GERMAN - MONGOLIAN INSTITUTE
FOR RESOURCES AND TECHNOLOGY

Street cleaning robot algorithm

Bachelor Thesis

Submitted by

Munkhtur Ganbat

Supervisor 1/Examiner 1	Prof. Dr. Odbileg Norovrinchen
Supervisor 2/Examiner 2	M. Sc. Myagmarjav Bold

Nalaikh, Ulaanbaatar
May 11, 2023

Statutory Declaration

Ganbat, Munkhtur	15667825826119
Last Name, First Name	Student ID Number

I hereby affirm in lieu of an oath that I provided the submitted bachelor thesis

Street cleaning robot algorithm

I did not use any sources other than those stated. In case that the work is additionally submitted on a data medium, I declare that the written and the electronic form are completely identical. The work was not submitted in the same or similar form to any examination authority.

Nalaikh, 5/11/2023	
Place, Date	Signature

Contents

Abstract

This thesis presents a street cleaning robot algorithm that efficiently plans the shortest path between points while considering dynamic obstacles. The proposed algorithm is designed to update the planned path in real-time based on the latest sensor data. The simulation and prototype demonstrate the effectiveness of the algorithm in navigating through a complex environment while avoiding obstacles. The results show that the proposed algorithm can work in outdoor environments in Mongolian different climates. The simulation and prototype results confirm the effectiveness of the algorithm. Furthermore, since these types of devices have not been used in Mongolia before, this research presents a new possibility for their application in the region.

Acknowledgement

I would like to express my sincere gratitude to my first supervisor, Prof. Dr. Odbileg Norovrinchen, for his invaluable guidance, unwavering support, and encouragement throughout my research journey. His expertise, insights, and critical feedback have been instrumental in shaping the direction of my thesis and in making it a success.

I would also like to extend my heartfelt appreciation to my second supervisor, M. Sc. Myagmarjav Bold, for his exceptional mentorship and assistance. His extensive knowledge, constructive criticism, and constructive suggestions were of great help in making my research more comprehensive.

Furthermore, I would like to acknowledge the support of the faculty members of the department and my friends for their constant encouragement, motivation.

Finally, I would like to thank all the participants of my research who generously shared their time and experiences, without whom this thesis would not have been possible.

List of figures and tables

Chapter 1. Introduction

Street cleaning has been an essential aspect of urban maintenance for centuries. The practice of cleaning streets dates back to ancient civilizations, where streets were swept manually using brooms or brushes. Over time, street cleaning has become more mechanized, with machines such as street sweepers and vacuum trucks being developed to assist in the process. However, these machines still require human operators, and their efficiency is limited by their size and maneuverability.

In recent years, there has been a growing interest in developing autonomous street cleaning robots that can operate without human intervention. These robots have the potential to increase the efficiency of street cleaning operations while reducing the need for human labor. However, the development of such robots requires advanced algorithms that can efficiently navigate through complex urban environments while avoiding obstacles.

The thesis is aimed at implementing algorithms for a street cleaning robot that can operate autonomously and efficiently navigate from one point to another location while avoiding obstacles. Using various sensors, we will detect the environment and gather information, which will then be processed by a microcontroller that controls the robot based on the implemented algorithms. In order to facilitate the robot's path planning and obstacle avoidance capabilities, we will develop a specific algorithm tailored to our particular environment. This chapter will present the problem statement and research questions related to our thesis.

1.1 Problem Statement

Autonomous cleaning robots are becoming increasingly popular in urban environments due to their ability to provide a more efficient, cost-effective, and sustainable solution to waste management. These robots operate continuously, without requiring breaks, which increases their efficiency and allows them to cover more ground in a shorter amount of time than human cleaners. In addition to their increased efficiency, they also improve sanitation by removing waste and

debris more frequently and thoroughly than manual cleaning methods, which helps to prevent the spread of disease and bacteria. Autonomous cleaning robots also reduce the risk of accidents or injuries to human cleaners by removing the need for them to work in hazardous environments.
In our case, we aimed to develop robot algorithms and it's coding that each of them works for specific tasks such as cleaning snow, garbage, sanitizing the ground and so on. That means we can change the algorithms for specific usages.

1.2 Research Questions

In this thesis, we delve into two key components of the algorithms that we have developed. The first one is an efficiently designed path finding algorithm that enables a robot to navigate from one location to another. The second component is a dynamic object avoiding algorithm that allows the robot to avoid obstacles in its path.

To prioritize these algorithms, we must first answer some crucial questions. These questions help the robot determine its current position and its desired destination. The first question is "Where am I?" This question allows the robot to determine its current location in its environment. This is a crucial piece of information that forms the basis for the path-finding algorithm. Once the robot knows where it is, it can then move on to the next question.

The second question is "Where am I going?" This question helps the robot determine its destination, which is also an important input for the path-finding algorithm. Knowing the destination allows the robot to plan the most efficient path to get there.

The third question is "How to avoid this obstacle?" This is where the dynamic object avoiding algorithm comes into play. As the robot moves towards its destination, it must constantly scan its environment for obstacles that may impede its progress. By detecting these obstacles, the robot can use the dynamic object avoiding algorithm to plan a new path around them and continue towards its destination.

Overall, the combination of these two algorithms allows the robot to navigate through its environment efficiently and safely, making it a valuable tool in a variety of applications.[4]

Chapter 2. Design and Methodology

As outlined in the research questions, our attention must be directed towards two distinct factors.

Firstly, the robot's locomotion requires precise and up-to-date knowledge of its spatial coordinates to avoid moving randomly without any spatial awareness. Therefore, we need to ensure that the robot is provided with accurate location information every time when it moves. For this reason, the robot has to have specific sensors and its low level of data processing follows.

Secondly, even though we provide the robot with the data of the specific environment and the certain goal points. The robot must be planning his path on his own while it’s avoiding obstacles. Usage of high level software is required in order to efficiently and effectively to estimate the path.

Within this chapter, we’ve introduced here about the design solutions of outdoor robots.
And included its specific hardware and software design possibilities in detail at the beginning.

2.1 Hardware design

When designing a robot, the physical components are essential for the algorithm to function correctly. These include sensors, motors, drivers, and other hardware components that are required to execute tasks successfully. In our design, we mainly used Arduino devices because they are a popular choice and readily available for robotic development. Arduino devices are friendly for users, easy to program, and accessible to beginners, hobbyists, students, and educators.

One of the significant advantages of Arduino devices is the wide range of compatible sensors and modules available. GPS modules, accelerometers, gyroscopes, cameras, and motor controllers are among the components that can easily integrate with Arduino devices. This allows users to create customized robots with specific functionalities.

Arduino devices are also cost effective compared to other microcontrollers, making them an excellent choice for hobbyist and educational projects where cost is a factor. Additionally, Arduino devices are open-source, which means that the hardware and software designs are freely available for modification and customization. This allows users to create their own versions of Arduino boards or customize the software to meet specific requirements.

Furthermore, the Arduino community is a valuable resource for robot developers. The community comprises a large and active group of users who share knowledge, tutorials, and code snippets. This community can be a great resource for beginners who need help getting started or for more experienced users who want to collaborate with others on larger projects.[1]

In this section, we divided our possible hardware design into three distinct categories based on their specific purpose. These categories are the hardware responsible for location determination, for data processing, and any additional hardware components that are crucial to the overall operation of our robot.

2.1.1 Location determining hardwares

1. The GPS

The GPS is short for "Global Positioning System," a navigation system that utilizes satellites to provide users with location and time data, even in adverse weather conditions. It is used in different modes of transportation such as planes, ships, cars, and trucks, and is critical for both military and civilian purposes worldwide. With GPS, users can obtain continuous real time positioning, navigation, and timing services in three dimensions across the globe[3]

The GPS operates on the principle of trilateration, using distance measurements to satellites to determine a receiver's location on Earth. The receiver needs signals from at least three satellites to calculate its location, but a fourth satellite is used as a backup and to improve accuracy. The gps consist of three segments

- Space segment: The space segment refers to the GPS satellites orbiting the Earth that transmit signals to GPS receivers on the ground. There are currently over 30 GPS satellites in orbit.
- Control segment: The control segment is responsible for monitoring and controlling the GPS satellites. It consists of a network of ground-based control stations that track the satellites and make adjustments to their orbits and clocks as needed.
- User segment: The user segment is made up of the GPS receivers that individuals and organizations use to obtain location and time information from the GPS system. Which means we belong to the user segment of GPS.[2]

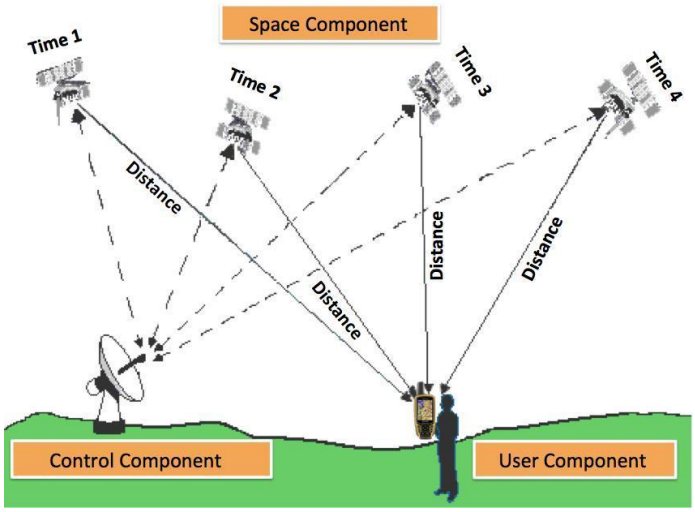

Figure 1:GPS components[47] https://trakkitgps.com/how-gps-works/

There are various types of GPS modulus possibilities when designing outdoor navigation robots. When selecting a GPS module, important factors to consider

- **Size:** Size is important for technical parameters like lock time and accuracy. For tracking applications, choose the smallest module without compromising accuracy or response time.
- **Update Rate:** The update rate of a GPS or GNSS module denotes the frequency at which it calculates and transmits its location data. The typical rate for such devices is 1Hz, equivalent to once per second. However, for faster vehicles or high-speed applications, update rates of 5-10 Hz can be considered. Nevertheless, such higher rates are not necessary in most real-life scenarios.

