This thesis is presented for the degree of Master of Professional Engineering at The University of Western Australia

GENG5511/5512 Engineering Research Project: Building a software stack for an autonomous shuttle with GNSS localisation

Shao-Ming Tan (20920822)
20 May 2023

School of Electrical, Electronic and Computer Engineering
Robotics and Automation Lab
The University of Western Australia

Supervisor: Dr. Thomas Bräunl

Word count: 7,912

Abstract

Autonomous vehicles pose a technological challenge spanning multiple domains including electronics, mechanical systems, computer science, and software development. To further advance research in this field, the University of Western Australia has been conducting research with two EZ10 shuttle buses acquired from EasyMile.

The focus of this research is twofold, firstly in improving software architecture using open-source tools, and secondly in navigation using satellite positioning. The former is aimed at improving software reliability and portability to keep up to date with the rapidly evolving open-source robotics ecosystems. A system using the containerisation technology of Docker is introduced to provide a platform that can keep up with the yearly release of the Robot Operating System 2 and Navigation2. Improvements to the existing launch system are also undertaken to increase the modularity of the system through Docker Compose.

The latter focus of this research is on autonomous navigation through the integration of a correction stream for Real Time Kinematic satellite positioning. Path planning and control algorithms within Navigation2 are tested, tuned, and evaluated within a university campus for exact path following. The system demonstrated accurate localization consistently achieving accuracy under 15cm and was able to plan a path to a goal point and determine the control effort to follow the planned path. Additionally, a software package was successfully developed to enable the shuttle bus to follow a set of GNSS waypoints for future on-road testing.

The results from this research improve the existing software architecture by making it more modular and demonstrate the feasibility of localisation with satellite positioning. In the future, the robustness of localisation can be improved by integrating odometry and data from LiDAR sensors.

List of Publications

Submitted for publication:

K. Quirke-Brown, Z. Lai, X. Kong, T. Tan, Y. Du and T. Bräunl. (2023, April 17). *Developing an Autonomous Shuttle Service*. Australasian Transport Research Forum 2023.

Contents

List of Figures

List of Tables

Nomenclature

AV Autonomous Vehicle

CAN Controller Area Network

DDS Data Distribution Service

EKF Extended Kalman Filter

GLONASS GLObalnaya NAvigatsionnaya Sputnikovaya Sistema

GPS Global Positioning System

IMU Inertial Measurement Unit

INS Inertial Navigation System

LiDAR Light Detection And Ranging

Nav2 Navigation 2

PLC Programmable Logic Controller

RADAR RAdio Detection And Ranging

REV Renewable Energy Project

RL Robot Localization

ROS Robotic Operating System

RTK Real-Time Kinematics

TEB Timed Elastic Bands

UWA University of Western Australia

Chapter 1

Introduction

1.1 Overview

With rapid advances in computing power and the electrification of vehicles, interest has grown exponentially in driverless technology over the last decade. Over $220 billion has been invested into more than 1,100 companies since 2010 [1]. While many companies are developing driverless technology behind closed doors, another approach with open-source robotics has been gaining popularity to tackle this immense challenge. Open-source frameworks like the Robotic Operating System (ROS) have become de facto standards and are now used globally by companies for a range of applications.

Autonomous vehicles (AVs) have to address a large number of challenges which can be categorised into perception, planning, and control (Figure 1.1). These challenges involve gathering information about the environment through sensors, making decisions about how to get from one place to another, and interfacing with a vehicle to physically move the platform to the destination.

The Society of Automotive Engineers J3016 standard [3] from 2016 defines several levels of autonomy for driverless vehicles. They range from level 0 in which there is no autonomy, to level 5 in which a vehicle can be said to be fully autonomous (Figure 1.2). At present, most applications of AV exist around levels 3 and 4 [4] in which the vehicle is driven autonomously under limited conditions and driver intervention is required at points.

1.2 Background information

The University of Western Australia (UWA) operates the Renewable Energy Vehicles (REV) project [5] which has two electric, shuttle buses originally developed by AV company EasyMile. The first shuttle bus, termed nUWAy, is an EasyMile EZ10 Gen1 electric shuttle bus [6] that was purchased with no existing software. The second shuttle bus is a Gen2 EZ10 and was acquired recently in August 2022 for use in on-road trials in a residential development in the north of Perth.

The bus is a four-wheeled bus powered by electric motors and a 48V battery pack. It includes eight Light Detecting and Ranging (LiDAR) scanners, an Inertial Navigation System (INS) which combines an Inertial Measurement Unit (IMU) with a Global Navigation Satellite System (GNSS) receiver, and a front and rear camera. The EZ10 is a small shuttle bus, designed for 10 passengers measuring 3.93 metres long and 2.00 metres wide. The shuttle bus has four-wheel steering which produces a minimum

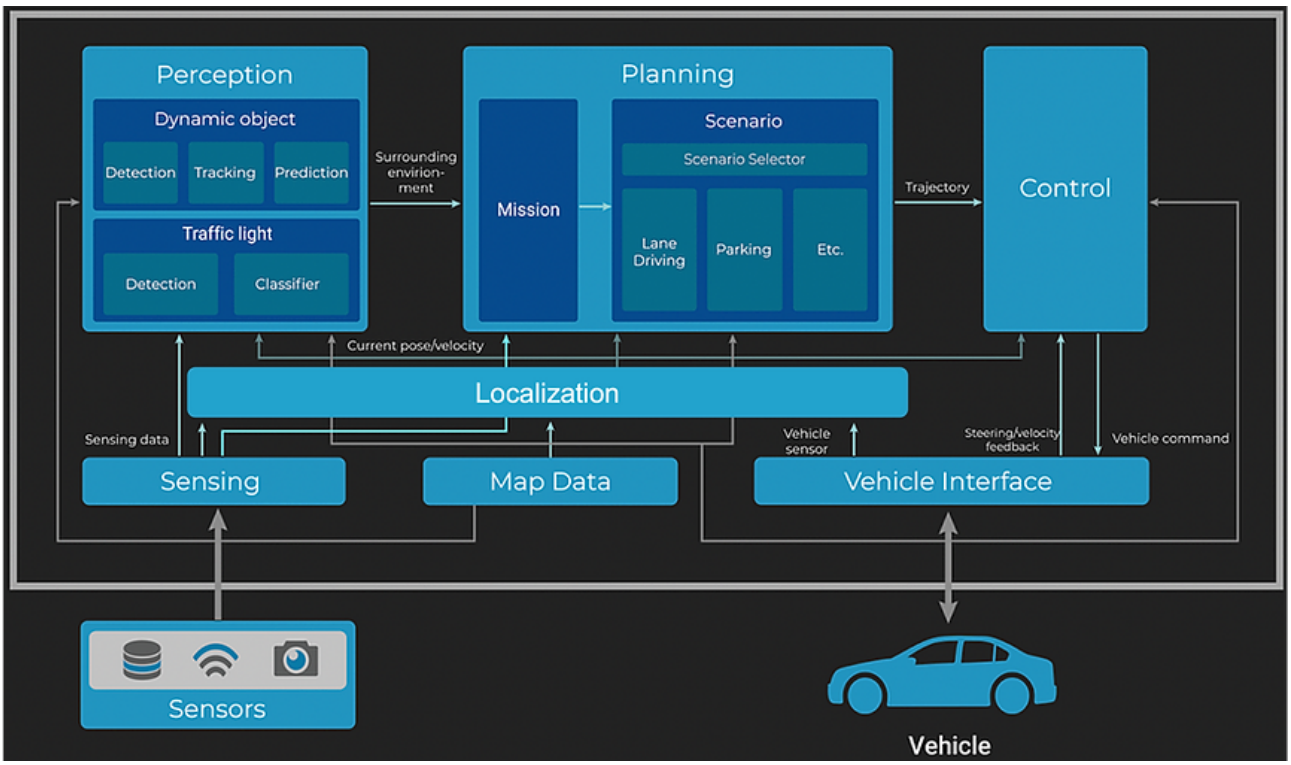


Figure 1.1: Tasks of Autonomous Vehicles. Adapted from [2]

turning radius of 5 metres. The vehicle has a top speed of 40km/hr but university rules dictate a maximum speed of 5km/hr for the shared laneways used by pedestrians and cyclists.

The vehicle has eight LiDAR sensors which it uses to gather information about its environment.

1. 4 SICK LMS1xx: One at each corner of the vehicle situated 30cm above the ground with a 270° single layer horizontal scan and a range of 40m.

2. 2 Velodyne VLP16 LiDARS: One at each end of the vehicle, mounted 0.8 metres above the ground. They provide laser scans of 16 layers with a range of 130m.

3. 2 SICK LD-MRS: Long-range LiDARs which are situated on the roof. One facing the front and the other the rear, they provide laser scans of 4 layers with a range of 250m and a 110° field of view.

