Analysis of Local Path Planner for Autonomous Driving

GENG5511 / GENG5512 Engineering Research Project Thesis

Student:
Daniel Trang (22003874)
Master of Professional Engineering (Electrical and electronic)
The University of Western Australia

Supervisor:
Professor Thomas Braunl
School of Electrical, Electronic and Computer Engineering
The University of Western Australia

Word Count (Abstract, acknowledgements and sections 1 to 6): 7705

Table of contents

Abstract

At the beginning of 2020, the Renewable Energy Vehicle (REV) team at the University of Western Australia (UWA) acquired a fully electric shuttle bus, later dubbed nUWAy. Upon purchase, all autonomous driving software used by the seller was erased from the vehicle's on-board computer. The eventual goal of this entire project is to deploy nUWAy onto the UWA campus, where it would autonomously drive between the Reid Library and the Business School, acting as a bus for the general public. Path planning is one of the most critical elements of autonomous driving. The university campus is a constantly changing dynamic environment; as such, the vehicle must navigate safely throughout the designated routes. Path planners are divided into a global planner and a local planner. The global planner requires a map of the environment in order to function, it is responsible for calculating the most efficient route from its current position to some user-defined end point, this is known as the global path. On the other hand, the local planner aims to generate new paths according to the vehicle's immediate environment, e.g. avoiding obstacles and undriveable terrain, while attempting to deviate from the global path as little as possible. Currently, a global path planner has already been implemented on the shuttle. The purpose of this research is to analyse different local path planning algorithms which can be installed on nUWAy's infrastructure and make a detailed comparison via simulations to find out which one is the most suitable for a campus environment. The two that were researched and experimented were the Dynamic Window Approach (DWA) and the Timed Elastic Bands (TEB). nUWAy utilises the Robot Operating System (ROS), a framework designed for robotics development. ROS has libraries that implements these two algorithms. Testing was performed within the ROS stage simulation environment; both algorithms were tested in different scenarios and the behaviour of the vehicle was analysed. Finally, the results of both algorithms were compared to determine whether or not they are suitable for use on the UWA campus.

Acknowledgements

First and foremost, I would like to thank my project supervisor; Professor Thomas Braunl, for providing me with the opportunity to work on such a unique and challenging project. Additionally, I would like to thank Yuchen Du, Kyle Carvalho and the rest of the REV team for their hard work, support and assistance over the course of the two semesters. I am very fortunate to have been able to learn and experience so much from this project.

Nonclementure

DWA	Dynamic window approach
PC	Personal computer
ROS	Robot operating system
SBPL	Search-based planning library
REV	Renewable energy vehicle
TEB	Timed elastic bands
UWA	The University of Western Australia
GPS	Global positioning system
IMU	Inertial measurement unit

List of figures

1. Introduction

In recent years, the demand for automation has been on the rise. This is no different for the automobile industry, companies such as Tesla and Google have been researching electric and autonomous cars for urban driving. There are many different components that make up a fully functional driverless vehicle; sensor fusion, localisation, computer vision and path planning. Just to name a few. Path planning is one of the most crucial abilities of any autonomous vehicle. To put simply, it is the vehicle's ability to plan a path between two points and to reach the destination as safely as possible.

1.1. Project background

In the beginning of 2020, the University of Western Australia (UWA) Renewable Energy Vehicle (REV) Project purchased an EZ10 Electric Cybercar, dubbed nUWAy. nUWAy is a 12-passenger, fully electric vehicle that has fully automated driving capabilities. However, all existing autonomous software produced by Easymile was removed from the shuttle's on-board personal computer (PC) upon purchase. All autonomous driving, navigation and path planning is being designed and implemented by the REV team via Ubuntu Linux and the robot operating system (ROS) framework. Some of the pre-installed onboard components include: a Linux PC (unknown specifications), one Curtis controller, two cameras (rear and front), six light detection and ranging (LiDAR) sensors and two touch screen displays.

Figure 1: nUWAy shuttle

The eventual goal of the nUWAy project is to deploy the vehicle onto the UWA campus, where it will autonomously drive between the Reid Library and the Business School, acting as a bus for the general public. This route could potentially change in the future to start from Ezone Central instead.

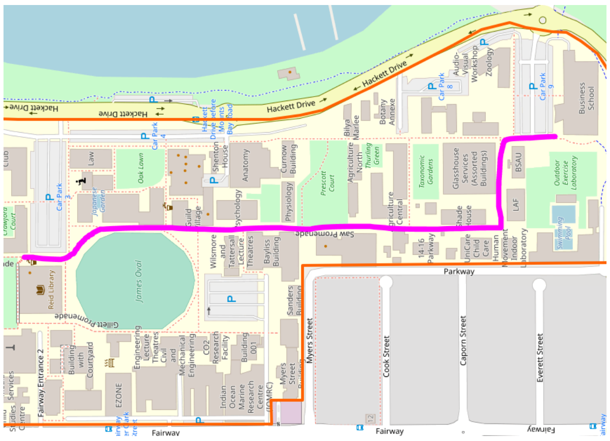
Figure 2: Route between Reid Library and Business School

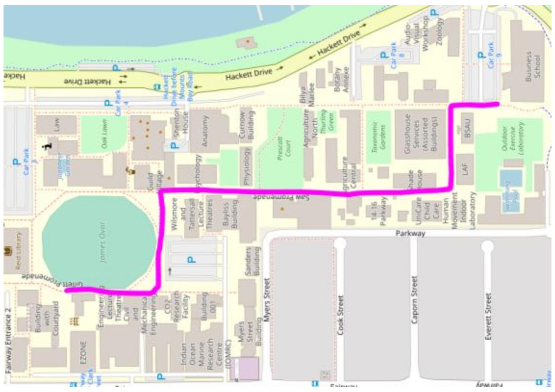
Figure 3: Route between Ezone and Business School

1.2. Scope

At the beginning of this project, another student had already begun their research on global path planners. As a result, someone was required to work on the local path planning. Path planning requires both a local and global planning algorithm, this will be discussed in further detail in the next section. At the time of writing this report, the nUWAy shuttle already has a high level global planner installed on the on-board computer and thanks to a low-level and basic local path planner developed by another student, it is able to drive autonomously. This local planner has no intelligent decision making and simply takes the global path generated by the global planner and divides it up into smaller segments which are then passed to the vehicle as drive commands. Using a low level local planner such as this one has many drawbacks because the UWA environment is dynamic and changes constantly, to ensure the safety of pedestrians and other vehicles, the shuttle needs to have the ability to make smart decisions in real-time; avoiding obstacles and other vehicles for example.

The scope of this project is to research and analyse the different high level local path planners which have more advanced functionality and decide on the most suitable one for the nUWAy shuttle and use on the UWA campus.

2. Literature review

In the field of robotics and automation, the term path planning usually refers to an autonomous vehicle's ability to plan a path between its current position and some user-defined goal, within a known environment. The path planning is divided into two sections; a global planner and a local planner. Both are discussed in further detail below. Before analysing complex algorithms, it was important to research the basics of path planning. The following section will outline the various areas of research that were reviewed throughout the project. Firstly, path planning is explored, followed by the different algorithms that were considered and compared. Finally, their implementation within the ROS framework were discussed.

2.1. Localisation

In order for an autonomous vehicle to successfully drive, it must know its own location within the environment it is travelling through. This means that nUWAy's position and orientation must be updated and known at all times. Depending on the technology available, localisation can be achieved through a variety of devices, e.g. GPS and IMU [1].

2.2. Coordinate system

There are two coordinate systems that need to be defined. Firstly, the global coordinates. This is the coordinate system of the map, which is used to pinpoint the location of the vehicle. Secondly, the vehicle has its own coordinate frame, also known as a local frame. The local frame is used to keep track of the vehicle's pose, orientation and movement as it travels [1].

