I can create a program to move a turtle robot (Figure 24) in many ways on this 2D model. Figure 22 is a program designed to move the turtle at a constant speed across the x-axis until the program shuts down. Figure 23 shows the movement of the turtle, indicated by the trail created.

With this program, I can demonstrate that ROS has the capability to create programs that allow robots to navigate a map that they are placed within. This also further proves the importance of message passing, as the main computation of this program is to publish a message of the constantly updating position change of the robot, as seen with the ‘*velocity_publisher*’ variable using the updating ‘*vel_msg*’ coordinates as the message in the topic.

```

#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist

#Code inspired by http://wiki.ros.org/turtlesim/Tutorials/Moving%20in%20a%20Straight%20Line

def auto_move():
    #name of node
    rospy.init_node('turtle_move')
    velocity_publisher = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    vel_msg = Twist()

    #moving only on x axis at speed 1
    vel_msg.linear.x = 1

    #since moving only horizontally, y and z are 0 and no angles at x so also 0
    vel_msg.linear.y = 0
    vel_msg.linear.z = 0
    vel_msg.angular.x = 0
    vel_msg.angular.y = 0
    vel_msg.angular.z = 0

    while not rospy.is_shutdown():
        #constantly publish the travel distance of going forward in x axis
        velocity_publisher.publish(vel_msg)

if __name__ == '__main__':
    try:
        auto_move()
    except rospy.ROSInterruptException: pass

```

Figure 24 - auto_move.py program for turtle to move automatically

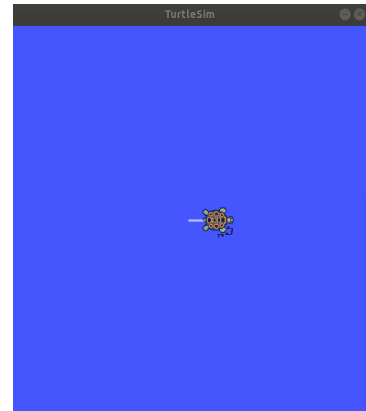


Figure 23 - turtlesim node moving after running auto_move.py

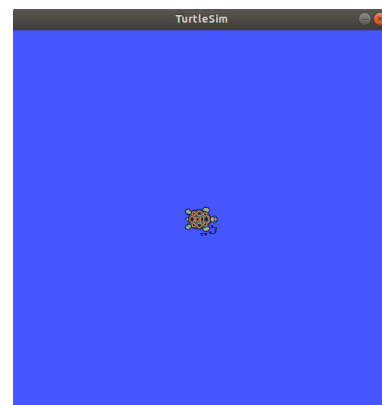


Figure 22 - Turtlesim node at starting location before auto_move.py is run

Chapter 7: References & Citations

- [1] D. Matine, "What Percentage of Car Accidents Are Caused by Human Error? | Virginia Law Blog", *Buck, Toscano & Tereskerz, Ltd.*, 2022. [Online]. Available: <https://www.bttl.com/what-percentage-of-car-accidents-are-caused-by-human-error/>. [Accessed: 30-Sep-2022].
- [2] J. Benson, Ed., "F1Tenth Build - Build Documentation," *f1tenth.org*, 2020. [Online]. Available: <https://f1tenth.org/build.html>. [Accessed: 30-Sep-2022]. F1Tenth Community in the University of Pennsylvania compiled this documentation
- [3] R. Ping Guan, B. Ristic, L. Wang and R. Evans, *Monte Carlo localisation of a mobile robot using a Doppler–Azimuth radar*, 97th ed. Automatica, 2018, pp. 161-166.
- [4] K.N. McGuire, G.C.H.E. de Croon, K. Tuyls, *A comparative study of bug algorithms for robot navigation*, Robotics and Autonomous Systems – Vol. 121, 2019
- [5] "Oracle® VM VirtualBox® - User Manual," *virtualbox.org*, 2004. [Online]. Available: <https://www.virtualbox.org/manual/UserManual.html>. [Accessed: 22-Nov-2022]. Sections 1.5 to 1.8 of the documentation are relevant
- [6] "Ubuntu install of ROS Melodic," *ros.org*, 25-Mar-2020. [Online]. Available: <http://wiki.ros.org/melodic/Installation/Ubuntu>. [Accessed: 22-Nov-2022].
- [7] J.-S. Zhao, X. Liu, Z.-J. Feng, and J. S. Dai, "Design of an Ackermann-type steering mechanism," *Proceedings of the Institution of Mechanical Engineers. Part C, Journal of mechanical engineering science*, vol. 227, no. 11, pp. 2549–2562, 2013, doi: 10.1177/0954406213475980.
- [8] Unknown, "Slash 4x4 Platinum: 1/10 scale 4WD Electric Short Course Truck with low CG chassis," *traxxas.com*, 02-Apr-2013. [Online]. Available: <https://traxxas.com/products/models/electric/6804Rslash4x4platinum?t=details>. [Accessed: 05-Dec-2022].
- [9] H. Gim et al., "Suitability of Various Lidar and Radar Sensors for Application in Robotics: A Measurable Capability Comparison," *IEEE robotics & automation magazine*, pp. 2–18, 2022, doi: 10.1109/MRA.2022.3188213.
- [10] S. Dehnavi, A. Sedaghatbaf, B. Salmani, M. Sirjani, M. Kargahi, and E. Khamespanah, "Towards an Actor-based Approach to Design Verified ROS-based Robotic Programs using Rebeca," in *The 16th International Conference on Mobile Systems and Pervasive Computing*, vol. 155, 2019, pp. 59–68. <https://doi.org/10.1016/j.procs.2019.08.012>.