The bus was originally equipped with an Xsens MTi-G-710 IMU [7] and a NovAtel OEM628 single antenna GNSS receiver. These were replaced by UWA with a high-performance INS from SBG Systems, the Ellipse-D, as it provided better accuracy in previous research [8]. The Ellipse is a dual-antenna multi-band GNSS receiver that provides attitude, heading, heave as well as navigation outputs [9]. It can receive a correction stream for Real-Time Kinematics (RTK) which allows sub-centimetre GNSS positioning. The Ellipse is able to provide accurate position tracking with the use of dual antenna GNSS and RTK capability when compared to the Xsens which relies on magnetic orientation, it provides significant improvements in accuracy over the previous IMU and GNSS receiver.

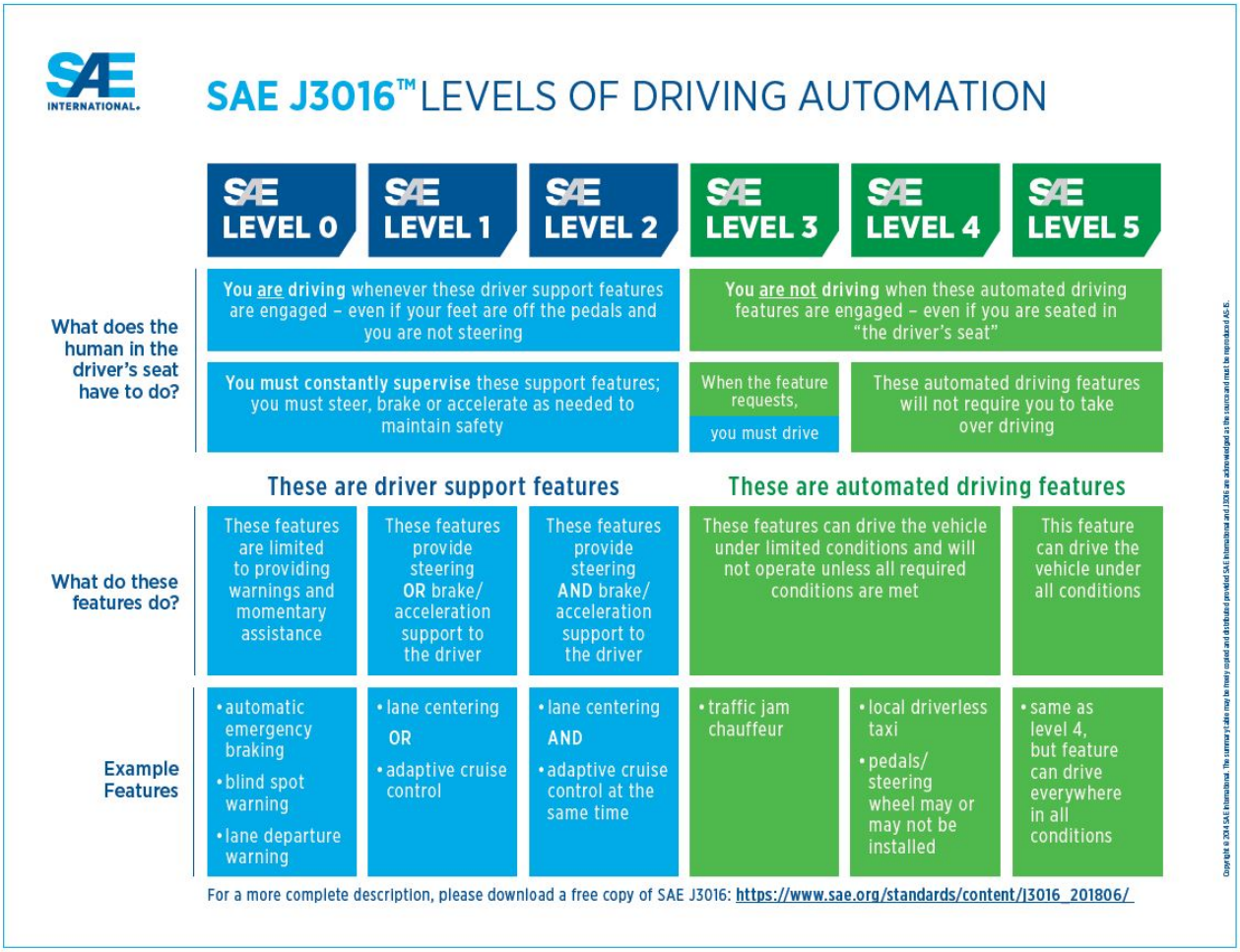


Figure 1.2: The SAE J3016 standard for driving automation. Adapted from [3]

1.3 Previous Research

nUWAy has been undergoing ongoing research at UWA beginning in 2020. Early research used ROS1 and performed basic functionality and has improved over the years with a migration to the more secure ROS2 in 2021 [10] in which dynamic obstacle avoidance with the Timed Elastic Bands (TEB) control algorithm [11] using LiDAR-based localisation was done successfully. Previous research has also looked at using Extended Kalman Filters (EKFs) with some success with sensor fusion of GNSS and LiDAR [12, 8].

Vehicle controls were locked by the manufacturer and without direct access to the motor controllers, previous research has emulated driving commands for a joystick over a serial interface in the vehicle's manual mode. An interface board was developed that sat as an adapter between the main industrial PC and the motor controllers, converting algorithm output into serial drive commands that would control the motor.

Figure 1.3: nUWAy shuttle bus

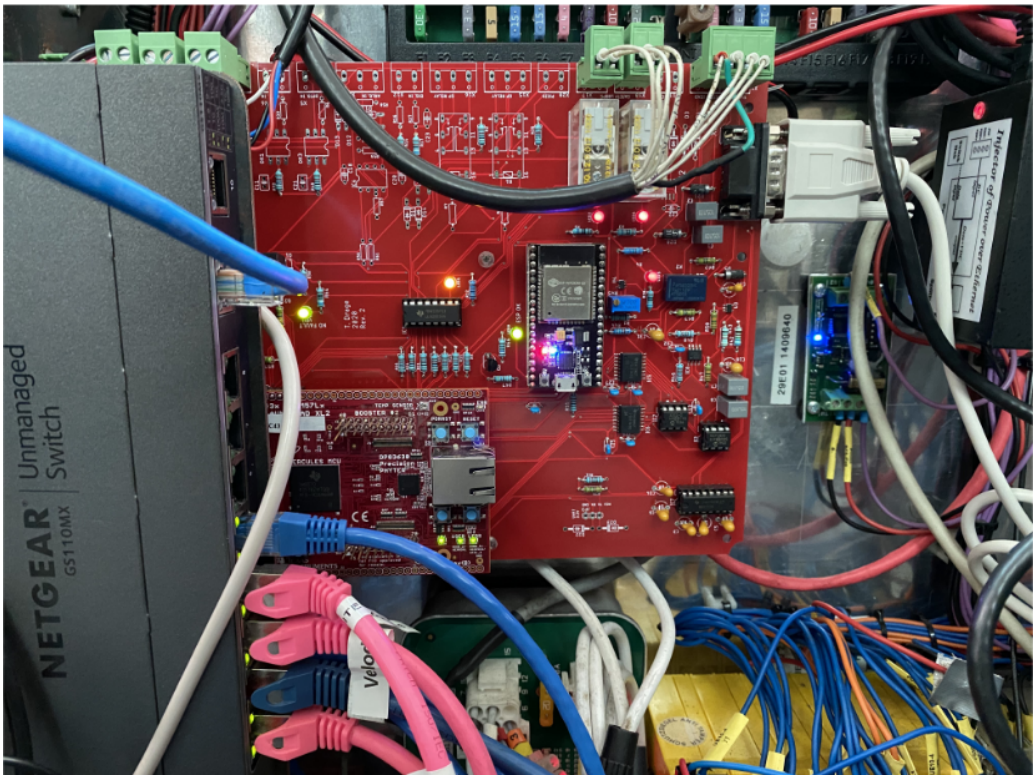


Figure 1.4: Interface board that emulates manual joystick control. Adapted from [10]

Chapter 2

Problem identification

2.1 General problem identification

Previous studies on nUWAy primarily used LiDAR-based localisation in a campus setting, achieving modest success with the integration of GNSS data. However, with UWA's acquisition of an additional shuttle bus for on-road testing in August 2022, a shift in the localisation strategy became imperative. The reason being, LiDAR's limitation in environments lacking landmarks made it unsuitable as a standalone solution for this new context.

On the other hand, GNSS is adept at precisely tracking a set path using geographic coordinates of latitude and longitude. Contrarily, LiDAR techniques depend on proximity measurements of objects in the vehicle's immediate surroundings to pinpoint their location on a map. While these techniques thrive in a campus scenario abundant with landmarks, they falter in open spaces or areas that lack distinct points of reference.

Consequently, the need arose for an alternative localisation method where GNSS emerged as a practical solution. Furthermore, on-road driving requires adherence to a set path, which in turn demands a set of different algorithms and strategies to achieve this goal.