- **Communication Interface:**The communication interface utilized to interact with a GPS module plays a vital role in its functionality. Among the various interface types available, the Serial/TTL and USB interfaces are the most commonly employed methods for GPS module communication. It is imperative to choose an interface that is compatible with the device's capabilities and the application's requirements to achieve optimal performance.
- **Communication Speed/Baud Rate** The baud rate, which determines the speed of data transfer between the microcontroller and GPS module, is a crucial factor in the communication interface. In the case of serial interface, the baud rate is used to measure the rate of data transmission. Higher baud rates facilitate faster transfer of GPS data to the microcontroller, resulting in quicker processing of location information. Therefore, selecting an appropriate baud rate is essential to ensure efficient and accurate communication between the GPS module and the microcontroller.
- **The Navigation Sensitivity dBm figure** tells us how well the GPS module can receive signals from satellites. If the number is higher, it means the module is better at picking up signals from satellites.
- **Power Requirements** Selecting the appropriate GPS module requires consideration of its working voltage and power consumption. This is particularly important when the module is battery powered, as it is necessary to conserve battery life for optimal operation. Most commonly used GPS modules consume an average power of around 30mA at 3.3V.
- **Number of Channels** The number of frequencies/channels a GPS module can check affects the time to first fix (TTFF). More channels checked at once will result in faster fixing. Once a fix is obtained, some modules shut down the extra channels to conserve power. 12-14 channels are sufficient for tracking if a longer TTFF is acceptable.
- **Accuracy:**The accuracy of a GPS module is inversely proportional to the minimum distance it can calculate. Thus, a lower minimum distance results in higher accuracy. Typically, GPS modules can determine a location within 30 seconds with an accuracy of around +/-10m. However, some modules can achieve even higher accuracy levels of +/-1cm by being equipped with more advanced hardware and software.
- **Antenna:** GPS modules receive signals from satellites located approximately 12,000 miles away. To ensure optimal performance, a clear path between the antenna and most of the sky is necessary. While weather conditions such as clouds and snowstorms don't affect the signal, obstacles such as trees, buildings, mountains, and roofs can create interference that affects GPS accuracy. Therefore, selecting the right antenna is critical.

- **Chipset:** GPS chipsets are responsible for performing various tasks, such as calculations, antenna circuitry, power control, and user interface. Despite their small size, they are capable of performing complex functions. These chipsets are compatible with various antenna types and are available in different types such as ublox, SiRF, and SkyTraq. The choice of the chipset depends on factors like power consumption, acquisition times, and hardware accessibility, and choosing the right one is crucial for the application.
- **Price:** Module price is important to consider and should be kept within budget.
- **Time to First Fix(TTFF):** the amount of time a GPS receiver takes to obtain its first accurate position fix after being turned on or reset.[9]

2. Ultrasonic sensor

Ultrasonic sensors are commonly used for distance measurement and object detection. They work by emitting a high-frequency sound wave and measuring the time it takes for the wave to bounce back after hitting an object. This time measurement can be used to calculate the distance to the object.The speed of sound in air is approximately 343 meters per second at standard temperature and pressure (STP). The distance to an object can be calculated using the following equation:

$$d = \frac{v \times t \times \ cos\theta}{2}$$

d- distance [m]

v- speed of sound [m/s]

t- time of flight [s]

θ- angle between horizontal and path taken [degree]

where time of flight is the time it takes for the ultrasonic wave to travel to the object and back to the sensor.Ultrasonic sensors typically operate at a frequency range of 20 kHz to 200 kHz. The wavelength of the sound wave can be calculated using the following equation:

$$\lambda = \frac{v}{f}$$

λ- wavelength [m]

v- speed of sound [m/s]

f- frequency [1/s]

The size and shape of the ultrasonic sensor and the frequency of the sound wave can affect the directionality and sensitivity of the sensor. For example, a smaller sensor will have a narrower beam angle and greater directional accuracy, but may have a shorter range and lower sensitivity.

Ultrasonic sensors can also be affected by factors such as temperature, humidity, and air pressure, which can alter the speed of sound in the environment and affect the accuracy of the distance measurement.[10],[11],[12]

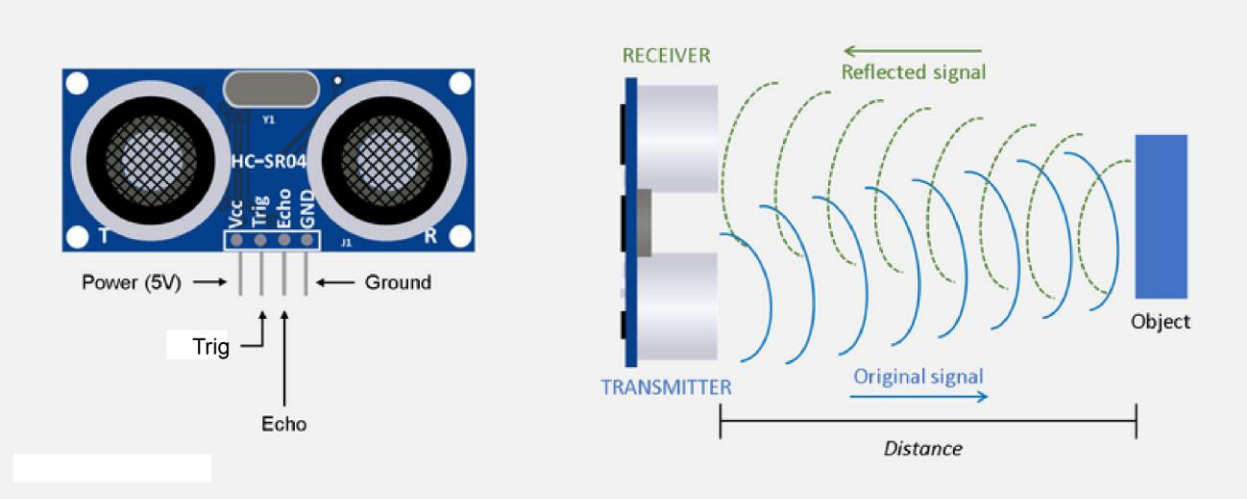

Figure 2. Ultrasonic principle[48]

Drawbacks of ultrasonic sensors include the effects of pressure, temperature, and humidity on air, which may impact the accuracy of calculations. Additionally, objects with sharp edges may not produce good echoes, and there may be a blind zone of a few centimeters (three centimeters or less) if the object is too close to the sensor. Moreover, ultrasonic sensors cannot operate in a

vacuum space, and the contribution to attenuation may result in various phenomena such as absorption, reflection, scattering, refraction, diffraction, interference, and divergence.

On the other hand, ultrasonic sensors offer several advantages. For instance, physical contact is not necessary for detection, and sound waves do not affect light and color. Furthermore, ultrasonic sensors can function effectively in diverse environments, including air, solids, water, or gas. They are also compact in size compared to other types of sensors, and have a relatively lower cost. Additionally, the error rate is slight, typically around ±3 centimeters (increasing with range).

Choosing the right sensor is important as there are many different types available in the market. The decision should be based on several factors including accuracy, range, distance, usability, cost, and other relevant criteria, all of which will affect the suitability of the sensor for the specific application.[13]

3. LIDAR Sensor

Lidar is a highly advanced sensing technology that uses laser pulses to measure distances and create 3D maps of the surrounding environment. Its ability to provide highly accurate depth information is critical for applications such as object detection and tracking, and it achieves this by using a photodetector, such as a Single Photon Avalanche Diode (SPAD) or Avalanche Photodiode (APD), to detect the reflected laser pulses from obstacles. Key advantages of LIDAR include high accuracy and reliability, versatility in different environmental conditions, and the ability to create highly detailed 3D maps of the environment.

One of the main reasons why lidar is a better choice as a standalone vision sensor is because of its ability to provide highly accurate depth information. This is critical for applications such as object detection and tracking, where the ability to accurately measure distances is essential. In addition, lidar can be combined with other sensing technologies such as cameras and radar to achieve a highly robust and reliable sensing system, which is a key area of ongoing research and development in the field of autonomous driving.

The working principle of lidar involves the use of laser technology to remotely sense and map the environment. A laser emits a highly focused and coherent beam of light, which is directed towards

objects in the environment. The reflected light is then detected by a photodetector, also known as a receiver, and the distance to the object is calculated based on the time it takes for the laser pulse to travel to the object and back to the receiver.

The distance calculation is based on a simple equation of motion, where the speed of light in air and the total time of flight of the laser pulse are used to determine the distance to the object. This allows for highly accurate and precise measurements of distances in the environment, making lidar a valuable tool for a variety of applications.

By employing a straightforward equation of motion and replacing the values of the speed of light in air and the total time of flight, it becomes effortless to determine the distance of an object from lidar.

The measured range of lidar can be found by the following equation.

$$R = V \times t \times \frac{1}{2n}$$

R = Measured range of lidar

v = speed of light in m/s

n = refractive index

t = time gap between transmission and reception of laser[14]

Material	Index of Refraction (n)
Vacuum	1
Air	1.000277
Water	1.333333
Ice	1.31
Glass	1.5

Diamond	2.417

Table 1. Index of refraction[57]

The three main components of LIDAR sensors are transmitter, receiver and time & signal controlling circuit. And there are many types of LIDAR depending on its laser source, photo detector, and timing & signal processing.

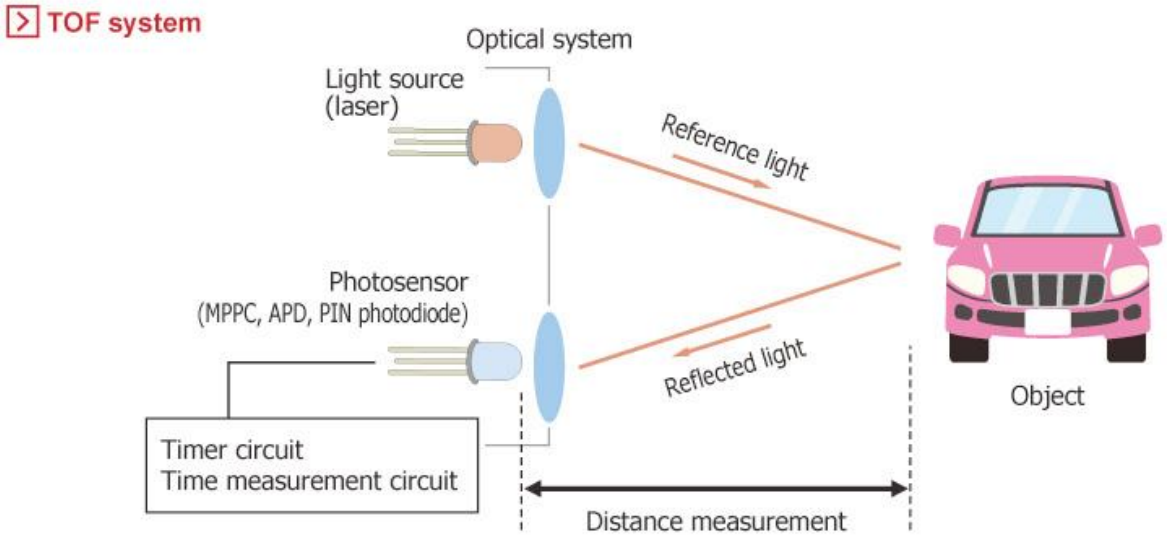

Figure 3. Lidar principle[49]

When choosing a lidar sensor, there are several important factors to consider to ensure that it meets the specific requirements of the application. For example: Range and accuracy, field of view, environmental conditions, data output, cost and power consumption, Integration with other sensors

4. Compass sensor

The magnetic field of the earth is detected and measured by an electrical sensor called a compass sensor for robots. It can reveal information about the robot's heading or orientation in relation to

the magnetic field of the planet. The sensor detects variations in the magnetic field and transforms these variations into electrical signals that the robot's control system can understand.