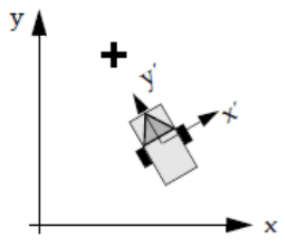

Figure 4: Combining global and local coordinates on one plane [1]

2.3. Path planners

2.3.1. Global path planner

Assuming that a map of the driving environment has been provided, the role of the global path planner is to plan develop a driveable path between two user defined points. The planned path is determined based on the properties of the provided map. At the time of writing this report, the nUWAy bus uses a ROS implemented global path planning algorithm known as search-based planning library (SBPL). The algorithm was tested and installed by a previous student and is in working condition.

2.3.2. Local path planner

The main objective of the local path planner is to alter the path and movement of the vehicle as it tries to follow the global path. In contrast to the global planner, it does not rely on a complete map of the environment, but only on the immediate reading from lidars and sensors. In order to transform the global path as required, the local planner will generate new waypoints after detecting obstacles and/or undriveable terrain. Considering the constraints of the vehicle, a new path is calculated. This newly created path attempts to utilise different avoidance methods, while adhering to the original global path as much as possible [2]. The two main advanced algorithms which this report will focus on are the dynamic window approach (DWA) and timed elastic bands (TEB), both of which are described in 2.3.2.1 and 2.3.2.2 below.

2.3.2.1. Timed elastic bands (TEB)

A traditional "elastic band" refers a path which can morph and change to avoid collisions and make adjustments in real time. An elastic band is represented by a sequence of *n* intermediate vehicle poses. Each robot pose is denoted by the following vector:

$$\mathbf{x}_i = (x_i, y_i, \beta_i)^T$$

Equation 1: Elastic band pose vector [3]

x and y refer to the position of the vehicle with respect to the world coordinates and β refers to the vehicle orientation as an angle.

The TEB algorithm augments the time intervals between all the intermediate robot poses, resulting in a sequence of *n – 1* time differences, denoted by Δt_i. Each of these Δt values is the time it takes for the vehicle to move from one pose to the next.

$$\tau = \{\Delta T_i\}_{i=0 \ldots n-1}$$

Equation 2: Sequence of time differences [3]

Therefore, the TEB algorithm can be denoted as a tuple of the pose sequence and time difference sequence:

$$B := (Q, \tau)$$

Equation 3: Tuple of both sequences [3]

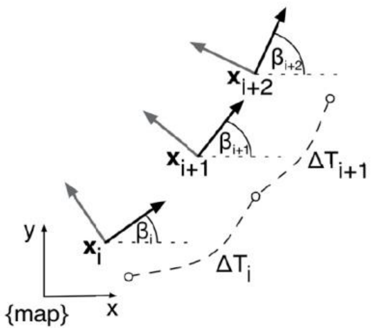

Figure 5: TEB sequence of poses and time differences [3]

TEB operates by constantly adapting and optimising the two sequences by using a weighted multi-objective optimisation in real time [3]. The objective function is denoted by:

$$f(B) = \sum_k \gamma_k f_k(B)$$

Equation 4: TEB objective function [3]

The objective function is a weighted sum of f_k constraints which rate the velocity and acceleration limits of the poses in terms of penalty functions. The penalties are based on the poses which result in the shortest and fastest trajectory. This makes TEB a very time-focuses algorithm, as it constantly attempts to find the next pose which results in the most efficient path. As a result, the local paths that it generates tends to have higher efficiencies than other algorithms.

2.3.2.2. Dynamic window approach (DWA)

The DWA is a commonly used sampling-based optimisation approach. In this algorithm, the search space is reduced to the velocities which are reachable under the dynamic constraints of the robot; e.g. its maximum steering angle, maximum speed, minimum speed, etc [4]. The algorithm firstly prunes the search space in an attempt to reduce the amount of possible paths that need to be considered. The search space of velocities is pruned using three steps. Firstly, DWA considers only circular trajectories, which are determined by pairs of translational and rotational velocities, this also results in a 2D search space. Assuming that the vehicle's rotational and translational velocities are controlled independently, the velocities can be modelled as a piecewise constant function in time. Under this assumption, the velocity trajectories can be modelled as a sequence of finite circular segments [4]. Next, only the admissible velocities are considered. Trajectories are considered admissible if the vehicle can stop before it results in a collision. Lastly, the admissible velocities are restricted by the dynamic window. The dynamic window contains only the admissible velocities which can be reached by the vehicle within the next simulation time interval, resulting in the search space being reduced even further. Figure 6 below shows an example of a dynamic window, indicated by the green square in front of the vehicle.

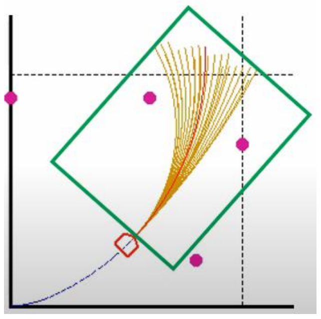

Figure 6: Dynamic window and admissible velocities

After the search space has been pruned, the algorithm will optimise the remaining velocities in the dynamic window using the following objective function:

$$G(v,\omega) = \sigma(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot vel(v,\omega))$$

Figure 7: Velocity optimisation function

The "heading" function measures the progress towards the final goal. Its weighting is maximised if the vehicle is orientated so that it faces the goal directly. "dist" stands for distance. This function is a clearance measurement. It is the distance between the vehicle and the closest object along a trajectory. A trajectory that contains no obstacles at all is given the highest possible weighting. "vel" stands for velocity. This weighting function evaluates the robot's progress on the corresponding trajectory. Its value is simply the translational velocity.

Each trajectory in the dynamic window is given a score based on this objective function, the trajectory with the highest score is deemed the most suitable path. The corresponding velocities are sent to the vehicle and the algorithm will repeat the process until the vehicle reaches its destination.

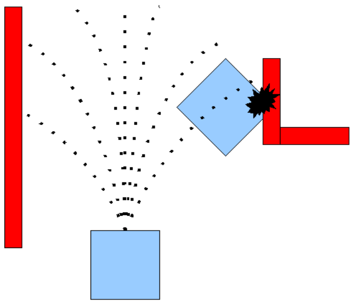

Figure 8: DWA Rollout [5]

Figure 8 above shows an example situation of the DWA algorithm in action. The blue square represents a vehicle. In this particular example, there are seven paths within the vehicles dynamic window, the maximum paths allowed in the dynamic window can usually be defined by the user. Out of those seven paths, three of them result in a collision, these will be instantly ignored. The figure does not show the end destination of the vehicle, but the remaining four paths will be chosen based on the one which has the highest scoring.

2.3.3. Cost map

A cost map is a representation of information that the path planners rely on, both the local and global path planners have their own distinct costmaps. Generally speaking, a costmap is a representation of the "cost" of traversing to different parts of the vehicle's map [6]. Costmaps are commonly composed of cells, which make up a grid-type structure. Conceptually, the costmap for both planners are designed to perform the same function and provide the same utilities. But in practice, there is a slight difference between the two. As mentioned previously, the global planner requires a pre-recorded map of the environment in which the vehicle will travel through. The information stored in the global costmap comes from the recorded map. On the other hand, the local costmap only contains information that the vehicle can see from its current position, e.g. walking pedestrians and walls. This information is usually obtained through sensors and/or lidars.

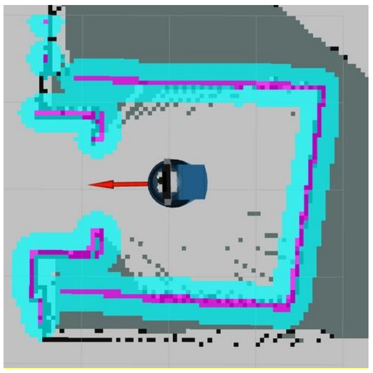

Figure 9: Example of a local costmap [7]

Figure 9 above shows an example of a local costmap. The highlights are used to represent the difficulty of the vehicle traversing to those particular positions. For example, the blue highlight will represent a medium-high difficulty, because those positions are close to walls and the vehicle should try to avoid those areas. Similarly, the solid walls are given a purple highlight, which represents an extremely high difficulty, as it is impossible for the vehicle to drive through walls.