One of the primary goals of this research was to improve the reliability and robustness of the system. One significant issue was the portability of the software stack, which was unable to keep up with the ROS2 development cycle. At the outset of this research, the ROS2 distribution on the main computer was two releases behind the latest release and so was not realising all the bug fixes and performance improvements. It was thus paramount to create a system that was portable such that migration efforts would be efficient.

Another area where improvements could be made was in the interface between the computing components and the vehicle. The interface board created by previous research efforts, which simulated joystick control, was found to be unreliable. For example, the ESP32 module responsible for Bluetooth connectivity with a game controller frequently failed to initialise. Although the shuttle bus could be driven in the vehicle's autonomous mode using the Controller Area Network (CAN) protocol, which is an automotive standard, this had yet to be tested. This would eliminate the need for the interface board and shift to an industry-standard protocol, enhancing the system's dependability. One of the research's objectives was to investigate this capability and assess its feasibility.

2.2 Scope

This project was primarily aimed at developing a software stack that can accurately follow a predetermined path of GNSS waypoints. However, due to the inherent complexities associated with AVs, several other areas must be addressed first, such as developing the vehicle interface and implementing robust software development processes.

The scope of this project was to establish a solid foundation for future research and development in the field of open-source robotics, with a specific focus on demonstrating the feasibility of GNSS waypoint driving using open-source solutions. This project aimed to provide a proof of concept for the practical implementation of GNSS waypoint driving in AVs and provide a system architecture that fulfils the portability and modular requirements.

2.3 Objectives

The main objectives behind this research were as follows:

1. **Upgrade to the latest ROS2 distribution:**
   ROS2 is still evolving and not yet as stable as its predecessor, thus keeping pace with the latest distribution becomes crucial for improving core functionalities and enhancing performance. Each distribution release introduces significant improvements over the previous ones, which makes it important to upgrade to the latest one to take advantage of the enhanced features and performance.

2. **Improve portability and maintainability of the code base:**
   To keep up with the yearly release cycle of ROS2 distributions, it is crucial to design a flexible system architecture that enables agility. This results in a maintainable code base can be achieved which is especially important for teams that may experience high turnover rates.

3. **Improve startup processes and fault isolation:**
   The startup process should be automated and easily configurable so non-technical persons can operate the system. Faults must always be promptly reported to developers as they happen so that they can be fixed.

4. **Provide software integration for RTK-enabled GNSS:**
   The GNSS positioning data must be integrated from its native format to be used within the software stack for localisation.

5. **Implement the CAN protocol for driving:**
   CAN offers the potential for a more streamlined vehicle interface that eliminates the need for an interface board. This transition would grant increased access to motor control as the bus could be driven autonomously using the CAN bus, rather than in manual mode with limitations imposed by the manufacturer.

6. **Develop a software package to follow a set of waypoints:**
   To follow a set of GNSS waypoints, software needs to be developed to integrate the data into the Nav2 stack as it is not natively supported.

Chapter 3

Literature Review

3.1 Localization

One of the primary functions of an AV is localization, which refers to the process of determining its position based on sensor data. To achieve this, AVs are equipped with a multitude of sensors that gather information about the surrounding environment. These sensors are technical devices that detect specific physical properties of the environment and their signals are managed by the platform. The sensors on AVs can differ significantly in terms of their characteristics, including their field of view, range, accuracy, resolution, and cost.

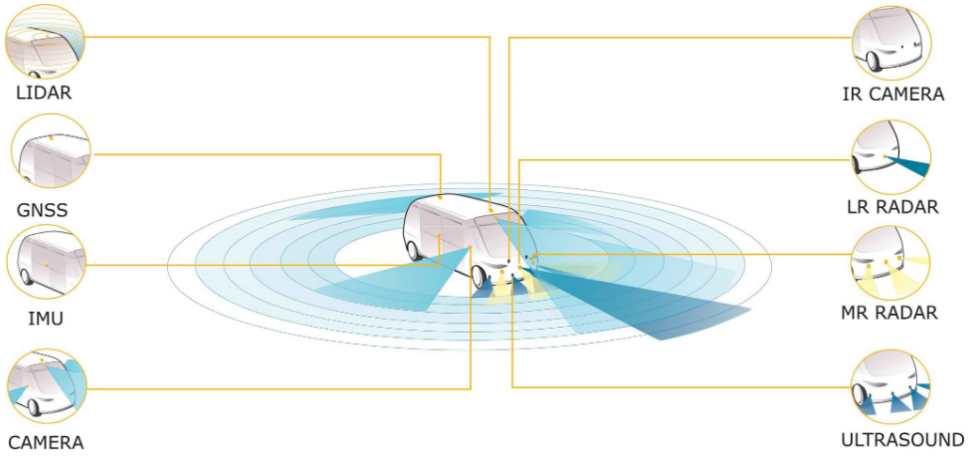


Figure 3.1: Autonomous Vehicle Sensors. Adapted from [4]

Various sensors are suitable for different contexts and environments. Since there is no human driver to decide which sensors to use, the platform must make intelligent decisions based on the context. For instance, LiDAR sensors may not be dependable in broad, open areas where the sensor cannot localize through range measurements of landmarks, in this case, a GNSS sensor would be a more suitable option. Conversely, when in an underground car park, the GNSS signal may be unreliable and using LiDAR would be more appropriate. Currently, the most commonly used sensors are GNSS, LiDAR, IMU, RADAR, and camera [4, 13].

The nUWAy shuttle bus was equipped with the SBG Ellipse-D which provides GNSS and IMU data

that provide information about orientation and position. There are eight LiDAR sensors as described in section 1.2 but only the four LMS1xx LiDARS are used for safety. When a range measurement within proximity of the vehicle is received, an emergency stop is triggered by the PLCs which causes the vehicle to stop immediately and requires an operator to reset the trip. nUWAy is also equipped with two cameras however they will not be used for the purposes of this research.

3.1.1 Global Navigation Satellite System

GNSS operates on the electronic distance measure, which involves calculating the time taken for a signal to travel through a medium. For example, RADAR is a well-known method that uses an electromagnetic pulse transmitted to an object and reflected back to a receiver to calculate the distance. By knowing the wave's velocity and the time taken for the transmit and receive process, a receiver can calculate the object's distance. GNSS employs this principle but uses one-way ranging since there is no return trip to a satellite. This makes it a passive system compared to RADAR, which is an active system.

A GNSS receiver decodes signals transmitted from satellites that contain information about their orbital parameters the receiver can use to calculate its position. The US Global Positioning System (GPS) is the oldest constellation with others including the European Galileo, Russian GLONASS, and Chinese BeiDou.

A basic GNSS receiver can achieve an accuracy of one meter which is insufficient for safety-critical applications. GNSS positioning is subject to errors such as the tropospheric and ionospheric effects. The ionosphere exhibits a dispersive nature, which means that the time delay induced by the ionosphere depends on the frequency of the signal [14]. As the signal traverses through the ionosphere, this causes the carrier wave to be influenced differently than the codes and modulations on the carrier wave. The navigation message is perceived to be slowed or delayed while the carrier wave appears to speed up. This has in effect overestimated the range determined by code observation and underestimated the range from carrier phase observation.

Another dynamic source of error is the tropospheric effect. The troposphere is refractive and is not related to its frequency. This is equivalent to a delay in the arrival of a satellite signal. This means that all signals are equally refracted and can result in the range between a receiver and a satellite being shown to be longer than it actually is [14].

Luckily, there are methods to address these error sources. One group of techniques called Differential GNSS improves accuracy by having a base station with a surveyed location estimating these error sources by the differences from satellite observations. Virtually all errors can be estimated accurately and with the aid of an algorithm called Real Time Kinematics (RTK), sub cm accuracy can be achieved [15, 9]. The corrections from the base station are transmitted to the rover via radio or the Internet through a protocol called Network Transport of RTCM over IP (NTRIP).