Compass sensors are useful for a number of robotic tasks, including localization and navigation. A robot may identify its heading and direction of movement by utilizing a compass sensor, which is helpful for traveling to certain areas and avoiding obstructions. For instance, a street cleaning robot with a compass sensor can use the sensor to follow a predetermined path and steer clear of barriers like walls or buildings.

Compass sensors of many kinds, including Hall effect sensors, magnetoresistive sensors, and fluxgate sensors, are available for robotics. The most prevalent kind of compass sensor is the hall effect sensor, which is frequently utilized in robotics applications. They are affordable, simple to use, and offer accurate magnetic field readings. Known for their excellent sensitivity and precision, magnetoresistive sensors are another type that is frequently employed. Fluxgate sensors are more expensive and sophisticated than other compass sensor types, but they produce extremely accurate magnetic field readings and are frequently utilized in high-end robotics applications.

In general, a compass sensor is a helpful and necessary part for many different kinds of robots, especially those that need accurate localization and navigational abilities. A compass sensor can significantly enhance the robot's performance and efficiency by giving precise information about the robot's orientation and heading.[56]

5. IMU Sensor

An IMU, or inertial measurement unit, is an electronic device that utilizes accelerometers, gyroscopes, and sometimes magnetometers to measure and provide information about a body's specific force, angular rate, and orientation. The addition of a magnetometer makes it an IMMU. IMUs are commonly used in the navigation and control systems of modern vehicles, including aircraft, unmanned aerial vehicles (UAVs), missiles, and spacecraft such as satellites and landers. They can also be integrated with GPS devices to allow for operation when GPS signals are not available due to factors such as interference or being in enclosed spaces.[15]

Accelerometers measure the force acting upon a proof mass, with two common implementations being open loop and closed loop. The accuracy of acceleration measurements is influenced by

thermal sensitivity and cross-axis sensitivity. Positioning using acceleration data is not possible for robot navigation due to the poor signal-to-noise ratio at low accelerations.

Gyroscopes sense the angular rate of the platform with respect to the inertial frame, with mechanical gyroscopes using rotating components to determine the angular rate. The main error sources for mechanical gyroscopes are mass imbalance effects, temperature sensitivity, and sensitivity to external magnetic fields. Optical gyroscopes use the relativistic Sagnac effect to determine the angular rate, and their accuracy is influenced by glass flow, polarization, and electric effects. High-accuracy gyroscopes are still expensive, and they are mainly used in commercial airlines. Gyroscopes are often used to minimize orientation errors because any small orientation error causes a growing lateral position error.

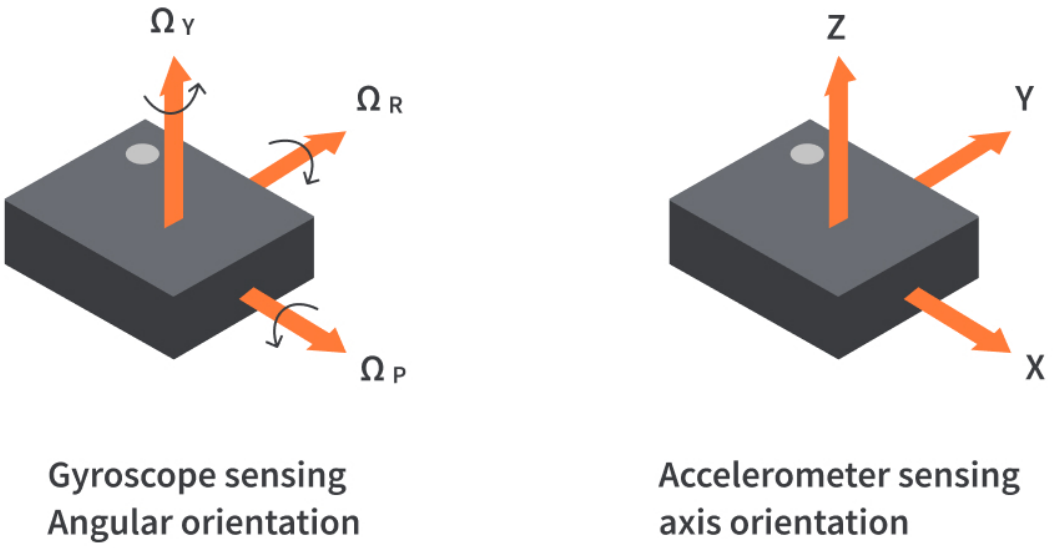

Figure 4. Gyroscope and Accelerometer sensing

Using IMUs for navigation has a significant drawback because they tend to accumulate errors over time. This is because the navigation system continuously integrates acceleration over time to determine velocity and position (known as dead reckoning). Even tiny measurement errors can accumulate over time, resulting in an increasing gap between the system's estimated location and the actual location, also known as "drift". The integration process causes a linear growth in velocity and a quadratic growth in position due to a constant error in acceleration, while a constant error in attitude rate (gyro) results in a quadratic growth in velocity and a cubic growth in position.

2.1.2 Data Processing hardwares

1.Microcontroller(MCU)

A microcontroller is a small computer that is incorporated into a device to control a specific function. It achieves this by processing data received from its input/output (I/O) peripherals through its central processor. The microcontroller temporarily stores this data in its data memory, and the processor accesses it to apply the appropriate instructions stored in its program memory. The microcontroller uses its I/O peripherals to communicate and execute the desired action. Microcontrollers are widely used in various systems and devices, and multiple microcontrollers may be used within a single device to handle different tasks.

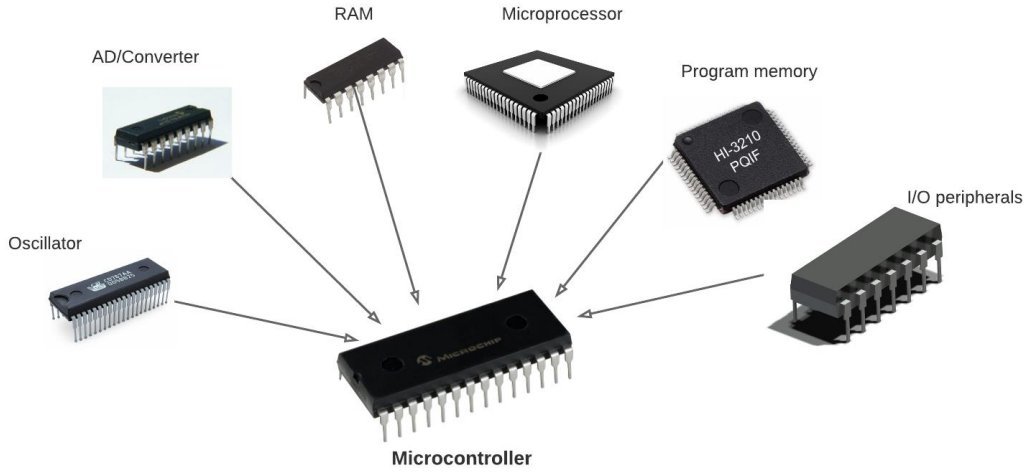

Figure 5. Microcontroller parts[51]

The core components of a microcontroller consist of three main elements:the processor, memory, and input/output (I/O) peripherals.

The processor acts as the device's brain and executes instructions that direct the microcontroller's function. This involves performing basic arithmetic, logic, and I/O operations, as well as data transfer operations to communicate commands to other components in the larger embedded system.

Memory is used to store the data that the processor receives and uses to respond to instructions. A microcontroller has two main memory types: program memory, which stores long-term

information about the instructions the CPU carries out, and data memory, which is required for temporary data storage while the instructions are being executed.

I/O peripherals are the input and output devices that interface with the processor to the outside world. Input ports receive information and send it to the processor in binary data format. The processor receives this data and sends necessary instructions to output devices that execute tasks external to the microcontroller.

Besides these core components, other supporting elements of a microcontroller include an Analog to Digital Converter (ADC) and a Digital to Analog Converter (DAC), which allow the processor to interface with external analog devices, such as sensors. The system bus connects all components of the microcontroller, and the serial port is one type of I/O port that allows the microcontroller to connect to external components. These supporting components are essential to the microcontroller's overall function as they enable communication between the microcontroller and the external world.

When selecting a microcontroller, there are several criteria to consider and their importance may vary depending on the specific application.

- **System Requirement**

The design process for a microcontroller-based system should begin with a clear understanding of the application requirements. This involves deciding whether a single chip MCU will suffice or additional peripherals will be needed. The selection process typically involves choosing between 4-bit, 8-bit, 16-bit, or 32-bit microcontrollers depending on the application needs. While developing code for 4-bit architectures can be challenging due to the limited arithmetic capabilities and handling of 4-bit instructions and data widths, 8-bit microcontrollers are commonly used for embedded applications due to their availability and low cost. The Atmel AVR series is an example of an 8-bit microcontroller that offers high performance and speed. If more processing power is required, 16-bit or 32-bit microcontrollers can be selected, with vendors offering cost-effective 32-bit devices. When selecting a microcontroller, the availability of on-chip peripherals such as timers, serial interfaces, ROM, RAM, A/D converter, and D/A converter should also be considered. The number of I/O ports should be sufficient for the application needs, but not excessive to avoid unnecessary costs.

- **Memory Architecture**

Volatile memory: This type of memory is used to store temporary data that is lost when power is turned off or reset. The most common type of volatile memory used in microcontrollers is Random Access Memory (RAM).

Non-volatile memory: This type of memory is used to store data that is retained even when power is turned off or reset. The most common type of non-volatile memory used in microcontrollers is Flash Memory, which is used to store the program code.

Accessing memory: This refers to the process of reading from or writing to memory. In microcontrollers, accessing memory is done through a memory access mechanism, which can be divided into two categories:
-Direct Memory Access (DMA): This mechanism allows data to be transferred directly between peripherals and memory without CPU intervention.
-Central Processing Unit (CPU) Memory Access: This mechanism involves the CPU reading from or writing to memory. This can be done using instructions such as Load and Store.

Overall, selecting the appropriate type of memory and accessing mechanism is an important consideration when choosing a microcontroller for a specific application. The amount of memory required depends on the complexity of the application and the specific needs of the project.

- **Availability and long-term availability:**

The microcontroller should be readily available and be produced for a reasonable amount of time to ensure long term availability. In some cases, the long-term availability of the microcontroller can be more important than the initial cost.

- **Size**

The size of an IC can be an important consideration when developing a system. If a system only requires an IC with 15 I/O pins, it would be unnecessary to use a larger IC with 32 I/O pins. By selecting an appropriately sized IC, the physical space required to implement the system can be minimized. Therefore, for certain applications, the physical size of the IC can be a crucial factor to consider.