3. Design philosophy

This section goes through the methodology and steps followed during the designing phase of the project.

3.1. Methodology

After the initial researching phase of the project, it became clear that it was not possible for one person to design a path planning algorithm within the given time frame. As such, the project became more research and experimental focused. Rather than design something from scratch, it was more realistic to analyse the existing algorithms which are supported by the shuttle's existing software framework, compare their differences and ultimately decide on a suitable one for use on the UWA campus.

3.2. Requirements

The following requirements have been identified as being critically important for the success of this project:

- Determining which criteria are the most crucial for assessing the suitability and efficiency of a particular path planner
- Obtaining simulation results with each algorithm through vigorous testing in common driving scenarios
- Determining which algorithm is the most suitable for the use of nUWAy in a campus environment

3.3. The robot operating system (ROS)

ROS is framework designed for robot software development. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. nUWAy utilises this framework in order to provide smooth communication between all onboard equipment (on board computers, sensors, LiDAR's, GPS, IMU, cameras, etc). At its core, ROS utilities a node to node type communication system. A node refers to an instance of a process. All of these nodes are registered to a "ROS Master". Which is the main computer that ROS is operating on. After registration these nodes are free to communicate with one another via a publish and subscribe messaging system. As mentioned previously in section 2.3.2, only two algorithms were analysed. This is because ROS has packages which implement these algorithms already. They are discussed in further detail in sections 3.3.5 and 3.3.6.

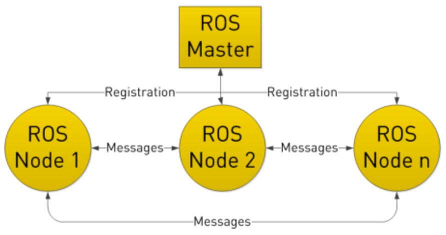

Figure 10: ROS communication structure [8]

3.3.1. Navigation stack

The navigation stack is a collection of nodes in the ROS framework that determines the motion planning of the vehicle. Conceptually, it is very simple. The stack takes in information from odometry and sensor streams. Based on the received data, it will output velocity commands to the vehicle [9].

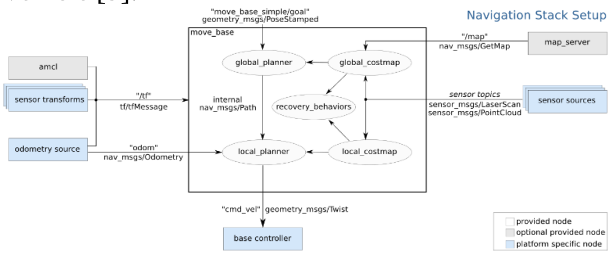

Figure 11: ROS navigation stack setup [9]

Figure 11 above shows the default navigation stack arrangement. There are three inputs which are required; sensor sources, the map and odometry sources.

3.3.2. move_base

move_base is the name of a provided ROS node that forms a major component of the navigation stack [10]. move_base aims to provide an interface for configuring, running and interacting with the navigation stack of nUWAy. Looking back at Figure 11, move_base is the group of nodes within the black rectangle. move_base is largely responsible for managing all of the autonomous path planning nodes. These nodes operate the local and global planners and their respective costmaps. The workflow of move_base is as follows:

- The coordinates of the final goal of the vehicle are sent to the global planner as a geometry message
- After receiving the final destination, the global planner will generate the optimal global path, which is calculated based upon the information in the global cost map
- The global path is passed to the local planner, which breaks the path down into smaller and more manageable motions which get sent to the vehicle as velocity messages. Like the global planner, the local planner also has a costmap which it uses to obtain information about its surroundings.

3.3.3. ROS stage simulator

Stage is the name of a basic simulation program in ROS. It provides a virtual 2-D world filled with user-defined objects, obstacles and robot models. The vehicle model will traverse through this world.

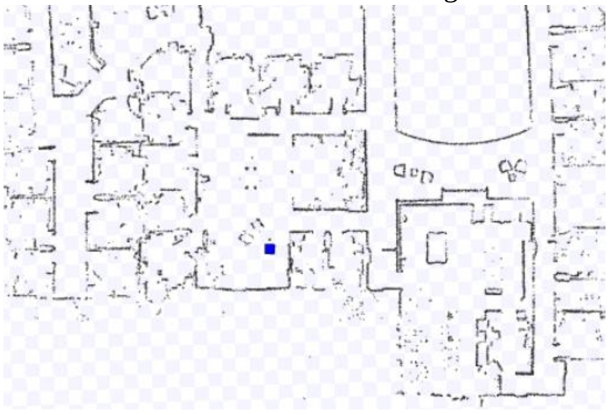

Figure 12: An example of a simulated world in ROS stage where the blue square represents a vehicle and the gray lines represent walls and undriveable terrain

3.3.4. RViz

RViz shorthand for "ROS visualisation". It is a 3D visualisation tool used for ROS applications which shows what the vehicle sees as it travels through its simulated environment. In this project, RViz is used to visualise the paths generated by both the global and local planner.

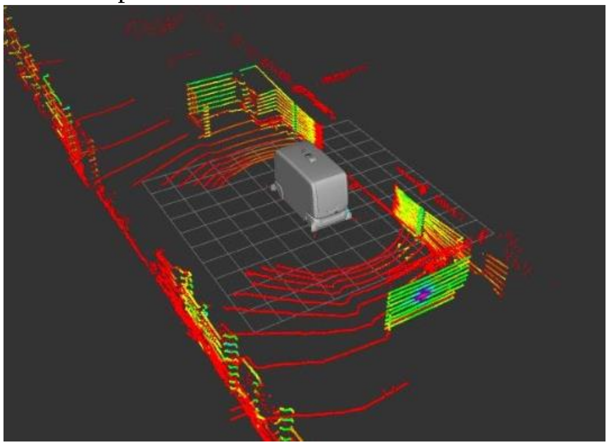

Figure 13: An example of RViz showing nUWAy's Lidar data [11]

3.3.5. teb_local_planner

This ROS package implements the TEB algorithm that was previously described in section 2.3.2.1. The current ROS implementation of this algorithm is designed for non-holonomic robots (car-like robots).

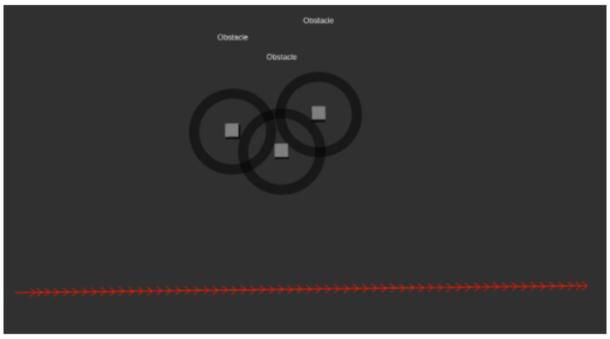

Figure 14: TEB path without obstacles

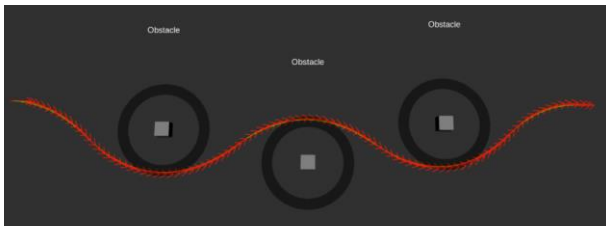

Figure 15: TEB path with obstacles

Figure 14 and Figure 15 above shows an example of the paths generated by teb_local_planner. This is a diagnostics ROS node which has been included in the package, which allows the user to visualise the behaviour of the path due to obstacles. The vehicle model is not shown in these figures, but the vehicles current position is the where the line starts on the left and the goal is where the line ends on the right. The TEB generated path is composed of many red arrows in sequence, which represent the poses the vehicle will take as time passes.