3.1.2 Light Detection and Ranging

LiDARs are used for localisation using algorithms like Adaptive Monte Carlo Localisation (AMCL) and Simultaneous Localisation and Mapping (SLAM) [16, 17, 18] which are probabilistic methods to determine the vehicle's position in space based on range measurements in the environment. This is in

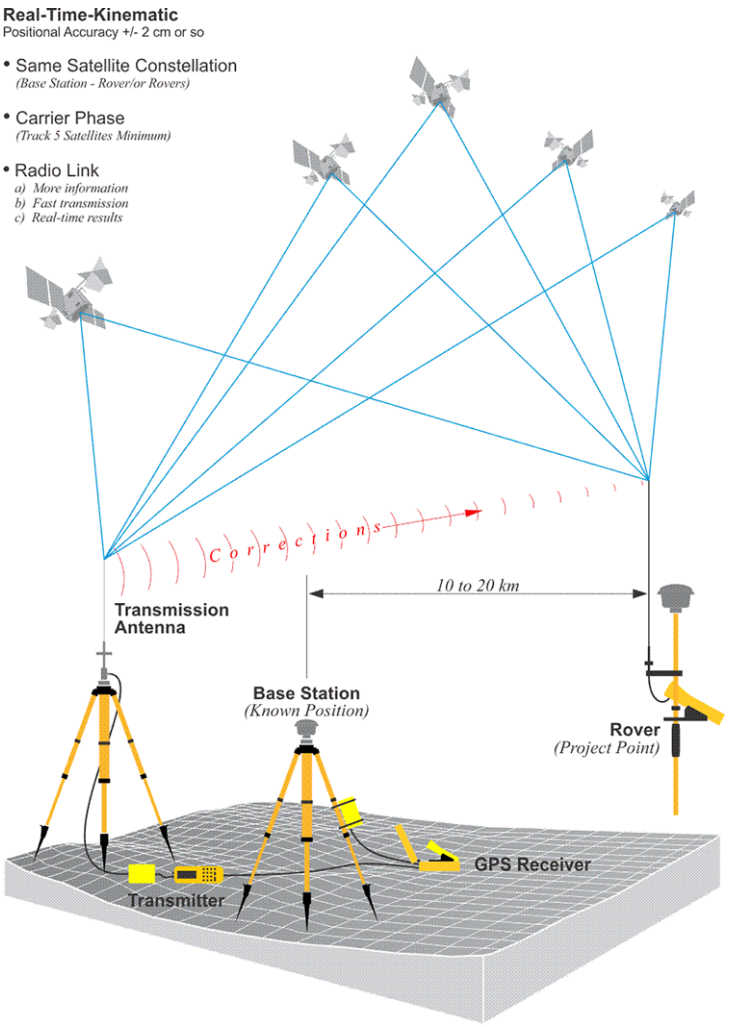


Figure 3.2: Real Time Kinematics for improved GNSS accuracy. Adapted from [14].

contrast to using satellite-based positioning [19] which provides real-time precise continuous coordinates relative to the Earth.

LiDAR sensors use a laser beam distance measurement to detect existing objects in the surrounding environment of the AV. The data from LiDAR can then be used for localisation. LiDARS are used to generate a map of static and dynamic objects on the road in 2D or in 3D for a 3D map.

LiDAR 3D sensors use a set of laser diodes mounted on a rotary device. The sensor scans the environment in a 360° horizontal and 20-45° vertical field of view [4]. LiDAR 2D sensors capture information by using a single circular laser beam to a flat surface perpendicular to the axis of rotation. An AV can use multiple LiDAR sensors with different characteristics to generate an accurate map of its surroundings. Additionally, LiDAR data can be used in conjunction with wheel odometry to accurately determine pose [20].

Inertial Navigation System

AVs are typically equipped with an IMU, which is a sensor that includes a three-axis accelerometer and a three-axis gyroscope [4]. It is intended to measure various forces that act on an AV when it is in motion amongst its three axes: lateral, longitudinal, and vertical. This information can be fused with other sensor data to improve localization. Dead reckoning is used in which the position of the AV is estimated through speed and attitude information provided by the IMU based on the position of the AV in the previous moment [21]. IMUs can suffer from accumulative drift that causes them to become less accurate over time, this can be corrected by fusing IMU data with odometry [22] or with GNSS data [23]. An Inertial Navigation System (INS) is the integration of an IMU and GNSS receiver to give position and orientation data.

Odometry

Wheel odometry provides one of the simplest forms of odometry, they are based on wheel encoders that track the number of revolutions each wheel has made. From this information, the pose of a robot can be estimated relative to a starting point [24]. Wheel odometry provides a simple and inexpensive localization method however it is insufficient on its own as it can suffer from position drift and will perform poorly on complex uneven terrain and slippery surfaces due to wheel slippage [22]. While IMUs suffer from accumulative drift over time, wheel odometry suffers from the position drift phenomenon wherein the error in the measurements accumulates over distance [22]. IMU and odometry data are often fused to counteract each other's negative characteristics through an EKF [25].

3.2 Robot Operating System 2

ROS2 is an open-source software project which provides a set of software libraries and tools for building robotic applications [26]. ROS1 was officially released in 2007 with its successor, ROS2 released in 2016 that was aimed at providing solutions to meet the challenges set forth by modern robotic systems. ROS2 is a big departure from the master-slave architecture in ROS1 and instead uses Data Distribution Services for its messaging architecture partly to address security issues present in ROS1 [27]. ROS2

provides better isolation between packages and provides an abstraction of complex robotic systems as a series of layers between hardware and software including sensors, protocols and security [28].

At its core, ROS2 is a middleware that functions as a message-passing framework and consists of nodes that can act as producers or consumers. The nodes can be categorized into two broad classes based on their relationship:

1. Publisher and subscriber: Where nodes exchange messages via topics
2. Client and server: Where clients request actions to be performed by servers to be fulfilled over a longer time frame

Publishers and subscribers provide nodes with the ability to exchange information with each other such as sensor data. Data published by one node can then be used by a subscriber for processing such as a filter or for decision making. Clients and servers provide ways for nodes to request actions to be performed by other nodes such as emergency braking or requesting a plan.

Previous research topics on the shuttle bus were conducted in ROS1 [12, 29] which have a master-slave architecture. ROS2 uses a completely different paradigm with a Data Distribution Service (DDS) which has no central point of control. The upgrade to ROS2 was performed in 2021 and other research topics have continued with ROS2 [10, 8]. This upgrade has reportedly provided improvements in the operations of the bus and maintenance of the nUWAy code base.

3.3 Navigation2

Navigation2 (Nav2) is an open-source project aimed at enabling mobile robots to navigate safely from one point to another. The project builds on the successful ROS Navigation framework, and Macenski demonstrated its effectiveness by conducting successful experiments in which a small mobile robot was safely operated alongside students on a campus [25]. Nav2 is the largest open-source project in ROS2 and provides a plugin-based architecture with implementations of various algorithms that are suitable for a wide variety of applications [30, 31]. Implementations of various algorithms for path planning, controlling, path smoothing and AMCL are provided [18] with additional libraries available for SLAM with slam_toolbox [32].

Chapter 4

Design Process

The design process followed consists of creating a list of requirements that are necessary to achieve the main objectives of the research. As this research builds on previous work, many design decisions are fixed to be compatible with work done. As a result, the decision to continue using ROS2 and Nav2 were fixed and the remaining decisions are discussed below.

At the beginning of the project, the most recent ROS2 distribution available was Humble Hawksbill, but it could only be used with Ubuntu 22.04. The main computer was running an older version of ROS2 called Foxy Fitzroy, which was scheduled to reach end-of-life in May 2023. Unfortunately, the main computer's operating system, Ubuntu 20.04, was not compatible with Humble so several options were considered:

1. Continue with the existing Foxy distribution
2. Upgrade the operating system so Humble can be used
3. Use Docker to bridge the operating system dependency

Option 1 was tested but many bug fixes were not ported to Foxy which meant that achieving the navigation objectives with GNSS localisation was not possible. Previous research in ROS1 was on the bus but ROS1 was only supported up to Ubuntu 20.04 and upgrading the operating system to 22.04 would lose access to previous code. There was a desire to have availability with the previous code so this option was not followed. This meant that another solution was required as one objective was to keep up wit the latest distribution of ROS2. Docker provided an elegant solution which also includes other benefits in system architecture.

In considering the system architecture, there were three options considered:

1. Continue with the existing system for startup and monitoring
2. Use a terminal multiplexer such as Tmux
3. Use Docker Compose

Each node in ROS2 requires a terminal to launch from. There was an existing developed system called nuway_ros2_monitor but it was difficult to diagnose faults and bugs. Several critical errors were

discovered early on in the developed package which pushed a search for a more reliable option. Option 2 was investigated and used as an interim solution using Tmux. Tmux is a terminal multiplexer and provides an automated way to start multiple processes at once which streamlines the startup procedure. While a better solution than the previous system with the monitor, it was found to be difficult to manage and sometimes failed to shutdown the system cleanly. As Docker was being used, other tools in the Docker ecosystem could be leveraged. Docker Compose is a container orchestration tool that could be used for system management. It solved all the previously encountered issues by

1. Providing an automated way to start up the system through a single file
2. Shut down the system cleanly
3. Provided access to all process logs to facilitate debugging

4.1 Requirements

The requirements can be broken down into two broad categories. The first category of system requirements relate to requirements that do not directly relate to the navigational functionality of the shuttle bus but are necessary so that they may take place. These are relevant software development methodologies and principles that improve the quality and reliability of the platform as a whole.

The second category of requirements relate to those that are directly associated with the navigational aspect of the AV. These encompass the processes and data streams required for localisation, planning, and control to move the bus safely to its destination.