- **Compatibility**

It is possible to modify or improve the functionality of a system without incurring significant additional costs. This can be achieved by either changing the software or replacing an IC(Integrated Circuit) with another one that is pin and function compatible. By doing so, the system

can be upgraded or altered without the need for extensive modifications or redesigns, making it a cost effective solution.

- **Functionality Testing**

In order to ensure that the overall system functions correctly, it is necessary to test the MCU's function before buying it or developing the entire system.

- **Power Management**

Generating more power leads to increased heat dissipation, resulting in energy being wasted. The battery's lifespan is determined by the power consumed by the system. As devices become smaller, machinery size decreases, and their placement within the design becomes more compact. As a result, the devices become more sensitive to heat generated by the MCU and other connected peripherals. It is the responsibility of an engineer to prioritize determining the necessary power required for the application's clock speed. Devices that use read-only memory (ROM) can operate on very low voltages, around 0.9v. For Atmel AVR devices, a Flash-based microcontroller can be operated at voltages as low as 1.8V. Most microcontrollers have power management features, such as power down, idle, and sleep modes, and careful consideration should be given to maximize the use of intelligent power management systems to reduce power consumption.

- **Availability of Development Support**

When selecting a microcontroller, it is important to consider the availability of an assembler, debugger, code-efficient C compiler, emulator, and technical support. There is a growing trend towards programming in high-level languages such as C, which allows for more portable code and libraries, making it easier to use different microcontroller families. Choosing appropriate hardware and software development tools is also crucial when selecting a microcontroller. An integrated development environment (IDE) can aid in development efforts by providing project management tools. The IDE allows for the creation of source files and organizing them into a project, creating a database for many devices.

- **Cost**

The cost of the microcontroller is always a consideration, and it must be balanced against the desired functionality and performance.[26-28]

2.Motor driver board

A motor driver is a device used to regulate the movement of a motor, but it cannot function alone and must be connected to a microcontroller. The reason for this is that motors and

microcontrollers operate on different voltage ranges, and motors require a higher current level than microcontrollers.

To connect two devices operating under different current levels to a power supply voltage, a motor driver module is necessary. The motor acts as a third device that adjusts the voltage supply up or down. Integrated circuits, or motor shields, are the most commonly available motor drivers in the market.

A motor driver board typically includes one or more motor driver chips, such as H-bridge chips, along with power regulators, voltage dividers, and other circuitry. These components ensure that the motors receive the correct voltage and current to operate efficiently.

When the microcontroller sends signals to the motor driver chip, the motor driver board translates these signals into the necessary voltage and current levels to drive the motors. Motor driver boards can work with various motor types, including DC, stepper, brushless DC, servo, and AC motor drivers, and they may also feature additional functionalities such as speed control, direction control, and braking

The principle of H bridge is:
To control the motion of a motor, the microcontroller sends signals to the motor, which are then interpreted and stepped up by the motor driver. The motor driver typically has two voltage input pins, with one pin turning on the driver and the other applying voltage to the motor via the motor IC. If the microcontroller sends a high input to the driver IC, the driver IC will send the same input without changing its type.

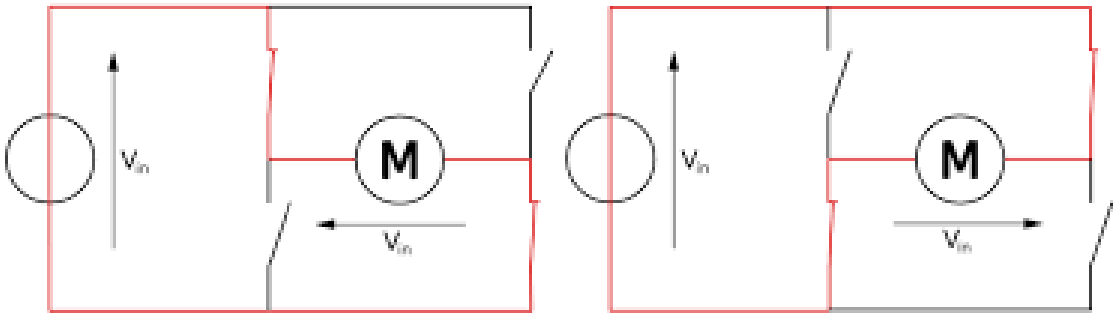

Figure 6. H bridge direction[29]

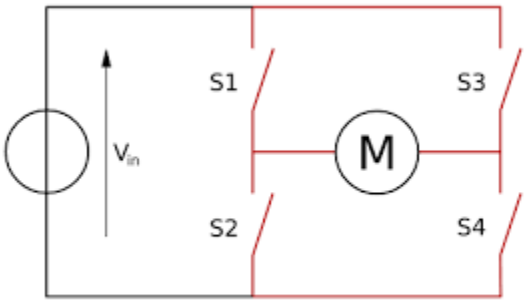

Figure 7.H bridge S1-S4[29]

To rotate the motor clockwise, switches S1 and S4 close while switches S2 and S3 open, allowing voltage to flow directly from S1 to S4 and completing the circuit. This causes current to flow from point V to point M via switches S1 and S4, resulting in the motor staying on and turning clockwise. Conversely, to rotate the motor anticlockwise, switches S2 and S3 are activated, which closes switches S2 and S3 to form a positive connection and cause the motor to rotate in an anticlockwise direction.[29]

2.1.3 Other hardwares

1.Electric engines

Robotic devices that incorporate electric motors generally exhibit better precision and repeatability. This quality makes them more suitable for accurate jobs, such as assembly, and also makes them more compact and portable. These systems generate electricity by means of an electric current and require minimal upkeep while running silently.

Electric motors are devices that use electrical energy to drive mechanical motion. They are classified into two types: Direct Current (DC) and Alternating Current (AC), each with unique characteristics that make them suitable for specific applications. Since the control of AC motors requires alternating current and is more challenging, DC motors are more commonly used in robotic applications. However, AC motors are used in scenarios that require strong initial torque. The selection of an appropriate motor must take into consideration factors such as torque, voltage, speed, accuracy, power factor, and motor size[43-44].

Main types of motors are included:

Type	
DC motor	Simple speed control, low cost, controls directions of motors
AC motor	High power, high starting torque, not suited for prototype robots
Servo motor	High precision, high torque, easy to controls rotation, low cost
Stepper motor	Precise Control,High Torque at low speeds, Complex control

Table 2. Main types of motors [42]

2.Kit

A robot kit is a collection of parts and components that may be put together to create a functional robot. Robot kits are created to give people the resources and information they need to create their own robots and learn about robotics.The following categories can be used to classify the many robot kits that are readily accessible on the market:

Educational Robot Kits: These kits are designed for beginners and students to learn about robotics and programming. They usually come with pre-programmed software, and the hardware is easy to assemble and use. Examples include Lego Mindstorms, Makeblock mBot, and VEX Robotics.

Hobbyist Robot Kits: These kits are designed for enthusiasts and hobbyists who want to build advanced robots with more features and functions. They often require some level of programming and electronics knowledge. Examples include Robotis Bioloid, RoboRealm, and Trossen Robotics.

Professional Robot Kits: These kits are designed for researchers and professionals who need advanced robotics capabilities for research or industrial applications. They are usually more expensive and require a higher level of expertise to assemble and program. Examples include Boston Dynamics' Spot robot, ABB's IRB 1200, and KUKA's LBR iiwa.

DIY Robot Kits: These kits are designed for those who want to build their own robots from scratch using their own designs and components. They often require a high level of technical knowledge and expertise, but they offer more flexibility and customization options. Examples include Raspberry Pi Robotics Kit, Arduino Robotics Kit, and RobotShop Rover[45].

3. Battery

The battery voltage directly influences the amount of electrical power available to drive the robot's motors and electronics, which can have a big impact on how well the robot performs. A robot's battery voltage may be impacted in the following ways:

Motor Performance: The speed and torque of the robot's motors are influenced by the battery's voltage. Generally speaking, higher voltage batteries will provide the motors more power, leading to faster speeds and more torque. The motors may not have enough power to move the robot adequately if the battery voltage is too low.

Electronics Performance:The battery voltage can impact how well the sensors and microcontrollers in the robot work. These components have an ideal voltage range they operate best at, so if the battery voltage goes beyond this range, their performance may be negatively affected. For instance, if the battery voltage is too low, the sensors may not be able to precisely detect the robot's surroundings.

Battery Life: The voltage of a battery affects its capacity and overall lifespan. Higher voltage batteries generally have a higher capacity, which means they can provide more power for a longer period of time. However, using a battery at a voltage outside of its recommended range can reduce its lifespan and cause it to degrade faster.

Charging: The way a battery is charged is also influenced by its voltage. Each type of battery needs a specific charging voltage and current, and if it's charged at a voltage that's either too high or too low, it can lead to the battery being damaged or failing earlier than it should.[46]

2.2 Softwares

When planning the path for a mobile robot, there are several general steps that need to be followed. Firstly, an environmental model must be created based on the known map information, which converts the actual environment for the robot into convenient map feature information.

Secondly, optimization criteria must be established. Finally, a path search algorithm is used to find a path that meets the optimization criteria while being collision-free between the starting and target points in the state space. Examples of optimization criteria may include path length, smoothness, and safety degree. Figure * illustrates the basic principle of path planning for mobile robots.

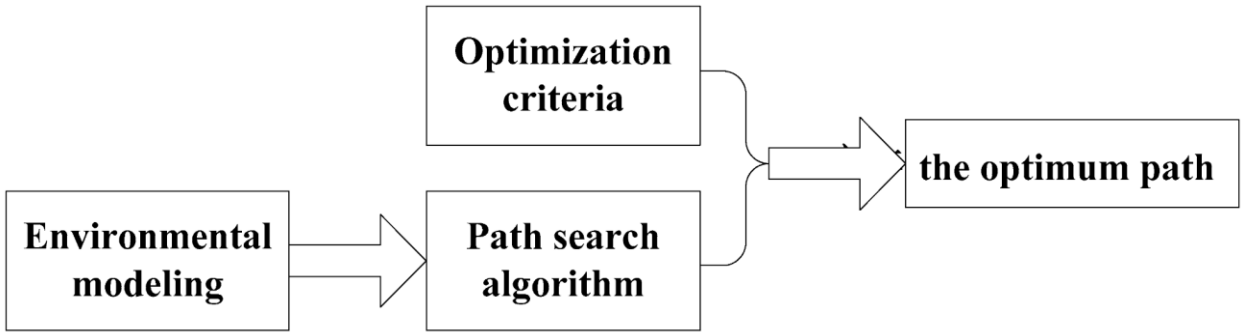

Figure 8. Path planning steps[52]

When it comes to the path search of our robot, it is important to use high-level algorithms. This is because the robot needs to consider several factors such as the efficient path and the obstacles that may encounter along the way. High-level algorithms are better equipped to handle these considerations and they offer more efficient solutions to path search problems. By choosing a high-level algorithm, we can ensure that our robot is not only efficient in reaching its destination, but it can also navigate around obstacles without getting stuck or causing any damage. Ultimately, this ensures that our robot can perform its tasks effectively and safely.