Traditionally, TEB algorithms have the tendency to get trapped in local optima, because the algorithm is incapable of traversing obstacles. As shown in Figure 15, when the first object is detected, a path is planned around it. But if that object moves to a different location, the algorithm still includes the object in its planned path, i.e. it doesn't have the ability to realise that the obstacle is in a location that no longer needs to be part of the altered path. This is shown below in Figure 16, it becomes apparent that the first obstacle on the left is extremely far away from its starting position from Figure 15 and should be ignored by the algorithm, as it no longer stands between the current position of the vehicle and its end point. Thankfully, the teb_local_planner has a way around this via parallel planning.

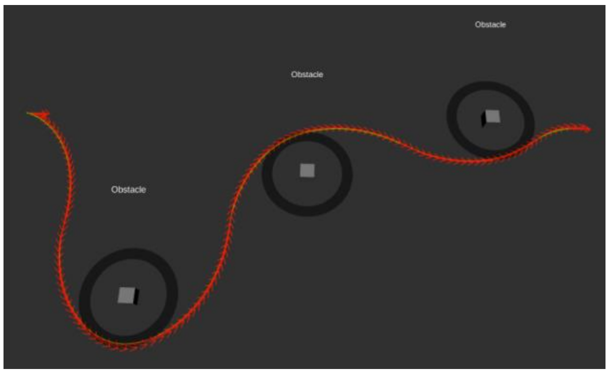

Figure 16: TEB stuck in a local optima

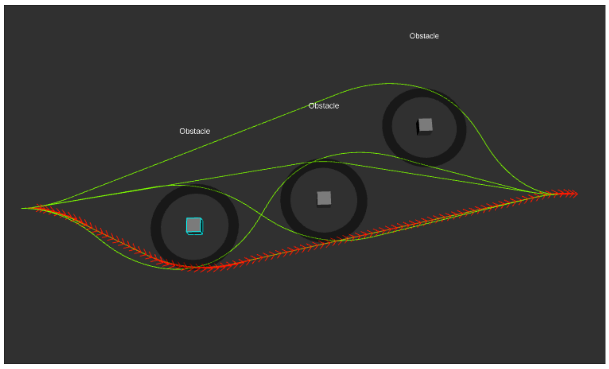

Figure 17: TEB generated parallel paths

Figure 17 above shows the generated paths from parallel planning. The path with the red arrows indicates the best possible route out of the potential candidates.

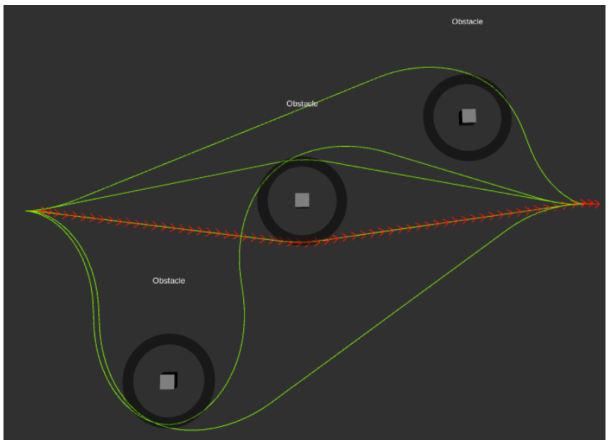

Figure 18: TEB no longer stuck in local optima due to parallel planning

Looking at Figure 18 above, this is the same obstacle orientation as shown in Figure 16. However, because parallel planning has been activated, TEB is able to choose a more efficient path and ignore the obstacle altogether.

3.3.6. dwa_local_planner

This ROS package implements the DWA algorithm that was previously described in section 2.3.2.2. It is actually the default local planner that the navigation stack utilises. Unlike the teb_local_planner package, the dwa_local_planner package does not come with any diagnostic tools, which made it harder to visualise the basic behaviour of the algorithm within the ROS framework.

In ROS the algorithm follows the same steps as described in section 2.3.2.2:

1) Discretely sample in the robot's control space
2) For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3) Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles).
4) Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
5) Repeat

Figure 19 below shows an example of DWA operation. The local costmap of the vehicle is also shown. The red line represents the global path and the shorter blue line represents the trajectory that was chosen after the control space is pruned and the admissible velocities within the dynamic window were scored.

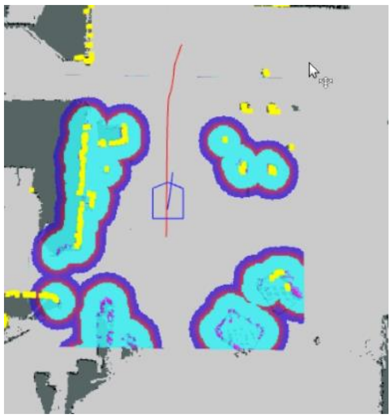
Figure 19: Example DWA operation in RViz

3.3.7. Vehicle modelling

The ROS planners allow the user to specify a footprint model of the vehicle, which approximates the vehicle's 2D shape. The footprint model is essential for optimisation and accurate simulations, as it determines the complexity of calculations and affects computation time [12].
There are a total of five footprint models to choose from:

1) Point
A single pixel point is used to represent the vehicle. This model results in the least amount of computation time as it requires very simple distance calculations.
2) Circular
A single circle with a user-defined radius is used to represent the vehicle. Calculations are similar to the point model, except with the added radius parameter.
3) Line
Vehicle is modelled using a line defined by a start point and end point.
4) Two circles
Vehicle is represented using two circles with user-defined radii and an offset value. Results in high computation times as two distance calculations are required, one for each circle.

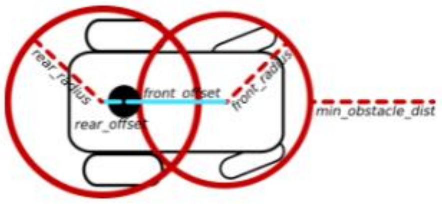

Figure 20: Two circle model [12]

5) Polygon
More complex vehicle models which are represented by shapes with many edges. The edges are defined by coordinate vertices. Computation time varies depending on the number of edges in the polygon.

For the simulations, the line model is chosen as it is the most suitable for modelling vehicles that have a rectangular shape. The exact length and width of the line was chosen to represent the shuttle as closely as possible.

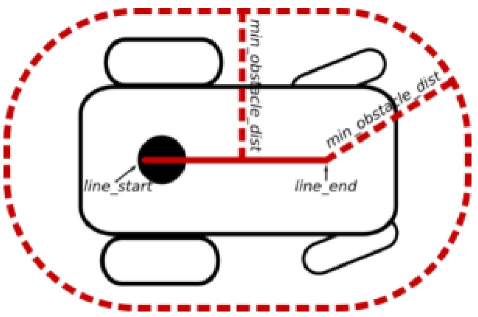

Figure 21: Line footprint model [12]

3.4. Final experiment setup

The final experiment design involves comparing the performance of the two algorithms through fair testing simulation experiments. Firstly, the ROS system was setup on a personal MacBook Air, running Ubuntu 18.04. nUWAy uses the Melodic version of ROS, to ensure fair testing and consistency, this version of ROS was also installed locally to house the simulations.

The simulations are setup using ROS stage and RViz, both of which have been described in section 3.3. ROS stage requires the user to input a map file, which the simulator uses to build the simulated world for the vehicle. A map of the entire campus has been recorded and developed by a previous student. Unfortunately, this map is very large and resulted in the simulations performing very poorly, mainly due to limited processing power. This led to inconsistent results and constant software crashes. As a result, two smaller and more resourceful maps were used, both of which were included within the ROS navigation package. These maps have a much lower resolution and more simplicity.