4.1.1 System requirements

A1 The system must be portable across operating systems

A2 The system must be portable across ROS2 distributions

A3 The system must be modular so components can be changed without affecting the rest of the system

A4 The system must provide access to process logs for debugging

A5 The system must provide a consistent, automated way to start up

4.1.2 Navigation requirements

B1 The INS must receive a correction stream for RTK

B2 The GNSS data must have an accuracy within 30cm

B3 The navigation stack must integrate the GNSS data

B4 The navigation stack must plan a feasible path to a goal point

B5 The navigation stack must follow the planned path accurately

B6 The navigation stack shall be able to follow a set of GNSS waypoints

4.2 Constraints

4.2.1 Manufacturer locks

The shuttle bus has two motor controllers and a set of PLCs that are responsible for safety operations. These include producing an emergency stop when an obstacle is in proximity and interlocking the motors when certain conditions are not satisfied. These systems act as black boxes which makes it difficult to understand system behaviour and are locked by the manufacturer. This impacts areas like tuning the control algorithm to match the kinematics of the vehicle and understanding the CAN network. As such, tuning efforts are a best guess rather than an optimised solution.

4.2.2 Algorithm choices

Developing effective algorithms are a time-consuming task and are not in the scope of this research. Nav2 provides a limited set of algorithms and so there is a constraint on only algorithms that have been implemented in Nav2 are available.

4.3 Evaluation method

The evaluation will be conducted by measuring if the requirements in 4.1 are met. Each category of requirements will be given a weighting of 50%. Within each category, the requirements will be allocated as in table 4.1.

Requirement No.	Requirement	Weighting (%)
A1	The system must be portable across operating systems	10
A2	The system must be portable across ROS2 distributions	10
A3	The system must be modular so components can be changed without affecting the rest of the system	10
A4	The system must provide access to process logs for debugging	10
A5	The system must provide a consistent, automated way to start up	10
B1	The INS must receive a correction stream for RTK	5
B2	The GNSS data must have an accuracy within 30cm	5
B3	The navigation stack must integrate the GNSS data	10
B4	The navigation stack must plan a feasible path to a goal point	10
B5	The navigation stack must follow the planned path accurately	10
B6	The navigation stack shall be able to follow a set of GNSS waypoints	10

Table 4.1: Weighting of requirements

Chapter 5

Final Design

A large portion of this research revolved around providing a stable foundation of software processes on which AV functionality would be built on top of. This meant modifying the system architecture with three characteristics in mind, modular, portable, and maintainable.

1. **Modular:** Components of the system can be changed without affecting other parts of the system. This makes debugging issues and fault isolation in complex systems significantly easier. Additionally, future functionality can be added easily.

2. **Portable:** As the system is anticipated to follow the ROS2 release cycle, the system should be constructed in a way that migrations should be easy. It should also be portable across different platforms.

3. **Maintainable:** The system should be easy to identify changes so that faults can be identified and reversed. Documentation is required so that future developers know how to use the code and design decisions can be understood.

5.1 Migration to Docker

Docker [33] is a containerisation technology for creating, deploying and managing applications. The use of Docker allows the use of software across different environments and operating systems and also provides tools for managing containers such as Docker Compose. This means that Humble could be used through Docker and removes the dependency on the operating system, thus making the stack portable. Moving to Humble was necessary for the would provide significant improvements and stability enhancements [34, 35], those which are relevant to this research and are listed below.

1. **Improved path planning:** Major improvements to Smac Planners which showed 2-3x speed improvements and significantly higher quality paths via improved smoothers

2. **New parameters for planner:** A new parameter (use_final_approach_orientation) for the planners which helps align the vehicle to its current orientation rather than trying to achieve a previously specified one.

3. **Dynamic Composition:** Launches all Nav2 nodes in a single process instead of separately. This is useful for embedded systems that need to make optimizations due to harsh resource constraints.

4. **Respawn support:** Enables respawning of servers that crash which allows for better recovery behaviour

5. **Velocity smoother:** Smooths velocity commands from Nav2 which allows setting minimum and maximum velocities and acceleration that the controller can send, provides a way to limit the output of Nav2 to feasible and safe commands.

There are additional benefits to using Docker in both development and production. In development, it allows teams to work with a shared environment that is common across the team such that code developed in that environment will work on any other that can run Docker. In production, there is the benefit of fault isolation between containers. A failure in one container will not affect the state of other containers which was found to be an issue when testing was done only on the host system. Additionally, the use of Docker Compose for system start-up and management is done easily through a single YAML configuration file [33]. All process logs are available to the Docker daemon which helps debug faults which was not possible with the previous system.

Using Docker has the added benefit of being able to create images based on different versions of the software, a feature that was not available in the original system. The lack of separation between production and development code previously lead to instances where the working configuration was lost and not able to be recovered leading to lost productivity. Using tagged images will ensure that there will be a known working version that can be used by less experienced shuttle operators that may not know how to rebuild the code.

5.1.1 Improving code maintainability

Prior to this research, the code base was stored across two repositories. Many of the packages shared code that was duplicated which lead to issues such as failure to update code across both repositories. The unification of the repositories results in a single point of entry where a linear project history can be maintained and changes tracked easily.

The previous system was tightly coupled and any modifications or changes resulted in a cascade of failures. This was partially caused by deeply nested launch files so efforts were made to divide the launch and parameter files into distinct responsibilities. The updated architecture aimed to increase the cohesion among the system's components to increase the modularity, with each Docker container responsible for a specific purpose. For instance, the GPS container is only accountable for converting GPS data to generate the necessary transforms for Nav2. In the future, this container can be swapped with another localisation mode, such as AMCL, without causing disruption to the remaining system.

5.2 Localisation with GNSS

Previous research topics centred around LiDAR-based localisation methods like SLAM and AMCL [12, 8, 10]. Switching to GNSS-based localisation required enlisting additional software packages.

The ROS2 package robot_localization (RL) was used to provide the transforms necessary to populate the transform tree to use GNSS data within Nav2. RL provides two nodes that are used, the first is an implementation of an EKF which takes in sensor data as an input and outputs a transform. The second node is the navsat_transform_node which converts latitude and longitude in an Earth-referenced frame into x/y coordinates in the map frame of the vehicle.

For Nav2 to function properly, it is essential to establish a transform tree that connects the GNSS data from the INS to the map. While running AMCL, the shuttle bus only needs a transformation from the bus to the map. However, GNSS provides position information that is relative to the Earth, and this requires a transformation from an Earth-referenced frame to the map.

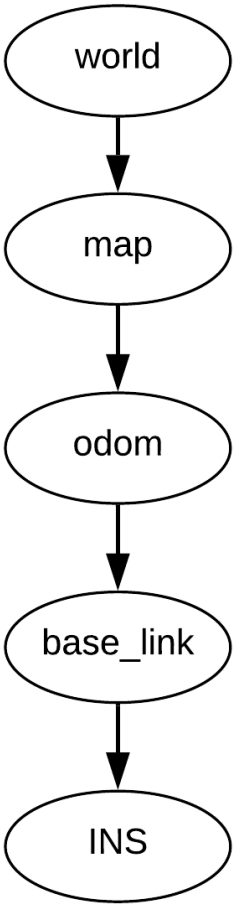


Figure 5.1: Required transform tree

In figure 5.1, each bubble represents a coordinate frame and each arrow represents a coordinate transform. The transform relates one frame to another and the tree structure necessitates that each frame has only one parent. Each transform needs to be provided to populate the tree how these are provided is discussed below.

1. world to map: This relates the vehicle's environment to the Earth-referenced frame and is

provided by the navsat_transform_node. The GNSS data from the Ellipse is sent as a /gps/fix message to the node which then projects the distance relative to a datum that is specified in the node's parameters.

2. map to odom: A transform that relates the location of the vehicle in relation to the origin of the map. This is provided by ekf_map_node which fuses the IMU sensor message from the Ellipse and the GPS odometry message from the navsat_transform_node. The former provides orientation while the latter provides position information.

3. odom to base_link: A transform that relates the odometry system to a point on the shuttle bus, commonly the centre. This is provided by the ekf_odom_node which takes in a ROS2 odometry message produced by the Ellipse.

4. base_link to INS: A static transform that relates where on the shuttle bus the INS resides. This is provided by the robot_state_publisher node.

5.2.1 Providing a datum

The sbg_driver in the updated 3.1.0 version provides an implementation of the ROS2 odometry message. One issue with the implementation is that it is projected to the first valid INS position received. This poses the problem of the vehicle starting with a random position and so would not be able to accurately localize. The sbg_driver was forked and a feature was added as a parameter to specify a datum. The datum is a point in the map with known coordinates, this was retrieved from previous research efforts which mapped landmarks around UWA's Crawley campus.

With synchronization of this datum with that of the navsat_transform_node, the vehicle has a map with known landmarks and can project its position accurately from the datum.

5.3 Navigation

This section goes through the methods conducted for navigation goals. This includes the algorithms for path planning and controlling in Nav2 and also covers the code developed to follow GNSS waypoints and implement the CAN protocol.

5.3.1 Path planner

For path planning, the Smac Hybrid A* Planner was used. This is a modified implementation of the A* algorithm in which additional enhancements to improve the computational speed and path quality are added. This algorithm was developed at Stanford and used successfully in the 2007 DARPA Urban Challenge [30]. The enhancements are done primarily through two search heuristics.