Chapter 8: Diary

Week 03/10/22 - 09/10/22

03/10

Completed overview of project, explained motivation behind doing this and how it can be solved using F1Tenth ROS simulator.

04/10

Created a planned timeline on what I will work on during this project. Described tasks and length of time each will take.

Completed risk assessment on the project and how it will be mitigated.

06/10

First meeting with supervisor, questions answered about where the simulator will be (installed in Ubuntu). Got comments about re-arranging parts of abstract. Comments on re-wording risks to make them more targeted for this project.

Project plan submitted after corrections were made from the meeting

Week 10/10/22 - 16/10/22

Focus was shifted to catching up with other modules on the course.

Some attention put on checking how to set up ROS and F1Tenth simulator.

Week 17/10/22 - 23/10/22

Goals for this week:

Set up ROS on computer (via Ubuntu)

Begin researching and learning documentation of ROS.

17/10

Started with downloading Ubuntu desktop to work with simulator

18/10

Begun reading lecture notes that are accessible on F1Tenth website (<https://f1tenth.org/learn.html>).

23/10

Began writing report on the F1Tenth hardware, sensors and communications. Created reports and programs folders.

Week 24/10/22 - 30/10/22

Goals for this week:

Complete research on hardware, sensors and communication of the F1Tenth car

25/10

Continued research on hardware of the F1Tenth. Researched Chassis, NVIDIA Jetson NX, Lidar sensor.

29/10

Continued hardware report write-up. Completed writing majority of F1Tenth physical car's hardware. Plan to start writing how aspects link to simulator soon. Hopefully can finish report by mid next week to organise meeting with supervisor.

Week 31/9/22 - 06/11/22

Week focusing on courseworks for other modules. Plans to fully initialise Ubuntu.

Week 07/11/22 - 13/11/22

Week focusing on finishing report on hardware, potentially finish set up for F1Tenth simulator and ROS system

07/11

Downloaded Ubuntu. Began setup of ROS-Melodic but encountered problems. Meeting with supervisor on Wednesday will discuss this problem

08/11

Finished Hardware communication with ROS section of hardware report.

Week 14/11/22 - 20/11/22

15/11

Discovered that ROS Melodic works most stable on Ubuntu v18.04. Downloaded this version, ROS and simulator. Can now start working on writing code.

Week 21/11/22 - 27/11/22

22/11

Began writing configuration and installation of ROS environment and F1Tenth simulator report

23/11

Completed configuration and installation of ROS environment and F1Tenth simulator report

24/11

Saved simulator and files in the repository under 'program' package.

Week 28/11/22 - 04/12/22

29/11

Researched how to use ROS in more detail. Created 2 simple programs for POC on ROS. 'Publisher.py' will 'publish' a random number in the ROS system at a constant rate. 'Subscriber.py' will read and display such information published from 'Publisher.py'

30/11

Interim report created. Added introduction talking about why reasoning behind the project, how I am going to overcome and prove this concept.

01/12

Added aims and objectives of the project. Further research on algorithms (Bug algorithms, Monte Carlo Localisation, Ackermann Steering Geometry).

02/12

Added Installation & Config report to interim report. Added F1Tenth hardware report. Talked about message passing, PID control and how it is used in simulator and physical car.

04/12

Created simple turtlesim robot program to show automatic forward movement.

Week 05/12/22 - 11/12/22

07/12

Talked about proof of concept of message passing (publisher-subscriber), importance in ROS. Talked about turtlesim program. Completed recordings of proof of concept demo.