When it comes to designing a street cleaning robot that can effectively navigate and clean outdoor environments, the choice of algorithm is critical. While there are numerous algorithms available, it's important to select the right ones that can meet the specific needs of the robot in question.
In the case of street cleaning robots, several algorithms are particularly well suited for the task. These include Dijkstra, A*, D*, and SLAM algorithms. One of the main reasons these algorithms are suitable is their ability to handle the unique challenges that come with outdoor environments, including varying terrain, unexpected obstacles, and other potential hazards.

Dijkstra algorithm

The Dijkstra algorithm is a graph traversal algorithm used to find the shortest path between two points. It starts by initializing a starting point with a weight of 0 and sets all other points to infinity.

The algorithm then iteratively considers neighboring points and calculates their weights based on the sum of the weight of the current point and the weight of the edge connecting it to the neighboring point. It chooses the neighboring point with the lowest weight and adds it to the set of visited points.

This process is repeated until all points have been visited or until the destination point has been reached. The resulting path has the lowest weight and is considered the shortest path.
In Dijkstra's algorithm, we begin at a node called the initial node. We assign an initial distance of infinity to all other nodes. The algorithm will then attempt to update and improve the distance values for each node in a series of steps.[32]

1.Start with all nodes unvisited, create a set of unvisited nodes.

2.Assign a tentative distance value to each node. The initial node has a value of zero and all other nodes have a value of infinity. The tentative distance of a node is the shortest path found so far from the starting node to that node. Set the initial node as the current node.

3.For the current node, calculate the tentative distance to each of its unvisited neighbors. If this new distance is smaller than the previously assigned tentative distance, update it. Keep track of the smallest tentative distance found so far.

4.After considering all unvisited neighbors of the current node, mark it as visited and remove it from the set of unvisited nodes. Visited nodes will not be revisited.

5.If the destination node has been visited, or if the smallest tentative distance among the unvisited nodes is infinity, then the algorithm has finished. Otherwise, select the unvisited node with the smallest tentative distance and set it as the new current node.

6.Repeat steps 3 to 5 until the algorithm has finished. The order of visiting nodes is always the one with the smallest tentative distance from the starting node.[31]

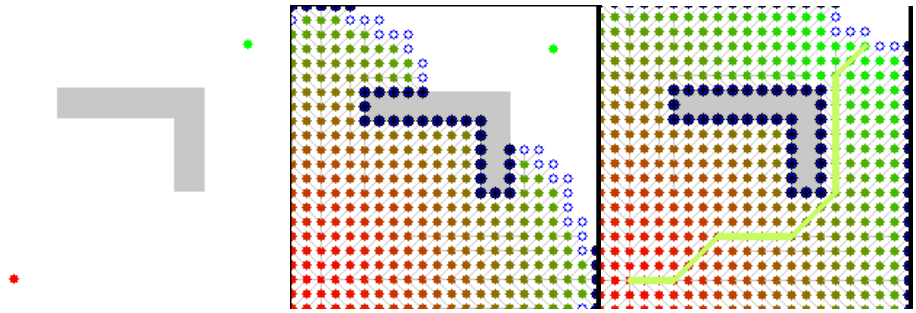

Figure 9.Dijkstra algorithm[53]

The top 3 pictures show the process of Dijkstra's algorithm visiting all points until it reaches the destination point.

A* algorithm

The A* algorithm is an extension of the Dijkstra algorithm and works by updating the weighted values of child nodes from a starting node. The child node with the smallest weighted value is chosen to update the current node, and this process is repeated until all nodes have been traversed. The algorithm's key feature is its evaluation function, f(n) = g(n) + h(n), which combines the actual cost of reaching node n from the initial node (g(n)) with an estimated cost of the optimal path from node n to the target node (h(n)). Typically, h(n) is calculated as the Euclidean distance between the two nodes. When g(n) is constant, the value of f(n) is mainly determined by h(n). As the node gets closer to the target node, h(n) becomes smaller and, therefore, f(n) also becomes smaller, ensuring that the shortest path is always pursued towards the target node. The A* algorithm takes into account the position information of the target node and searches along this direction, making it more efficient than the Dijkstra algorithm in terms of path searching.[32]

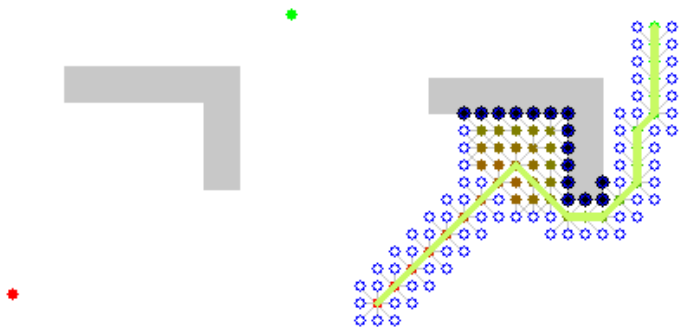

Figure 10. A* algorithm [54]

The top 2 pictures show the process of A* algorithm pursued towards the target node.

D* algorithm

The Dynamic A* (D*) algorithm is a method for quickly finding short routes in graphs where the cost of moving between points can change over time. It is frequently employed in robotics to plan paths through dynamic environments, where the robot's surroundings are in constant flux.

D* has a few key advantages. Most of the time, changes to the cost of moving between points only affect areas close to the robot's current location, so only the part of the path beyond the robot needs to be re-planned. This results in a map of optimal costs that only covers areas the robot may visit. Furthermore, D* uses states that are likely to result in new optimal paths to update the cost map, making it efficient and partially re-entrant.

The algorithm determines whether to update the path by searching for a new optimal path or checking if the previous path is still optimal. D* is active in memory, can function in unbounded environments, and is computationally efficient. By sorting arc costs using transfer cost values that can vary continuously, the algorithm repeatedly discovers the best path through the graph. Arc costs can be adjusted at any time, and the map of the environment includes both estimated and measured costs.[33]
D* algorithm equation:
$f(n) = h(n)$

2.3 Design solutions

Solution 1. GPS based localization, odometry and ultrasonic/lidar/beam obstacle detection

The effectiveness of using GPS-based localization, odometry and obstacle detection sensors such as ultrasonic, lidar, and beam can be significant for outdoor navigation prototypes. The GPS sensor is essential in this setup as it enables the device to pinpoint its location through signals received from multiple GPS satellites, which can also provide additional data such as speed, altitude, and direction. Obstacle detection sensors such as ultrasonic, lidar, and beam, which work independently, can be used to complement GPS-based localization, offering unique features to detect and avoid obstacles in the device's path. Odometer can provide important information about a robot's orientation and heading, which is useful for tasks like navigation and localization.

Ultrasonic sensors work by emitting sound waves, while lidar sensors use lasers to create a 3D map of the environment, and beam sensors use infrared light to detect obstacles in their path.

However, challenges arise in the form of creating accurate 2D maps of specific environments and finding high-quality sensors that can accurately detect obstacles and provide reliable data. High-level algorithms such as Dijkstra's algorithm and A* algorithm can be used alongside GPS-based localization and obstacle detection sensors to optimize the navigation system. For example, A* algorithm can be used to find the shortest path between two points, while considering the cost of reaching the destination.

Solution 2. Camera based localization, antenna and ultrasonic/lidar/beam obstacle detections

This solution is ideal for small areas, where a camera can be utilized to provide visual information about the environment and create 2D maps showing obstacles and other crucial navigation details. The camera's placement should be carefully considered to ensure a clear view of the area being navigated, which allows for detailed image capture to create accurate maps and obstacle identification. An antenna can be used to receive signals from Wi-Fi access points or other sources for location determination.

However, implementing this solution requires careful consideration of several factors such as selecting appropriate sensors, installing the camera in the correct location, and calibrating the system for optimal performance. Additionally, environmental factors such as lighting and the presence of reflective surfaces can impact the system's accuracy and reliability.

Chapter 3. Implementation

3.1 Assembly of prototype

We have come up with a design solution for our robot that will operate in outdoor environments. After exploring different options, we have chosen solution one, which relies on GPS localization and obstacle avoidance using ultrasonic sensors.

The GPS localization system will accurately pinpoint the robot's location on the map and facilitate easy navigation along the designated route. It will also provide real-time monitoring of the robot's movements and progress. When performing tasks like navigation and localization, a robot's odometer can provide crucial details about its orientation and heading.

Obstacle avoidance is a critical feature for our robot's safe and efficient navigation outdoors. For this reason, we have integrated ultrasonic sensors that can detect any obstacles in the robot's path and alert it to take necessary evasive action. This feature will help our robot avoid any potential obstacles and move forward without interruption.

Furthermore, we will use the A* algorithm to enable our robot to navigate smoothly around obstacles. This algorithm calculates the shortest possible path for the robot to take while avoiding obstacles and can make real-time adjustments as needed. This will enable our robot to move quickly and safely while maintaining optimal efficiency.

3.1.1 Robot base selection

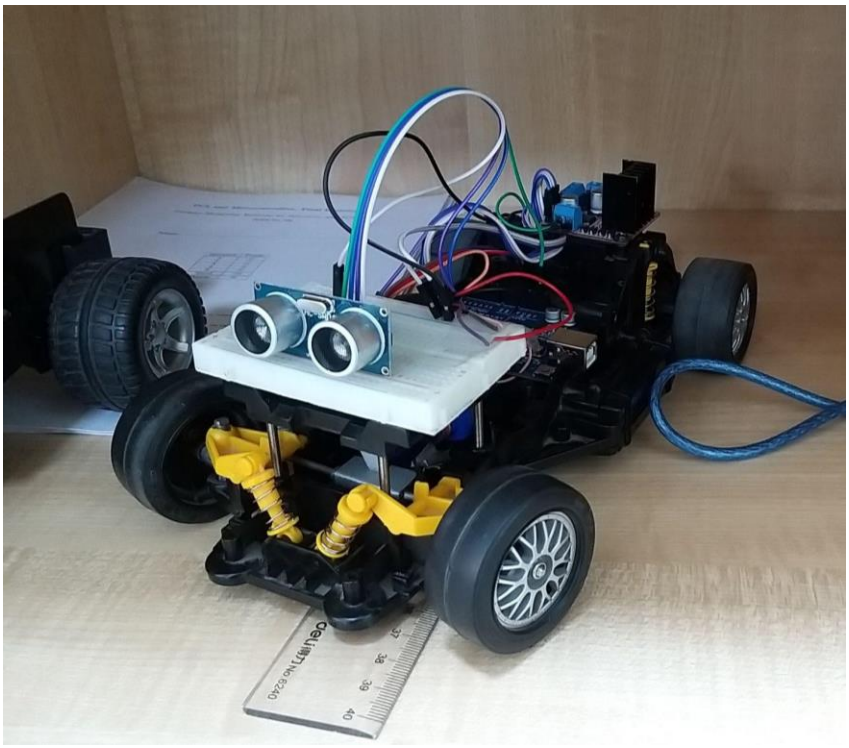

Figure 11. Robot base

The base serves as the framework for the entire robot and needs to be built to withstand the weight of both the base itself and additional components. Therefore, it is essential to choose a base with powerful motors capable of supporting the weight of the robot. A microcontroller, driver, circuit breadboard, and sensors will be added to the robot, thus the size of the base must be able to handle all of these components.