While A* associates costs with the centres of grid cells, Hybrid A* associates cost in a continuous state with each cell by taking into account the non-holonomic nature of the vehicle. This means that the heuristic prunes search branches that are not kinematically feasible and were shown to provide an order-of-magnitude improvement if the number of nodes expanded.

The second heuristic ignores the non-holonomic nature of the vehicle and computes the shortest distance to the goal with dynamic programming

The rationale for choosing the Smac Planner Hybrid is that Nav2 provides two path planners that are feasible for this research. These are the Smac Planner Hybrid and the Smac Planner Lattice. The Hybrid was designed specifically for Ackermann and car-like robots while the Lattice provides the flexibility to work with Ackermann types.

5.3.2 Controller

The control algorithm used was the Regulated Pure Pursuit (RPP) algorithm. This is an implementation of the Pure Pursuit algorithm first published in 1992 [36]. The aim of the algorithm is to follow a path generated by a planner. This is done by following a carrot which is a point within a certain radius of the vehicle. This radius is known as the lookahead distance and must be tuned for the specific application. The algorithm picks the closest point on the path and generates a spline to the carrot and the motor control required to follow this spline. In Nav2, this output is the /cmd_vel message.

The rationale for choosing the RPP controller was that within the plugins available for controllers, there are only RPP and TEB which support Ackermann drives. As the requirement is for exact path following, TEB cannot be used as this is commonly used for dynamic obstacle avoidance and would not result in the requirement being met. This means that the only feasible controller available is the RPP controller.

5.4 gps_waypoint_follower package

The gps_waypoint_follower ROS2 package was created to integrate GNSS data into Nav2. Nav2 does not natively support GNSS data and so this package is required as an adapter to convert GNSS waypoints into the coordinate frame of the vehicle to utilise the Nav2 stack.

The primary method this is achieved is through a server exposing a followGpsWaypoints service. A client calls this service by passing a list of coordinates which is transformed by the navsat_transform_node from RL into a list of x/y coordinates in the coordinate frame of the vehicle. This transformed list is then passed into the Nav2 stack by calling the followWaypoints server exposed by Nav2.

The package provides three nodes for use:

1. **Server:** The server provides the followGpsWaypoints service which calls the followWaypoints service from Nav2.

2. **Client:** The client reads in a CSV file provided as a parameter to the node and calls the server with the list of coordinates

3. **Logger:** The logger is intended to serve as a means to log the coordinates of a driven route to form the basis for waypoint following. It logs the latitude, longitude and orientation of the bus and has parameters to log based on time or distance.

5.4.1 Implementing the CAN protocol

Before the current research, the manual control of the vehicle's motors was achieved by emulating a serial interface, using an "interface" board that was developed in earlier research. This interface board was prone to errors and limited functionality of the motor. CAN is an automotive standard protocol that is used widely for communications between electronic devices in embedded systems. EasyMile provided documentation on how to interface with the motors using the CAN protocol, but upon testing, it was discovered that the documentation was outdated and initial attempts to control the motors failed.

In March 2023, a representative from EasyMile visited Perth and updated the firmware on the PLCs to match the documentation, allowing the process of transitioning the vehicle interface to the CAN protocol to proceed. To accomplish this, a ROS2 node was created in Python, called "can_bridge", which translated commands from the Nav2 stack into CAN frames that specified the speed and direction of the motors. The ability to use CAN to drive the bus provides greater control over the system as the motors are not restricted in autonomous mode and are also more reliable than the previous interface board.

Chapter 6

Results

6.1 System results

The introduction of Docker for software development operations made significant improvements to the development process.

1. The code is portable across operating systems and ROS2 distributions by using Docker
2. System startup is managed through a simple YAML file with Docker compose
3. Docker Compose provides access to all process logs for debugging
4. Using multiple containers provides a modular system architecture and allows the change of components in the future without affecting the rest of the system

6.2 Navigation results

With a correction stream using RTK provided by an AUSCORS station at Curtin University, it was found that position accuracy with GNSS was consistently under 15cm. The shuttle bus was able to accurately localize itself in the map and navigate to goal points. Algorithm tuning within Nav2 constituted the main body of work where there are three main components that were required to be tuned for the application, these are the path planner, costmap, and the controller.

6.2.1 Path planner

The optimal path has the characteristics described below:

1. Gives sufficient clearance to obstacles.
2. Favour straight motion and minimise any turning.
3. In corridors, the planner should aim to take the middle to maximise the clearance to obstacles.

A tuning guide was followed [37] and it was found that the costmap had a major impact on the quality of the paths generated. Overhanging tree branches from LiDAR mapping left artifacts as

lethal obstacles in the costmap of the vehicle. This causes the path planning algorithm to produce a sub-optimal path as shown in 6.1 where the planned path is in red and the optimal path is in green. The orange circle shows the tree branches that are mapped as obstacles that constrict the corridor such that the planner avoids them.

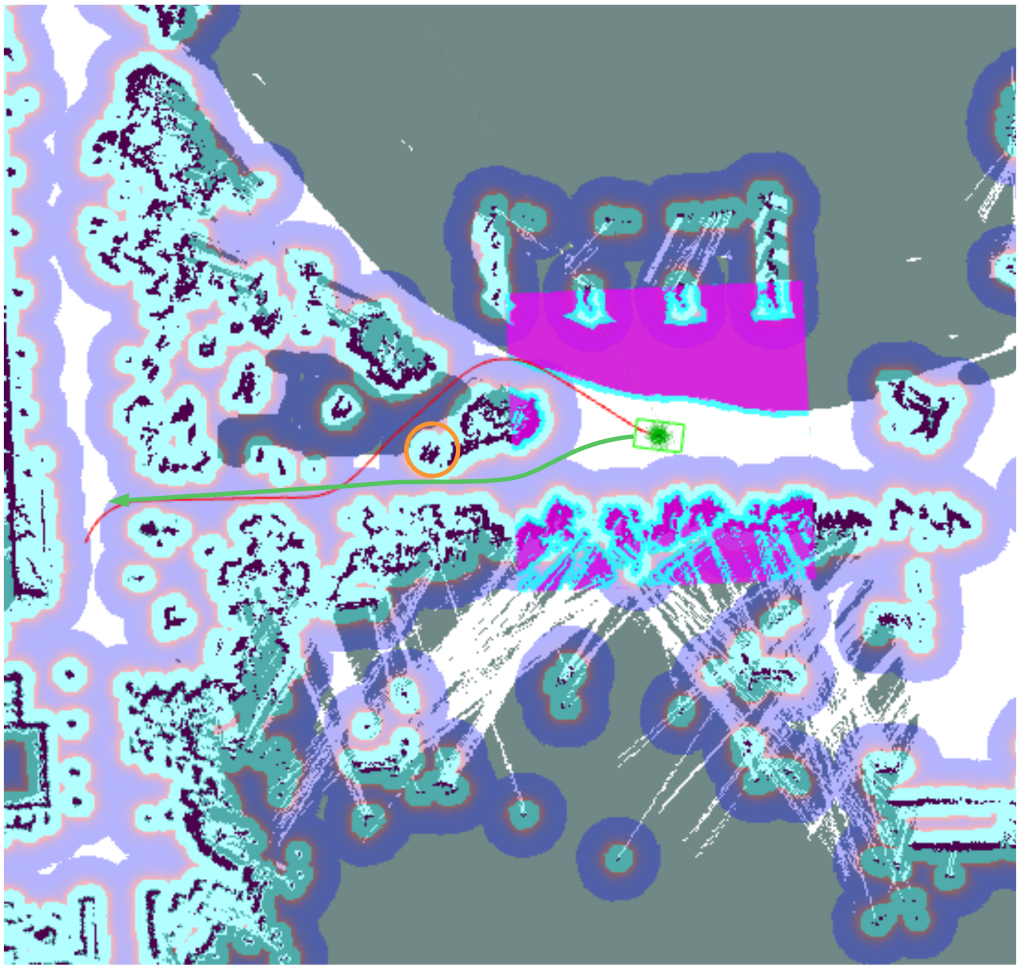
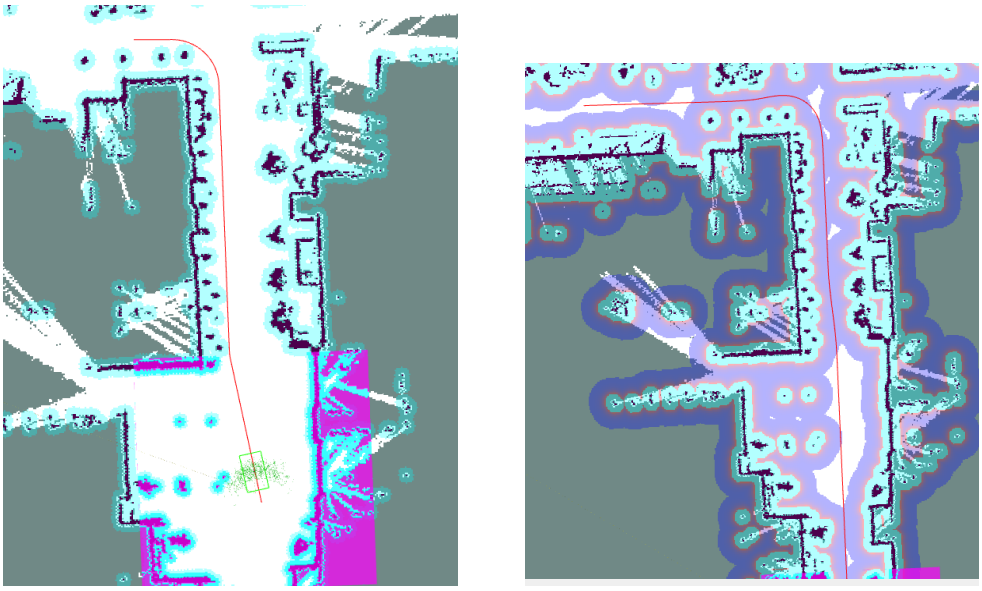