Figure 22: Corridor map

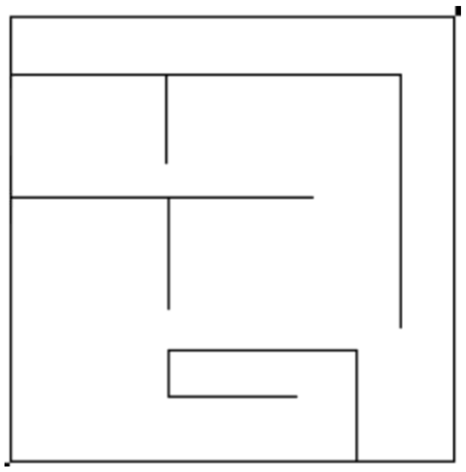
Figure 23: Maze Map

Figure 22 and Figure 23 above show the example maps that were used throughout the simulations. The black lines in the maps represent walls, which the vehicle will not be able to drive through. When these maps are loaded into ROS stage, obstacles can be placed inside the virtual world.

RViz is then used to send goal positions to the vehicle and visualise how the vehicle views its immediate environment and the behaviour of the path planners. Looking at Figure 24 below, the purple arrow is controlled by the user via a click and drag of the mouse. The location of the arrow on the map represents the destination coordinates of the goal and the direction of the arrows represents the orientation that the vehicle must finish in. In Figure 24, the arrow is pointing upwards, which means that the vehicle must reach that destination with the front facing upwards on the map.

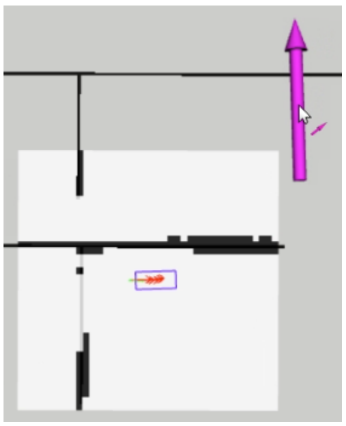
Figure 24: Setting a goal in RViz

In order to replicate the situations that the shuttle will encounter as it drives on campus, both algorithms will be simulated in the following scenarios: straight-line driving, static obstacle avoidance, simple turns, turning with limited space and complex manoeuvres.

<u>Straight line driving</u>
This is a very simple scenario that tests the local planner's ability to stick to the global path when there are no path changes required. Although it might seem unnecessary, this is an extremely important test because approximately 80% of the route between the Reid Library and Business School is a straight road. As such, the algorithms' ability to adhere to a straight global path must be confirmed.

<u>Static obstacle avoidance</u>
When driving along the designated route, there will be many cases where nUWAy will need to plan a path around obstacles. A common example would be parked security vehicles. On campus there are many security buggies that usually park towards one side of the road. nUWAy always be driving on the left hand side of the path, as a result there will be many cases where a parked vehicle needs to be avoided. Another potential need for avoidance will be road construction, where the vehicle will have to avoid undriveable terrain.

<u>Simple turns</u>
Along the two potential routes there are times when the shuttle needs to follow a curved path, next to James Oval for example, and also make simple turns. Once again, this is testing the planner's ability to stick to the global path when no path changes are necessary.

<u>Turning with limited space</u>
There is one turn near the business school where there is limited room for turning, even driving the bus manually through this turn proved difficult at times. By implementing this scenario in the simulation, the intelligence of the algorithms can be tested to see if they can perform the necessary adjustments to the vehicle to perform the turn.

<u>Complex manoeuvres</u>
Although nUWAy mainly does simple straight driving, it will be a huge bonus if the vehicle can

perform manoeuvres such as reverse parking. This also further tests the limits and intelligence of the algorithms.

In each of the scenarios mentioned above, the performance of each algorithm will be compared and discussed. Additionally, the following factors will also be discussed: computational intensity, parameter customisation, documentation and resource availability.

4. Results

The following section outlines the simulation results of the two algorithms.

4.1. Straight line driving

The corridor map is used to test the straight-line driving ability of both algorithms. The initial pose of the vehicle is set towards the upper wall of the corridor, to simulate the left-hand side of the road if the vehicle's forward movement is towards the right of the map.

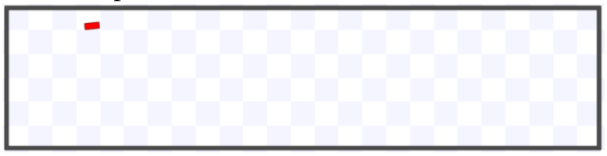

Figure 25: The starting position of the vehicle for the straight-line driving test

To no surprise, both algorithms perform the same in this scenario. Once the global path has been generated by the global planner, the local path simply follows it. This behaviour is expected, as there are no obstacles in the vehicle's immediate viscinity and there is no need to deviate from the global path.

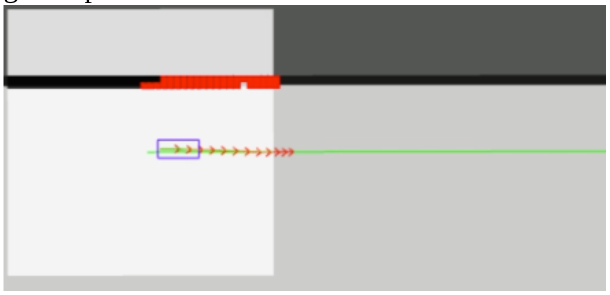

Figure 26: RViz showing straight line driving using TEB

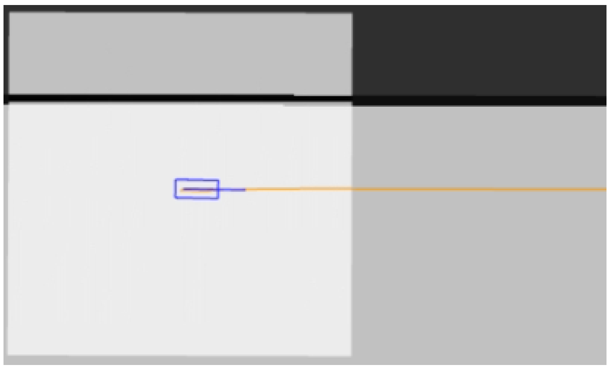

Figure 27: RViz showing straight line driving using DWA

Figure 26 and Figure 27 show the RViz display of the two planners during this scenario test. The path colours are set to different values for easier distinction.

4.2. Static obstacle avoidance

This test uses the same map setup as the previous straight-line test. Except this time, an obstacle is added, which is represented as a blue square in the ROS stage simulator. It is important to observe how the algorithms react when an obstacle is detected while the vehicle drives along the global path. Just like the previous test, the initial pose of the vehicle is set towards the top wall of the corridor.