We were thrilled to discover a robot kit in the mechanical laboratory that we could potentially utilize for our project. After conducting a thorough examination of the kit, we found that it consisted of four wheels that are essential for its movement. Out of these four wheels, the back two wheels are designed to work with a single DC motor, while the front two wheels, which are the leading wheels, work with servo motors. The DC motor is responsible for driving the back wheels and powering the robot's movement, while the servo motors provide precise control over the direction of the front wheels.

The use of servo motors in the front wheels enables the robot to have excellent maneuverability, allowing it to move in different directions and turn sharply without losing balance. This feature is crucial for our project, as we need the robot to navigate through challenging terrains and obstacles

in the outdoor environment. The robot's four-wheel design also provides a stable base for it to move on and ensures that it can carry the necessary payload for its intended purpose. This design is particularly useful for our project, where we need to transport our hardwares through outdoor terrain

3.1.1 GPS selection

Ublox NEO-M6: This is a popular GPS module that provides accurate positioning information with a low power consumption. It supports GPS, GLONASS, Galileo, and QZSS satellite systems.

MediaTek MT3339: This GPS module is designed for portable devices and provides high sensitivity and fast time-to-first-fix performance. It supports GPS, GLONASS, and QZSS satellite systems.

Skytraq Venus838LPx-T: This GPS module is a low-power, high-sensitivity receiver that supports GPS, GLONASS, and QZSS satellite systems. It is suitable for battery-powered devices such as handheld GPS units.

Quectel L80: This GPS module is a compact, high-performance receiver that supports GPS and GLONASS satellite systems. It has a small form factor and low power consumption, making it suitable for a wide range of applications.

Adafruit Ultimate GPS: This is a popular GPS module for Arduino and other microcontroller projects. It supports GPS and GLONASS satellite systems and provides real-time position and velocity information. It also includes an external antenna for improved reception.

NEO-M8P-2 GPS: This module achieves 25mm accuracy and has geofencing, variable I2C address, and multiple communication interfaces, including USB, UART, I2C, and SPI. It's easy to use with an Arduino-compatible library and has a Time to First Fix of 29 seconds (cold) and 1 second (hot).
[5-8],[40]

Because of financial limitations, the only one has been found from Mongolia was the NEO-6M GPS module.The central component of the module is the U-blox NEO-6M GPS chip, which is

capable of tracking up to 22 satellites on 50 channels with remarkable sensitivity of -161dB while consuming only 45mA of current. The chip's U-blox 6 positioning engine can achieve a Time-To-First-Fix (TTFF) of less than 1 second. One of the NEO-6M GPS chip's standout features is its Power Save Mode (PSM), which enables a reduction in the system's power consumption by selectively turning certain parts of the receiver ON and OFF. This feature significantly reduces the module's power consumption to only 11mA, making it suitable for power-sensitive applications like GPS wristwatches.

The NEO-6M GPS chip's necessary data pins are available through 0.1" pitch headers, providing the pins required for communication with a microcontroller over UART. The module supports baud rates ranging from 4800bps to 230400 bps, with a default baud rate of 9600.[35]

Technical Specifications:

Type of receiver: GPS/GLONASS/QZSS/BeiDou receiver with 50 channels
Accuracy of position: Within 2.5 meters (CEP50)
Accuracy of velocity: Within 0.1 meters per second
Frequency of updates: Up to 5 Hz
Time-To-First-Fix (TTFF): Less than 1 second for a hot start, less than 27 seconds for a warm start,and less than 30 seconds for a cold start

Figure 12. NEO-6M module[35]

Sensitivity: Capable of tracking signals at -161 dBm and acquiring signals at -148 dBm
Power supply voltage: Ranges from 3.3V to 5V
Power consumption: 45 mA in normal mode, 11 mA in power save mode
Operating temperature range: -40°C to 85°C
Dimensions: Measures 25mm x 35mm x 4mm[34]

3.1.2 Ultrasonic selection

The HC-SR04 ultrasonic ranging module is capable of measuring distances ranging from 2cm to 400cm without making physical contact, with an accuracy of up to 3mm. It is composed of ultrasonic transmitters, a receiver, and a control circuit. The module operates based on the following principles:

(1) A high level signal of at least 10us is triggered using IO.

(2) The module then automatically sends eight 40 kHz signals and checks if there is a pulse signal received back.

(3) If a signal is detected, the high-level output IO duration represents the time it took for the ultrasonic signal to be sent and received.

The measured distance can then be calculated using the formula: Test distance = (high level time x velocity of sound (340M/S)) / 2.[37]

Technical Specifications of HC-SR04:

Voltage: 5 volts DC
Current: 15 milliamperes
Frequency: 40 kilohertz
Farthest detectable range: 4 meters
Closest detectable range: 2 centimeters
Precision: 3 millimeters
Trigger input signal: a TTL pulse of 10 microseconds
Echo output signal: a TTL-level signal that varies based on the distance to the object
Module dimensions: 45 millimeters x 20 millimeters x 15 millimeters[36]b

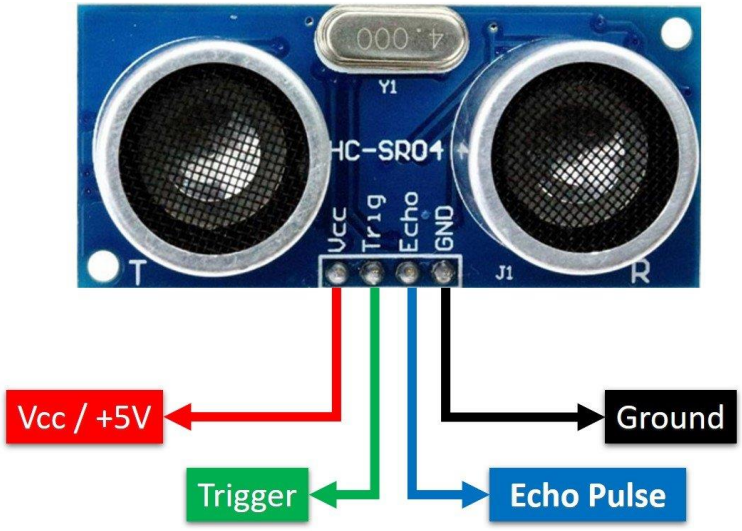

Figure 13. HC-SR04[36]

3.1.3 Microcontroller board selection

Some common examples of microcontroller devices:

Atmel AVR: a family of microcontrollers that use RISC architecture and are commonly used in embedded systems and hobby projects.
Microchip PIC a family of microcontrollers that have been popular in industrial and consumer electronics applications for many years.
Texas Instruments MSP430: a low-power microcontroller family used in applications where power efficiency is critical, such as in portable medical devices and energy management systems.
NXP LPC: a family of microcontrollers based on the ARM Cortex-M architecture that are often used in industrial automation, medical devices, and other embedded applications.
STMicroelectronics STM32: a family of ARM-based microcontrollers that are widely used in industrial control systems, robotics, and other embedded applications.
Arduino Uno: a popular microcontroller board based on the Atmel AVR microcontroller that is often used in hobbyist projects.
Raspberry Pi Pico: a microcontroller board developed by the Raspberry Pi Foundation that is based on the RP2040 microcontroller and designed for embedded systems and Internet of Things (IoT) applications.
ESP32: a microcontroller board based on the ESP32 microcontroller that includes Wi-Fi and Bluetooth connectivity, making it popular for IoT applications.
Renesas RX: a family of microcontrollers that are often used in automotive, industrial, and medical applications due to their high performance and reliability.
Silicon Labs EFM32: a family of low-power microcontrollers that are often used in battery-powered applications, such as sensors and wearables.[16-25]

The Arduino board is built around the ATmega328P microcontroller and is relatively easy to use when compared to other boards like the Arduino Mega. It comes equipped with a range of digital and analog Input/Output pins (I/O), circuits, and shields.

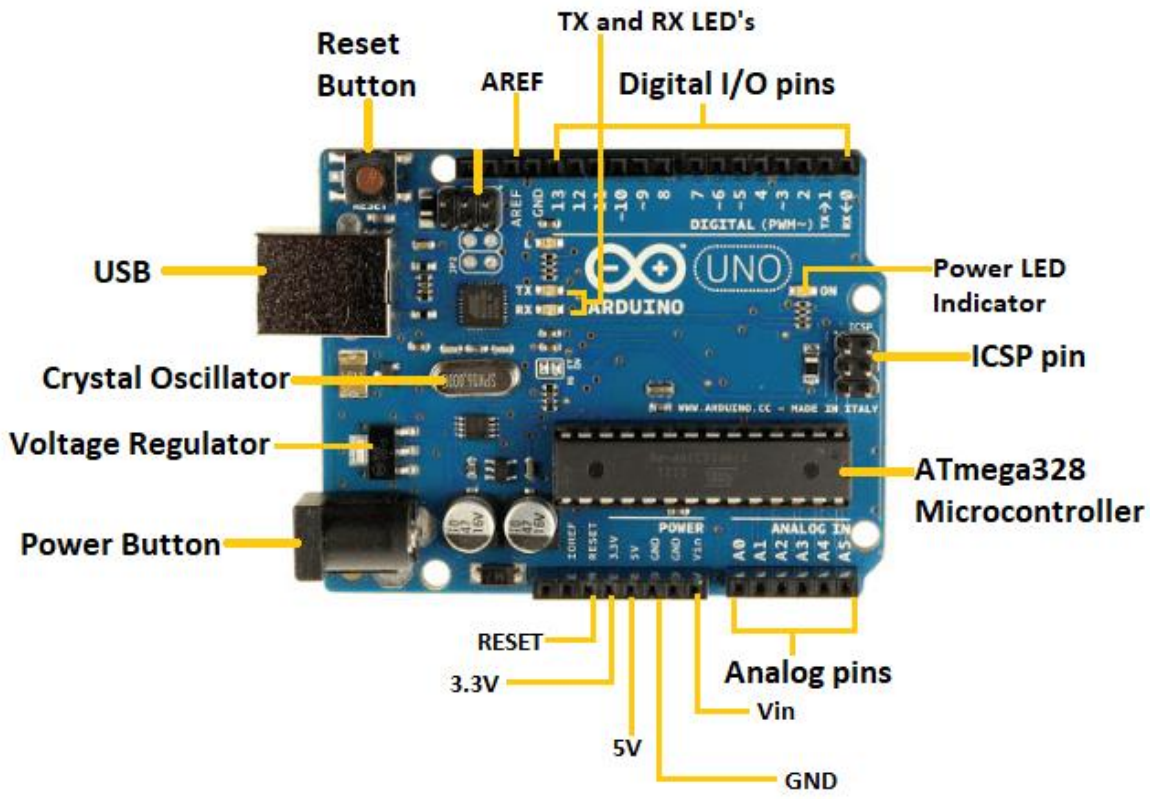

Figure 14. Arduino UNO parts[39]

In details:

The ICSP pin allows programming using the firmware of the Arduino board, and the Power LED Indicator shows if the power is on or off.