Figure 6.1: Generated path avoids the optimal route because of overhanging tree branches mapped as obstacles

Inflation radius

The costmap is the internal representation of the environment of the vehicle. There are several plugins available for the costmap, the most important one was the inflation layer. The inflation layers extend a higher cost area around lethal obstacles and can be configured with the inflation distance and the cost scaling factor. The inflation radius was tuned to push the generated path towards the centre of corridors to give sufficient clearance to obstacles. When the radius was set to a low value, the planner would create a plan that was too close to obstacles. Additionally, if the scaling factor was too low, the costs to move were too high and the planner could not calculate a feasible path. The inflation radius was set to 6.0m with a cost scaling factor of 2.0 and these parameters showed good performance

for the planner. Figure 6.2a shows the planner with a path that comes too close to obstacles while b shows the path in the centre of the corridor with a high inflation radius around obstacles.



(a) Low inflation radius **(b)** High inflation radius

Figure 6.2: Impact of inflation radius on path planning

6.2.2 Control Algorithm

The lookahead distance was the most important parameter for the controller. Tuning this parameter requires a delicate balancing act between two objectives: regaining the path and following the path. If the lookahead distance is set too small, it may cause the vehicle to oscillate along the path. On the other hand, if the lookahead distance is set too large, the vehicle may experience instability and overshoot the path (Figure 6.3).

A lookahead distance of 9.0m was found to be a good compromise between the two goals. The RPP controller performed well on straight sections however had difficulty on high curvature paths where the campus environment has sharp 90-degree turns. Additionally, the proximity of obstacles on the costmap in tight corridors often caused the controller to fail as it is designed to stop if the footprint of the vehicle is in the proximity of a lethal obstacle.

The regulation features for scaling speed around proximity to obstacles and on high curvature paths did not perform as expected and scaled the speed to minimal values such that no progress was observed in these situations. As a result, these features were turned off and velocity controlled through the velocity controller node from Nav2 to limit maximum velocities and accelerations.

6.3 Overall evaluation

The research will be evaluated against the requirements identified in section 4.1 in accordance with the method in section 4.3 for the two categories of requirements in figures 6.1 and 6.2. Using the framework developed in section 4.3, all the requirements identified for both categories have been met. The final

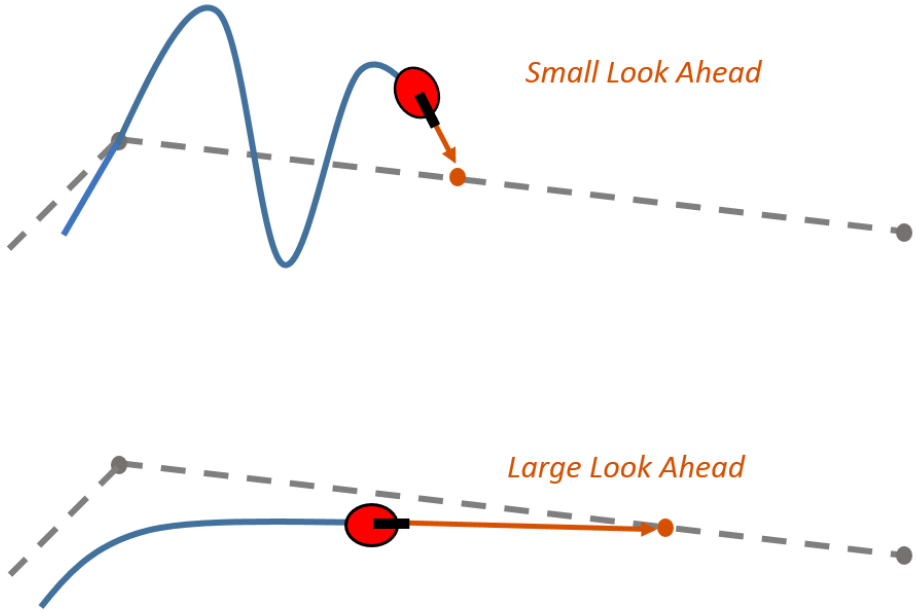


Figure 6.3: Impact of lookahead distance on the controller. Adapted from [38].

design utilises the latest ROS2 distribution and incorporates Docker to improve the robustness and portability of the system. The stack is no longer tied to the operating system and future migrations to newer distributions of ROS will be significantly easier. System management for startup, development and testing are also greatly enhanced with Docker Compose which gives full access to process logs that will aid in troubleshooting. The maintenance of the code base is improved with the unification of the two repositories which will allow greater development productivity in the future.

Requirement No.	Requirement	Met
A1	The system must be portable across operating systems	Yes
A2	The system must be portable across ROS2 distributions	Yes
A3	The system must be modular so components can be changed without affecting the rest of the system	Yes
A4	The system must provide access to process logs for debugging	Yes
A5	The system must provide a consistent, automated way to start up	Yes

Table 6.1: System requirements evaluation

GNSS localisation was shown to achieve the required accuracy for the application and was successfully integrated into the Nav2 stack to leverage the tried and tested path planning and control algorithms. The path planner, costmap, and controller were tuned to fit the system and showed moderate success although the vehicle had trouble with high curvature paths. Additionally, a ROS2 package was developed to follow a set of GNSS waypoints. Further tuning of the costmap and algorithms would enhance the performance of the system but is left to future research efforts.

Requirement No.	Requirement	Met
B1	The INS must receive a correction stream for RTK	Yes
B2	The GNSS data must have an accuracy within 30cm	Yes
B3	The navigation stack must integrate the GNSS data	Yes
B4	The navigation stack must plan a feasible path to a goal point	Yes
B5	The navigation stack must follow the planned path accurately	Yes
B6	The navigation stack shall be able to follow a set of GNSS waypoints	Yes

Table 6.2: Navigation requirements evaluation

6.4 Limitations

6.4.1 Nav2

Nav2 was designed for small mobile robots such as warehouse robots. Extending the Nav2 framework towards a passenger-carrying vehicle includes more risks and these require to be appropriately managed. Further testing over a wide range of scenarios and speeds is paramount before any production use.

6.4.2 PLC and motor controller locks

As the motor controller and PLCs are protected by the manufacturer, it is difficult to tune the navigation algorithms to match the vehicle's kinematics. With no information about the behaviours of these other components, tuning efforts are a best guess, iterative process that may not produce the best results.

Chapter 7

Future Work

There is a multitude of avenues for future work related to improving the robustness of localisation through other sensors. Additionally, software methodologies could be put in place to ensure

These include:

1. Integrate CAN odometry
2. Sensor Fusion with LiDAR
3. Continuous Integration and Development

7.1 Integrate CAN odometry

Odometry is an important source of localization data for autonomous vehicles (AVs). Wheel encoders can provide this data by estimating the distance travelled through wheel revolutions. This is particularly useful when sensor data is lost, such as in GNSS-denied environments in urban canyons. In such cases, the AV can estimate its position until sensor data becomes available again.

The shuttle bus used in the research also outputs odometry information over the CAN network, which can be integrated into an EKF to improve localization through dead-reckoning. Although initial investigations into integrating the odometry data into the EKF on the Ellipse-D were conducted during the research, time constraints and complexity prevented it from being achieved. Nonetheless, integrating the odometry data into the EKF would be a valuable addition to future research efforts.

7.2 Sensor Fusion with LiDAR

To enhance the robustness of localization, an EKF can integrate data from multiple sources, including LiDAR sensor data. By fusing data from different sources, the EKF can address situations where one data source is unreliable, such as in an urban canyon where GNSS may fail. Previous research [12, 8] has fused LiDAR with IMU data but did not successfully fuse GNSS data.

By utilizing multiple modes of localization through an EKF, a vehicle can switch between different localization modes or fuse the data from different sources to improve its position estimates. For instance, in the previous example of GNSS-denied environments, LiDAR-based localization through AMCL can

be used when GNSS is unavailable, as the two methods complement each other. Additionally, odometry information can also be fused in the EKF, as discussed earlier, to further improve the robustness of localization.