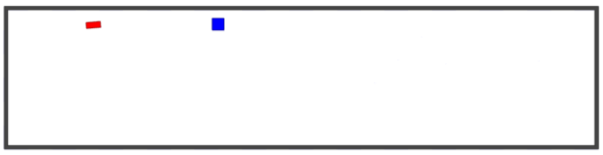

Figure 28: Initial position of the vehicle for straight line driving simulation

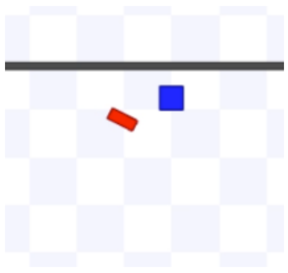

Figure 29: Vehicle avoiding the obstacle in ROS stage

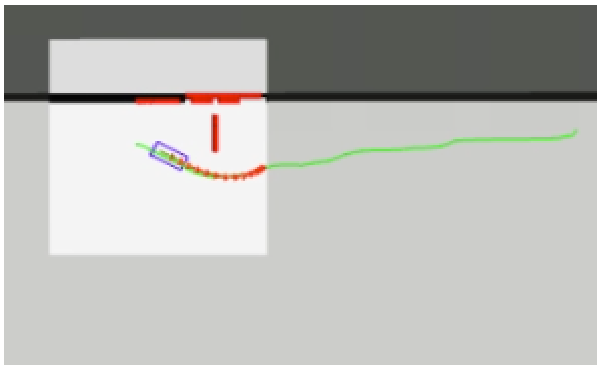
Figure 30: RViz showing the morphed path that TEB generates upon obstacle detection

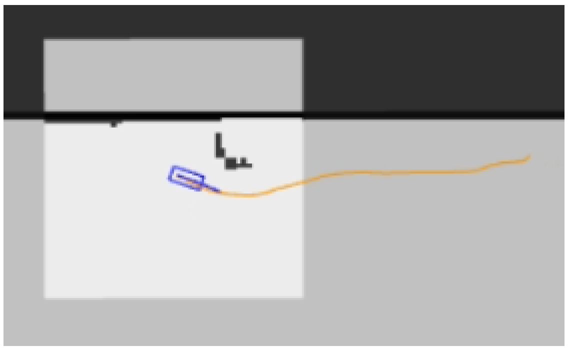
Figure 31: RViz showing the chosen simulated path that DWA has chosen to avoid the obstacle

Both algorithms were able to successfully pass the test of this scenario. Once the vehicle detected the object, it appears on RViz. The local paths generated by the local planners successfully deviate the vehicle from the global path in order to avoid the obstacle. Once the vehicle has moved past the obstacle, the local planners bring the vehicle back onto the original global path and continues travelling along that path until it reaches the end goal.

4.3. Simple turning

The various walls and corners in the maze map were used to test the turning ability of the planners. For all the turns, both planners were able to make the necessary adjustments to avoid hitting corners and reach their goals.

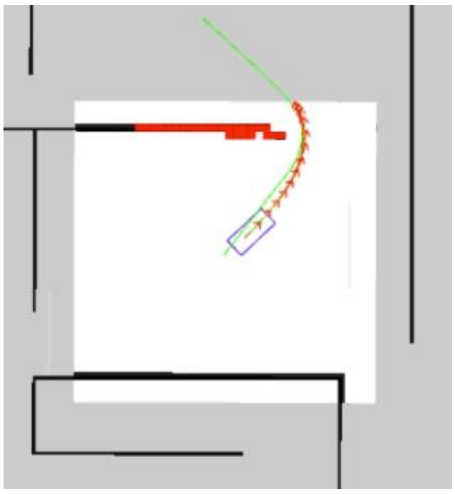
Figure 32: TEB doing a simple turn

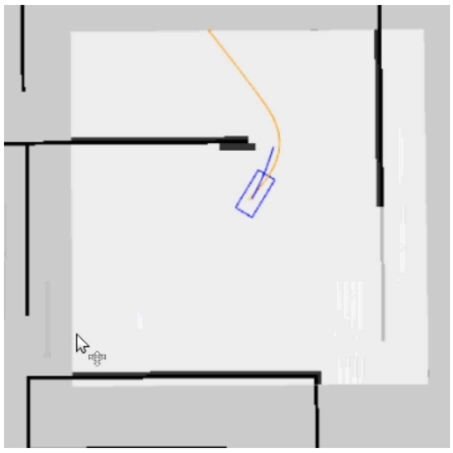
Figure 33: DWA doing a simple turn

However, it is worth noting that TEB results in much smoother vehicle movement. With DWA, the vehicle is very jerky and shaky when it makes a turn. In practise, this could potentially lead to passengers becoming off balance, which is a safety risk.

4.4. Turning tight corners

The initial position of the vehicle is set near the maze wall to simulate the edge of the driveable terrain and a rectangular obstacle is placed on the corner to restrict the amount of room available for the vehicle to turn.

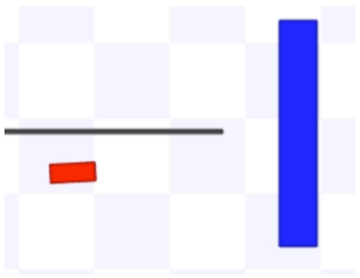
Figure 34: Initial starting position for the tight corner turn test, the blue obstacle is positioned to act as a wall

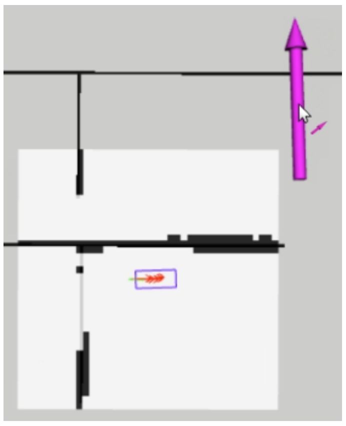
Figure 35: Setting the destination for the tight corner turn

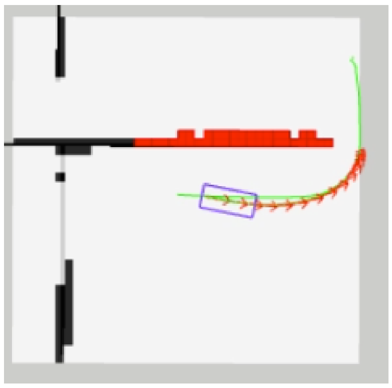
Figure 36: RViz showing the TEB generated path for turning the tight corner

The TEB algorithm successfully navigates the vehicle around the corner without any issues. The algorithm is intelligent enough to realise that there isn't enough room for the vehicle to make the turn with its current distance from the wall. This can be seen in Figure 36. The red arrows show the morphed local path produced by TEB, which has deviated from the original global path (indicated by the green line), notice how the local path is not aligned with the global path and causes the vehicle to turn slightly outwards before performing the left turn. After the turn has started, the local path also goes back to following the global path.

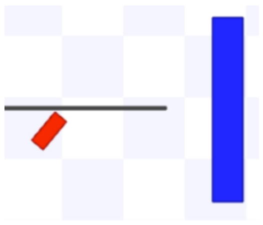
Figure 37: Vehicle colliding with wall while using DWA

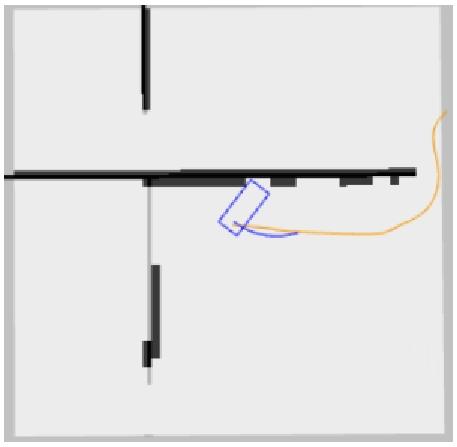
Figure 38: RViz showing vehicle collision and generated paths while using DWA

Unfortunately, the DWA algorithm is not able to successfully perform the turn. Looking at Figure 38, after the destination is set, it seems that DWA is able to produce the first segment of the local path, which aligns with the direction of the global path (indicated by the orange line), but the physical movement of the vehicle does not obey the planned paths. Unfortunately, this problem was not successfully debugged, as there are many possibilities for the cause.

4.5. Complex manoeuvres

This scenario tests the planners' abilities to make the vehicle perform a reverse park, starting at a reasonable distance from the parking space. A rectangular obstacle is placed parallel to the maze wall to represent a space for the vehicle to park.