The Digital I/O pins, which are numbered from DO to D13, have the value HIGH or LOW. The successful data flow is represented by the lighting of the TX and RX LEDs.

The AREF pin feeds a reference voltage to the Arduino UNO board from the external power supply, while the Reset button is used to add a Reset button to the connection.

The USB is essential for the programming of the Arduino UNO board, and the Crystal Oscillator has a frequency of 16MHz, making the Arduino UNO board powerful.

The Voltage Regulator converts the input voltage to 5V, and the GND pins act as a pin with zero voltage.

Vin is the input voltage, and the Analog Pins numbered from A0 to A5 are used to read the analog sensor used in the connection and can also act as GPIO (General Purpose Input Output) pins.[39]

The ATmega328P microcontroller is used in the system.
The operating voltage of the system is 5 volts
The recommended input voltage is 7-12 volts, while the limit is 6-20 volts
The system has 14 digital input/output pins, six of which can be used for pulse-width modulation (PWM).
There are six analog input pins.
The direct current (DC) per input/output pin is 20 milliamperes (mA), while the DC current for the 3.3V pin is 50 mA.
The system has 32 kilobytes (KB) of flash memory, but 0.5 KB is already used by the bootloader.
There are 2 KB of static random-access memory (SRAM) and 1 KB of electrically erasable programmable read-only memory (EEPROM).
The clock speed of the system is 16 MHz.

Table 3. Technical specifications of Arduino uno[38]

3.1.4 Motor driver selection

Here's some different common types of motor driver boards.

Motor driver boards	Short informations
1. L293D IC	Designed to control 2 DC motors simultaneously.
2. BTS7960B motor driver board	Controls the speed and the direction of the DC motor based on the PWM signal

3. TB6612FNG motor driver board	Can control 2 DC motors at 1.2 A constantly. 2 input signals (IN1 and IN2) can be used to control the motor in one of four function modes CW, CCW, short-brake, and stop.
4. PCA9685 16 channel Servo motor driver board	The PCA9685 16-Channel 12-bit PWM/Servo Driver will drive up to 16 servos over 12C with only 2 pins. Can be interfaced with the microcontroller like Arduino and Raspberry Pi etc. Can control 16-servo motors with just 2 12C pins.
5. MACH3 Interface Board CNC 5 Axis	Maximum support 5-axis stepper motor driver controllers.Compatible with MACH3, Linux CNC (EMC2),etc. parallel-control CNC software. USB power supply and peripherals power phase are separate to protect computer security.
6. L298 Motor driver board	Can control 2 DC motors with speed and direction. Can be interfaced with the microcontroller like Arduino, Raspberry Pi, ESP32, etc. A 12V input voltage can be given to driving the motors. And has a 5V onboard voltage regulator.
7. PWM DC motor speed controller	DC motor speed control using PWM signal. Onboard Potentiometer to vary the duty cycle of the PWM signal. Can handle loads of 9V-50V and up to 500W.

Table 4. Types of motor driver board[30]

The reason we chose The L298N Motor Driver Module is because we already had one available. This module is specifically designed to control high-power DC and stepper motors, utilizing both an L298 motor driver IC and a 78M05 5V regulator. With the capability of controlling two DC motors with both speed and directional control, it is a versatile solution for motor control needs.

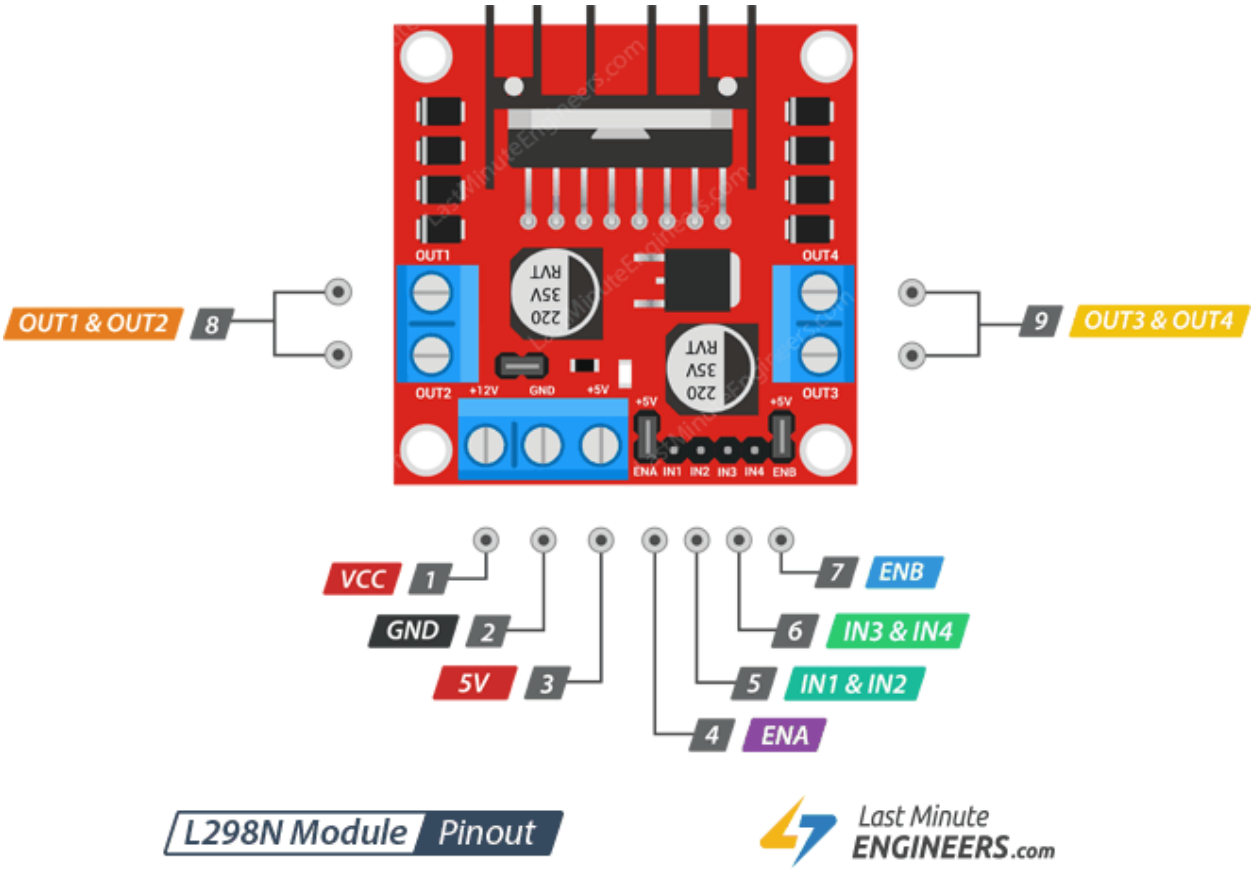

Figure 15. L298N input and outputs[55]

Pinout Configuration:

IN1 and IN2 are input pins for Motor A. They are utilized to manage the spinning direction of Motor A.

IN3 and IN4 are input pins for Motor B. They are utilized to control the spinning direction of Motor B.

ENA is the pin that enables the PWM signal for Motor A.

ENB is the pin that enables the PWM signal for Motor B.

OUT1 and OUT2 are output pins for Motor A.

OUT3 and OUT4 are output pins for Motor B.

12V is the input pin for a DC power source that provides 12 volts of power.

5V is the power source that supplies power to the switching logic circuitry inside the L298N IC.

GND is the ground pin.[55]

The type of driver used is L298N 2A.
The driver incorporates a double H bridge L298N chip.
The maximum voltage that can be provided to the motor is 46V
The maximum current that can be supplied to the motor is 2A.
The voltage used for logic is set at 5V.
The driver can operate at a voltage range of 5V to 35V.
The maximum current that the driver can handle is 2A.
The logical current can range from 0mA to 36mA.
Each motor is equipped with a current sensing feature.
The maximum power that can be delivered by the driver is 25W.
The driver has a power-on LED indicator.
A heatsink is included to improve the driver's performance.

Table 5. Technical specifications[55]

3.2 Simulation

The prototype's simulation was a critical component of our project, as it allowed us to evaluate the obstacle avoidance system's performance in a controlled environment before deploying it on the physical robot. We chose Unity as our IDE because of its cross-platform capabilities, which enabled us to develop a virtual environment that closely resembled the physical environment that the robot would be operating in.

To create the simulation, we first modeled GMIT's environment in Unity, using a combination of 3D modeling tools and texture mapping techniques. Which is shown in Figure 16.

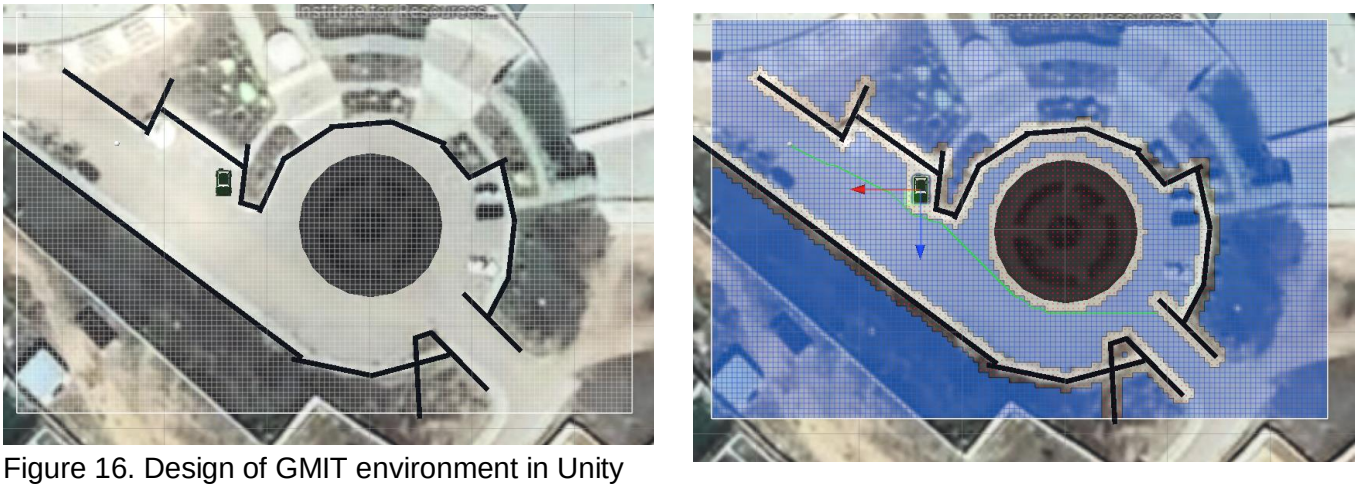

Figure 16. Design of GMIT environment in Unity

We then programmed the A* algorithm into the simulation, which allowed the robot to navigate the virtual environment while avoiding obstacles. The A* algorithm was designed to consider the position and size of the obstacles, as well as the distance between the robot and the obstacles, in order to calculate the safest and most efficient path through the environment.