7.3 Continuous Integration and Development

Developing maintainable, high-quality software requires effective software development operations. When team members leave, the knowledge they possess regarding the technology used can be lost, making code written without proper documentation useless to new developers who are unfamiliar with the system. To address this issue, Continuous Integration and Development (CI/CD) flows can be implemented during development. This encourages developers to think more about the long-term and improve development practices by automating code testing and other processes.

CI/CD flows also help improve team onboarding processes, which can often become a bottleneck when teams change frequently. Moreover, implementing CI/CD can ensure that code standards are met, such as linting, documentation, and formatting. For example, PEP-8, pylint, and mypy are commonly used for Python, while clang is often used for C++.

Creating a suite of unit tests can significantly improve software quality and assurance by allowing the code to be verified for expected behaviour. This also facilitates automated testing, which can identify when a code change has broken the system in other areas.

Chapter 8

Conclusions

Driverless technology is an area with increasing interest and poses a serious technological challenge. It was demonstrated that an electric shuttle bus can be driven with open-source solutions using GNSS for localisation. The research contributed to the existing nUWAy project in developing software processes and evaluating different path planning and control algorithms to advance the objectives of the project.

The system architecture was remodelled to improve development efforts by introducing Docker and making the system modular, portable and maintainable. Future migration efforts to keep up with the yearly ROS2 release cycle will be significantly easier and system management through Docker Compose provides improvements in fault isolation and debugging capabilities through process logs. Overall system reliability was also improved by migrating the vehicle interface to the CAN protocol.

RTK-corrected GNSS data is integrated into Nav2 which consistently showed positional accuracy of 15cm and under. Nav2 was leveraged in the path planning and control algorithms which were tuned to operate successfully on a university campus at low speeds for exact path following. Additionally, a software package was developed to perform GNSS waypoint driving which included functionality for following and logging paths.

The robustness of localisation can be improved in the future by integrating other sources of position information like LiDAR sensor data and odometry through sensor fusion. Additionally, software processes and quality can be improved through the practice of continuous integration and development which will also increase development velocity.

Bibliography

[1] D. Holland-Letz, M. Kässer, B. Kloss, and T. Müller, "Start me up: Where mobility investments are going," *McKinsey Insights*, 2019, place: New York Publisher: McKinsey & Company, Inc tex.copyright: Copyright McKinsey & Company, Inc. Apr 17, 2019.

[2] "Autoware Overview." [Online]. Available: https://www.autoware.org/autoware

[3] "J3016_202104: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International." [Online]. Available: https://www.sae.org/standards/content/j3016_202104/

[4] C. Iclodean, N. Cordos, and B. O. Varga, "Autonomous Shuttle Bus for Public Transportation: A Review," *Energies (19961073)*, vol. 13, no. 11, p. 2917, Jun. 2020, publisher: MDPI.

[5] "The REV Project." [Online]. Available: https://therevproject.com/

[6] EasyMile, "EZ10 passenger shuttle." [Online]. Available: https://easymile.com/vehicle-solutions/ez10-passenger-shuttle

[7] "MTi-G-710 GNSS/INS." [Online]. Available: https://www.xsens.com/mti-g-710

[8] Z. H. T. Wong, "A Robust Localisation Framework for Autonomous Applications in Urban Environments," Oct. 2021.

[9] "Ellipse Series - Miniature Inertial Navigation Sensors." [Online]. Available: https://www.sbg-systems.com/products/ellipse-series/

[10] J. Castle, "Designing an autonomous navigation system for an autonomous shuttle bus with real-time obstacle avoidance," Oct. 2021. [Online]. Available: http://roblab.org/theses/2021-REV-AutonomousShuttle-Castle.pdf

[11] C. Rösmann, F. Hoffmann, and T. Bertram, "Timed-Elastic-Bands for time-optimal point-to-point nonlinear model predictive control," in *2015 European Control Conference (ECC)*, Jul. 2015, pp. 3352–3357.

[12] F. Ahmed, "Sensor fusion and pose determination on an autonomous vehicle," 2020. [Online]. Available: http://roblab.org/theses/2020-REV-ShuttleSensorFusion-Ahmed.pdf

[13] Y. Li and J. Ibanez-Guzman, "Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, Jul. 2020, conference Name: IEEE Signal Processing Magazine.

[14] J. Van Sickle, *GPS for land surveyors*. Ann Arbor Press, 1996, tex.lccn: 95007365. [Online]. Available: https://books.google.com.au/books?id=E85RAAAAMAAJ

[15] H.-J. Woo, B.-J. Yoon, B.-G. Cho, and J.-H. Kim, "Research into navigation Algorithm for unmanned ground vehicle using Real Time Kinemtatic (RTK)-GPS," in *2009 ICCAS-SICE*, Aug. 2009, pp. 2425–2428.

[16] M.-A. Chung and C.-W. Lin, "An Improved Localization of Mobile Robotic System Based on AMCL Algorithm," *IEEE Sensors Journal*, vol. 22, no. 1, pp. 900–908, Jan. 2022, conference Name: IEEE Sensors Journal.

[17] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, Sep. 2017, conference Name: IEEE Transactions on Intelligent Vehicles.

[18] S. Thrun, "Probabilistic robotics," in *Probabilistic robotics*, ser. Intelligent robotics and autonomous agents. Cambridge, Mass: MIT Press, 2005, tex.lccn: 2005043346.

[19] H. Min, X. Wu, C. Cheng, and X. Zhao, "Kinematic and dynamic vehicle model-assisted global positioning method for autonomous vehicles with low-cost GPS/camera/in-vehicle sensors," *Sensors (Basel, Switzerland)*, vol. 19, no. 24, pp. 5430–, 2019, place: BASEL Publisher: Mdpi tex.copyright: Copyright 2019 Elsevier B.V., All rights reserved.

[20] S. Zhang, Y. Guo, Q. Zhu, and Z. Liu, "Lidar-IMU and Wheel Odometer Based Autonomous Vehicle Localization System," in *2019 Chinese Control And Decision Conference (CCDC)*, Jun. 2019, pp. 4950–4955, iSSN: 1948-9447.

[21] M. Zhang, J. Yang, J. Zhao, and Y. Dai, "A Dead-Reckoning Based Local Positioning System for Intelligent Vehicles," in *2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*, Jul. 2019, pp. 513–517.

[22] S. A. S. Mohamed, M.-H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila, "A Survey on Odometry for Autonomous Navigation Systems," *IEEE Access*, vol. 7, pp. 97 466–97 486, 2019, conference Name: IEEE Access.

[23] C. Melendez-Pastor, R. Ruiz-Gonzalez, and J. Gomez-Gil, "A data fusion system of GNSS data and on-vehicle sensors data for improving car positioning precision in urban environments," *Expert Systems with Applications*, vol. 80, pp. 28–38, Sep. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417417301641

[24] J. Zwierzchowski, D. Pietrala, and J. Napieralski, "Fusion of Position Adjustment from Vision System and Wheels Odometry for Mobile Robot in Autonomous Driving," in *2020 27th International Conference on Mixed Design of Integrated Circuits and System (MIXDES)*, Jun. 2020, pp. 218–222.

[25] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The Marathon 2: A Navigation System," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 2718–2725, iSSN: 2153-0866.

[26] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, May 2022, publisher: American Association for the Advancement of Science. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abm6074

[27] V. DiLuoffo, W. R. Michalson, and B. Sunar, "Robot Operating System 2: The need for a holistic security approach to robotic architectures," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, p. 1729881418770011, May 2018, publisher: SAGE Publications. [Online]. Available: https://doi.org/10.1177/1729881418770011

[28] "Changes between ROS 1 and ROS 2," 2017. [Online]. Available: http://design.ros2.org/articles/changes.html

[29] K. Carvalho, "Autonomous Vehicle Waypoint Path Planning," Dec. 2020. [Online]. Available: http://roblab.org/theses/2020-REV-ShuttlePathPlanning-Carvalho.pdf

[30] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *AAAI Workshop - Technical Report*, Jan. 2008.

[31] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Aug. 2005, pp. 3231–3237, iSSN: 2153-0866.

[32] S. Macenski and I. Jambrecic, "SLAM Toolbox: SLAM for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, May 2021. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.02783

[33] "Docker Compose overview," May 2023. [Online]. Available: https://docs.docker.com/compose/

[34] ROS2, "Foxy to Galactic — Navigation 2 1.0.0 documentation." [Online]. Available: https://navigation.ros.org/migration/Foxy.html

[35] ——, "Galactic to Humble — Navigation 2 1.0.0 documentation." [Online]. Available: https://navigation.ros.org/migration/Galactic.html

[36] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-92-01, Jan. 1992.

[37] "Tuning Guide — Navigation 2 1.0.0 documentation." [Online]. Available: https://navigation.ros.org/tuning/index.html

[38] "Pure Pursuit Controller - MATLAB & Simulink - MathWorks Australia." [Online]. Available: https://au.mathworks.com/help/nav/ug/pure-pursuit-controller.html