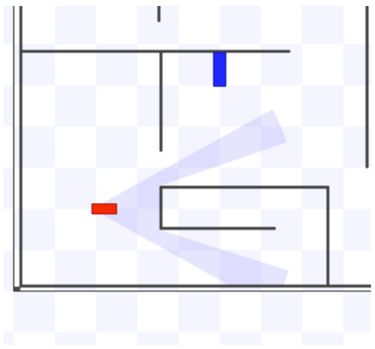
Figure 39: Starting position of the vehicle

Figure 39 above shows the starting position of the vehicle. In this test, the line of sight of the vehicle is also shown, which makes it easier to visualise the front of the vehicle.

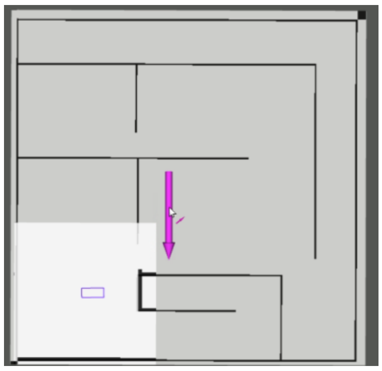
Figure 40: Setting the end goal and end pose using RViz

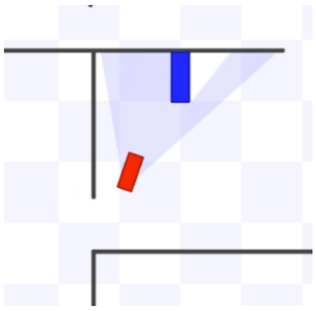
Figure 41: TEB moving the vehicle towards the parking space

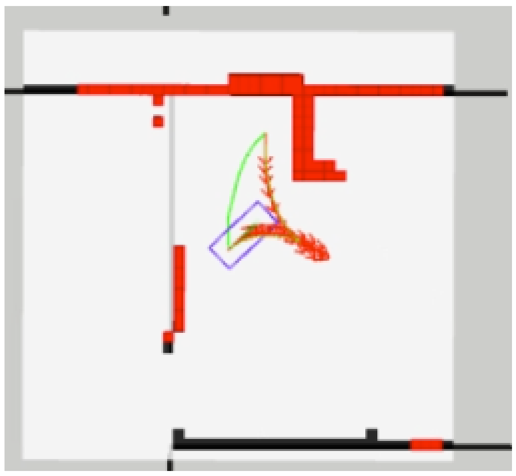
Figure 42: RViz showing the local paths generated by TEB when performing a reverse park

After receiving the end goal in the parking spot, TEB firstly brings the vehicle towards the parking spot. Figure 42 shows the decision making of the planner, after the blue rectangular obstacle has appeared in the vehicle's detection range. The local path is morphed so that the vehicle performs a right turn, which then allows for a reverse motion to move the vehicle to the set goal with the correct orientation. Figure 43 below shows the finishing position of the vehicle.

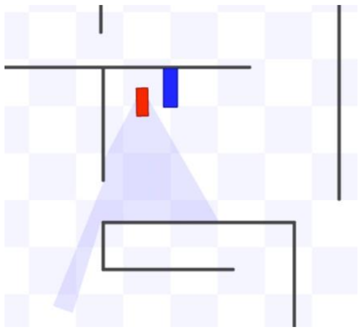
Figure 43: TEB successfully completing a reverse park

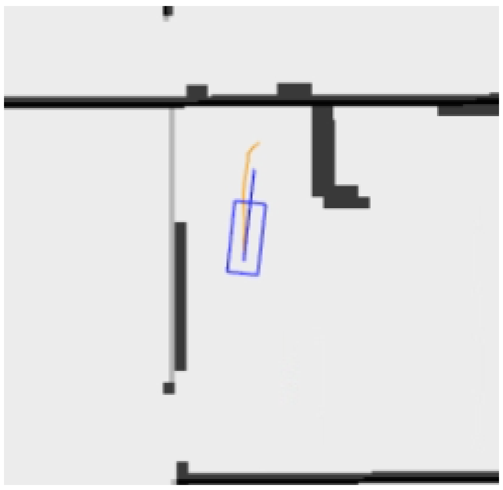
Figure 44: DWA fails to plan local paths required for reversing

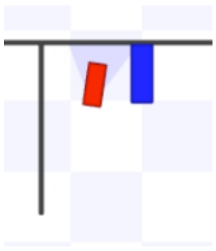
Figure 45: Final pose while using DWA

On the other hand, DWA was not able to successfully perform a reverse park. The vehicle simply drives forward until it reaches the destination. It became extremely apparent in this experiment, that DWA has issues obeying the final orientation of the user defined goal. It is able to bring the vehicle to the end point, just without the correct orientation. Similar to the previous issue, where the vehicle drove into the wall, the orientation error was unable to be debugged successfully.

4.6. Computational intensity

Since there is limited computational power on the nUWAy shuttle and new software is constantly being added to it, it is important to note which algorithm is more resourceful. As mentioned previously, all testing and simulations were done on a personal laptop which had limited computing power. The CPU usage of the simulator is found by using the "activity monitor" program on the Ubuntu operating system. Throughout all the simulations, the CPU usage stayed fairly consistent for the two algorithms. However, TEB seemed to be the slightly less demanding algorithm, averaging at around 14% CPU usage. On the other hand, DWA resulted in the simulation process averaging at around 21% CPU usage. This is a minor difference but it is still worth keeping in mind.

4.7. Ease of use, customisation and flexibility

Ease of use is an important factor when comparing the algorithms because eventually they will be installed into the existing infrastructure of the shuttle and future students joining the project will need to learn how to operate them. As such, it is important that the algorithms are able to be easily integrated into nUWAy's existing ecosystem. Because both of the packages which implement the algorithms are designed for use within ROS, installing them on a local machine was relatively straight forward. Installation on the nUWAy system should follow the exact same steps. Operating the algorithms is also extremely straightforward, all that is required is for a goal to be set using RViz.

The parameter settings of the two algorithms can be modified by changing the values within their configuration files. The parameters allow the user to customise the behaviour of the vehicle and tune certain settings to optimise performance. As mentioned previously in section 3.3.7, the line model footprint was chosen, as it best represents the shape of the nUWAy shuttle. Both algorithms offer full customisation of the vehicle footprint, which means that the algorithms can be tuned to obey the physical dynamics and constraints of nUWAy.

When it comes to the driving behaviour, TEB has a much wider range of parameters which can be customised. It also allows for more specific customisations. For example, when setting the minimum distance of the vehicle from obstacles, TEB allows the user to input specific numerical values, down to two decimal places. This is a great feature, as it allows the user to specify the exact distance from obstacles that is deemed safe. With the distance parameter settings of DWA, the user is unable to specify exact numeric values, only the obstacle weighting values. This means that it is ultimately up to the algorithm to decide the exact minimum distance between the vehicle and obstacles.

TEB also has more optimisation settings, which adds flexibility. These settings allow the user to control how strictly the vehicle must adhere to the weightings and penalties that the algorithm calculates from the costmap data. In practise, sometimes the vehicle will miss waypoints, miss the end goal or fail to reach the velocity that the algorithm passes to it. If the strictness is set to high values, the algorithm will think the vehicle has failed to respond to its commands, which leads to unwanted and unexpected behaviour. This was a common problem during testing of the global path planner. The shuttle would drive back and forth between two way points because one of them was missed by a small distance. By allowing the user to customise these strictness settings, it gives the vehicle more room for error without resulting in software crashes or unexpected behaviour. TEB offers many optimisation parameters for the vehicle speed, vehicle angular velocity, end goal tolerance and waypoint tolerance, just to name a few. On the other hand, DWA only offers a goal distance and path distance tolerance.

4.8. Resource availability

The ROS system operating on nUWAy will only get more complicated as newer software is constantly being added. As mentioned previously, there are many elements which are responsible for successful autonomous driving. If any of these elements run into any problems, the entire driving system collapses. Installing and debugging these planners can be a difficult task and any help from online resources and/or documentation can go a long way. By far, TEB is the newer and more popular algorithm. Not only does it have more online resources, it is also the topic of many more published papers. The ROS wiki also has a list of in-depth tutorials that guides the user through installation, testing and parameter tuning. For DWA, there are barely any guides or documentation on the ROS wiki and it is not nearly as popular as TEB.

4.9. Summary of findings

In the straight line test both algorithms behaved in the exact same way. This was a simple but important experiment that aimed to test the algorithms' ability to follow the global path when no changes to the vehicles route were required. As expected, both algorithms were able to follow the global path without ever deviating from it.

In the static obstacle test, once again both algorithms behaved in the exact same way. Once the obstacle appeared in the vehicles line of sight, it was shown on their costmaps and the algorithms were able to plan a path around the obstacle and deviate from the global path as required.

In the regular turning test, both planners were able to successfully turn corners without collisions. When needed, both planners could make subtle adjustments to their driving angles in order to create more space. However, because TEB results in smoother vehicle movement during turns, it is the superior planner in this test.

In the tight corner turn test, only TEB was able to successfully turn the vehicle without collisions. TEB was intelligent enough to adjust the driving angle and create more space in order to make it past the corner in one smooth arc. On the other hand, although RViz showed that DWA was able to generate a local path, the vehicle was not able to comply, and simply drove into the wall. This happened every single time, after countless tests and parameter changes, DWA was never able to successfully complete the turn.

In the manoeuvre test, the algorithms were tested to see if they could perform a reverse park. It became very obvious in this experiment that DWA is unable to obey the final pose set by the user. Whereas TEB was able to intelligently position the vehicle, make enough room for itself and reverse into the parking space.

Unfortunately, the tests that DWA failed were not able to be debugged successfully. After many parameter changes, costmap setting changes and configuration file changes, the exact same problems still persisted. For the tight corner test, it can be seen that the algorithm actually produces the correct local path, but for some reason the vehicle in ROS stage is unable to follow. This is a very unusual problem that is likely due to software bugs in the actual simulator itself. Because the planner worked fine for the scenarios without the added obstacle, there is a possibility that there is some sort of version incompatibility which causes unexplained behaviour. The same can be said for the failure of the reverse park manoeuvre.

Additionally, TEB is less computationally intensive, offers more customisability, parameter settings and contains much more tutorials and documentation.

After doing a side by side comparison of the two algorithms, it can be concluded that TEB is the more superior planner as it didn't run into any errors during the experiment and was able to successfully pass all of the simulated scenarios. It seems to be a more intelligent algorithm than DWA, has better decision making and offers more customisation and flexibility.

Test case	TEB	DWA
Straight line driving	✓	✓
Static obstacle avoidance	✓	✓
Simple turns and curves	✓	
Tight corner turning	✓	
Complex manoeuvres	✓	
Computational intensity	✓	
Customisation and flexibility	✓	
Documentation and resource availability	✓	

Table 1: TEB vs DWA results summary, ticks show the better algorithm in each comparison

5. Future work

This section will outline any possible improvements to the current research and future tasks that can expand upon the discussed functionalities.

5.1. Performing more advanced simulations

As mentioned in section 3.3.3, ROS stage is a very simple simulation tool. It can be argued that the simulation environment is not very realistic, as it is only modelled as a 2D world. A group of past students have developed a 3D map of the entire UWA campus within a simulation system known as CARLA. The CARLA simulator is an open-source simulator designed specifically for autonomous driving research. Unfortunately, during the duration of this research project the PC which runs the CARLA system was out of order and could not be used to test any of the discussed algorithms. If this system is fixed in the near future, it would be worthwhile to do additional testing with CARLA, as it offers a more realistic representation of the real world.

5.2. Testing on nUWAy

Simulations and comparisons can provide a general insight into the characteristics and tendencies of the algorithms, but their effectiveness in practice cannot be perceived until they have been integrated into the existing ROS system on the shuttle and thoroughly tested. Because the experiments suggest that TEB is the more suitable algorithm, it should be the first to be installed and tested. If nUWAy performs as the simulations suggest, then no more needs to be done. However, if it turns out to be unsuitable, then DWA should be tested. If problems still persist, then further research needs to be done on the countless path planners outside of ROS and a custom ROS node needs to be developed by future students.

5.3. Obstacle detection

Since nUWAy currently drives using a pre-recorded map, it cannot recognise obstacles which appear in its immediate viscinity. There are many devices on the shuttle which allow for obstacle detection development. The TEB algorithm already has the ability to plan paths around static obstacles, the detection is the only thing that needs to be implemented.

5.4. Dynamic obstacle avoidance

The TEB package has dynamic obstacle avoidance capabilities included. However, due to the nature of the nUWAy shuttle and also the campus environment, it was decided for this project that such capabilities are not needed. The shuttle drives at an incredibly slow pace (near walking speed) and most dynamic obstacles that will be encountered are pedestrians. Due to the size of the vehicle and its speed, it is very unrealistic to make the vehicle manoeuvre around people, especially during the times between classes where the paths are extremely congested.

6. Conclusion

To conclude, the goal of this project was to investigate the different path planning algorithms which are available on the ROS framework. The Dynamic Window Approach and the Timed Elastic Bands algorithms were the two that were researched. Through simulation experiments, the performances of the two planners in different driving scenarios were observed and discussed. After testing and comparing them side by side in a simulated environment, it was concluded that TEB outperforms DWA in a majority of the tests. Based purely on the simulation results, TEB is definitely the more suitable algorithm for the nUWAy shuttle as it didn't run into errors during testing, is more efficient and has more to offer. However, simulation behaviour can be very different from real-world performance and some factors still remain unknown until testing has been done on the nUWAy shuttle.

References

[1] T. Braunl, Embedded Robotics: Mobile Robot Designa nd Applications with Embedded Systems, Berlin: Springer, 2006.

[2] A. H. D. M. A. d. l. E. Pablo Marin-Plaza, "Global and Local Path Planning Study in a ROS-Based Research," Wiley, 2017.

[3] W. F. T. W. Christoph Rosmann, "Trajectory modification considering dynamic constraints," 2012.

[4] W. B. S. T. Dieter Fox, "The Dynamic Window Approach to Collision Avoidance," Pittsburg, 1997.

[5] "ROS.org," [Online]. Available: http://wiki.ros.org/dwa_local_planner. [Accessed 24 3 2021].

[6] "ROS Answers," [Online]. Available: https://answers.ros.org/question/309308/what-is-a-costmap/#:~:text=A%20costmap%20is%20a%20fundamental,the%20ground%20is%20rough%2Fsloping.. [Accessed 3 5 2021].

[7] "Research Gate," [Online]. Available: https://www.researchgate.net/figure/Left-Local-costmap-around-the-robot-Right-Laser-scan-simulated-by-raytracing-30-points_fig6_326798547. [Accessed 2 3 2021].

[8] "Clearpath Robotics," [Online]. Available: https://www.clearpathrobotics.com/assets/guides/kinetic/ros/Intro%20to%20the%20Robot%20Operating%20System.html. [Accessed 4 4 2021].

[9] J. S. Gill, "ROS.org," [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup. [Accessed 29 3 2021].

[10] N. Fragale, "ROS.org," 03 09 2020. [Online]. Available: http://wiki.ros.org/move_base?distro=noetic. [Accessed 2 4 2021].

[11] "The REV Project," [Online]. Available: https://therevproject.com/vehicles/nuway.php. [Accessed 24 2 2021].

[12] C. Roesmann, "ROS.org," [Online]. Available: http://wiki.ros.org/teb_local_planner/Tutorials/Obstacle%20Avoidance%20and%20Robot%20Footprint%20Model. [Accessed 25 3 2021].

[13] C. Roesmann, "ROS.org," 11 03 2020. [Online].

[14] O. K. Sean Quinlan, "Elastic bands: Connecting path planning and control," Standford University, Stanford.