During our testing of the prototype, we were impressed with how well the obstacle avoidance system handled dynamic obstacles. Even when we placed obstacles in the path that the robot had planned to take, the system was able to quickly update its map and generate a new, more efficient path based on the weights of the nodes in the environment.

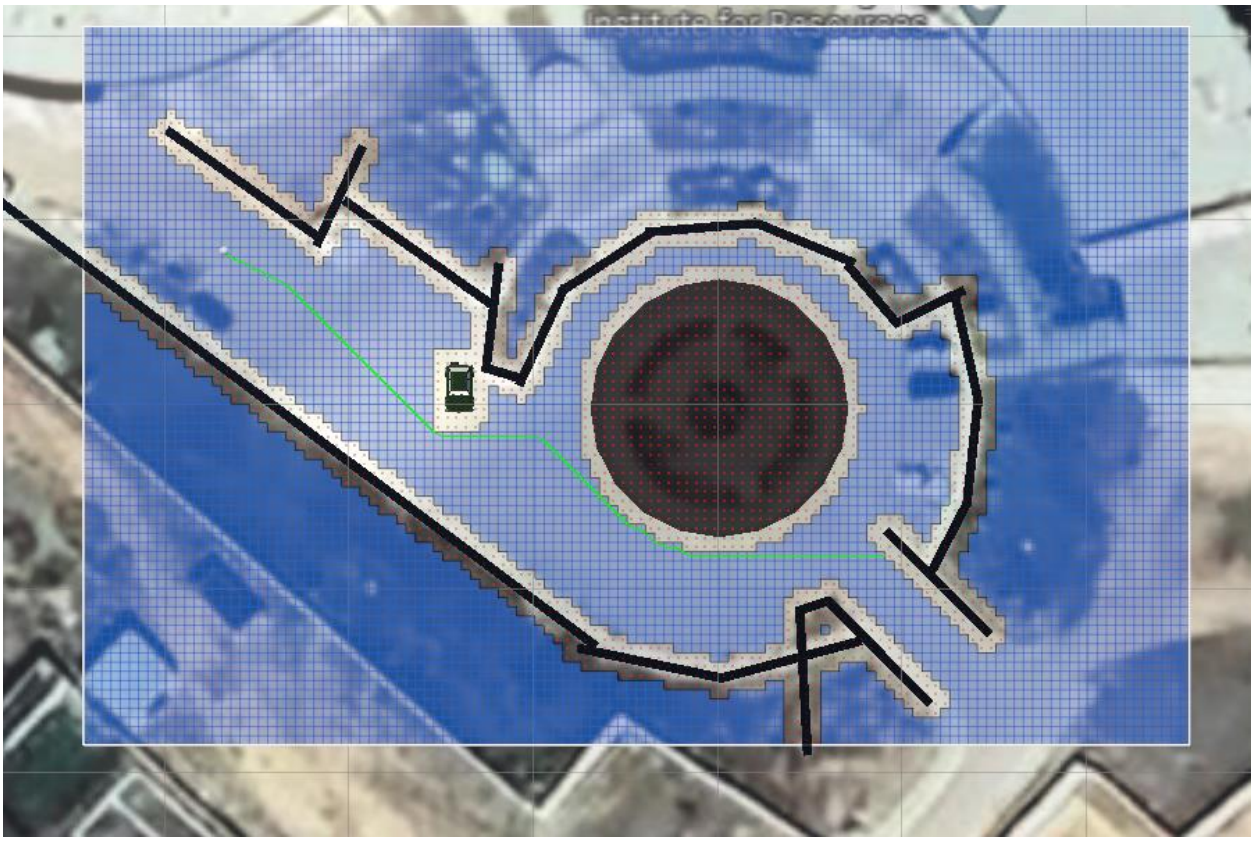

Figure 17. Dynamic obstacle experiment

As illustrated in the figure 17, we placed a car in the path of the robot to test its ability to handle dynamic obstacles. The car served as a moving obstacle, and we observed how the system responded to its presence. Despite the added complexity of a dynamic obstacle, the system was able to update its map and generate a new, more efficient path for the robot to follow. This demonstrated the system's ability to adapt to changes in the environment and handle unexpected obstacles, making it a valuable tool for navigating real-world scenarios.

In conclusion, the simulation we conducted using Unity and our A* path planning algorithm has demonstrated that our system is capable of handling complex environments and dynamic obstacles. Our system was able to generate efficient paths for the robot to follow, even in the presence of obstacles that were added to the environment during testing.

3.3 Coding

3.3.1 Low level of software codings

Motor control code

Our prototype model has a DC motor in the back two wheels and a servo motor in the front that is responsible for directing the prototype. However, these motors and microcontrollers have varying voltage ranges and current levels. In order to connect these two devices to a power supply voltage, we require a motor driver module. Thus, we connected our DC and servo motors to the motor driver's outputs one to four, and the Arduino Uno's digital pins were connected to the motor's enable and input pins. To control the motor, we implemented the following code.

Figure 18. Motors code

```
#include <Servo.h>
Servo myservo;
int trigPin = 10;
int echoPin = 11;
int fwdback = 4;
int revback = 5;
int servo = 9;
int servoPOS = 0;


String serial_in;


void setup() {
  Serial.begin(4800);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  myservo.attach(9);
  analogWrite(3, 100);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  myservo.write(70);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
}
```

Figure 19 Motors code

```
void loop() {

  while (Serial.available()) {
    serial_in = Serial.readString();
    if (serial_in.equals("forward")) {
      forward();
    } else if (serial_in.equals("backward")) {
      backward();
    } else if (serial_in.equals("right")) {
      right();
    } else if (serial_in.equals("left")) {
      left();
    } else if (serial_in.equals("stop")) {
      stop();
    } else if (serial_in.equals("straight")) {
      straight();
    }
  }
}
void forward() {
  digitalWrite(4, HIGH);
  digitalWrite(5, LOW);
}
void backward() {
  stop();
  delay(100);
  digitalWrite(4, LOW);
  digitalWrite(5, HIGH);
}
void right() {
  straight();
  delay(100);
  myservo.write(95);
}
void left() {
  myservo.write(45);
}
void stop() {
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
}
void straight() {
  myservo.write(70);
}
```

GPS sensor data code

We utilized a piece of code that makes use of a GPS library to obtain these coordinates. The latitude and longitude, as well as other important locational data, are extracted from the GPS device's transmitted data by this library. In order to manage other devices or carry out other relevant tasks, the retrieved data is subsequently saved in variables.

GPS Sensor

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>


static const int RXPin = 2, TXPin = 3;
static const uint32_t GPSBaud = 9600;


// The TinyGPS++ object
TinyGPSPlus gps;


// The serial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);


void setup() {
  Serial.begin(9600);
  ss.begin(GPSBaud);
}


void loop() {
  // This sketch displays information every time a new sentence is correctly
encoded.
  while (ss.available() > 0) {
    gps.encode(ss.read());
    if (gps.location.isUpdated()) {
      Serial.print("Latitude= ");
      Serial.print(gps.location.lat(), 6);
      Serial.print(" Longitude= ");
      Serial.println(gps.location.lng(), 6);
    }
  }
}
```

Figure 20. GPS code

Ultrasonic sensor data code

The ultrasonic sensor is a device that can be used to determine the range of obstacles. To calculate the range, several equations are utilized. These equations take into account the speed of sound and the time it takes for the sound waves to travel to the object and back to the sensor. By measuring the time difference between the transmission of the sound wave and its return, the distance between the sensor and the object can be determined. The following code can detect the distance

Ultrasonic sensor code

```
int trigPin = 12;    // TRIG pin
int echoPin = 13;    // ECHO pin

float duration_us, distance_cm;

void setup() {
  // begin serial port
  Serial.begin (9600);

  // configure the trigger pin to output mode
  pinMode(trigPin, OUTPUT);
  // configure the echo pin to input mode
  pinMode(echoPin, INPUT);
}

void loop() {
  // generate 10-microsecond pulse to TRIG pin
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // measure duration of pulse from ECHO pin
  duration_us = pulseIn(echoPin, HIGH);

  // calculate the distance
  distance_cm = 0.017 * duration_us;

  // print the value to Serial Monitor
  Serial.print("distance: ");
  Serial.print(distance_cm);
  Serial.println(" cm");

  delay(500);
}
```

Figure 21.Ultrasonic sensor code

3.3.2 High level of software coding

A* algorithm

In order to simulate the A* algorithm in the Unity game engine, we have implemented the algorithm using a combination of C# scripting and Unity's built-in functions. The algorithm involves creating a graph of nodes and edges, where each node represents a point in space and each edge represents a path between two nodes. By assigning weights to the edges based on their distance and any obstacles in the way, the algorithm can determine the most efficient path from the starting node to the goal node.

A* algorithm C#

```
using System.Collections.Generic;
using UnityEngine;


public class Astar : MonoBehaviour
{
   public Transform startNode;
   public Transform endNode;
   public LayerMask obstacleLayer;
   public float nodeSize = 1f;


   private Node[,] grid;
   private List<Node> openSet = new List<Node>();
   private HashSet<Node> closedSet = new HashSet<Node>();


   void Start()
   {
       // Initialize the grid
       int gridSizeX = Mathf.RoundToInt(Mathf.Abs(startNode.position.x -
endNode.position.x) / nodeSize);
       int gridSizeY = Mathf.RoundToInt(Mathf.Abs(startNode.position.z -
endNode.position.z) / nodeSize);
       grid = new Node[gridSizeX, gridSizeY];


       // Fill the grid with nodes
       for (int x = 0; x < gridSizeX; x++)
       {
           for (int y = 0; y < gridSizeY; y++)
           {
               Vector3 nodePosition = new Vector3(startNode.position.x + x *
nodeSize + nodeSize / 2f, 0f, startNode.position.z + y * nodeSize + nodeSize
/ 2f);
               bool isObstacle = Physics.CheckSphere(nodePosition, nodeSize /
2f, obstacleLayer);
               grid[x, y] = new Node(nodePosition, x, y, isObstacle);
           }
       }
```

Figure 21. A* algorithm

```
        // Run the A* algorithm
        List<Node> path = FindPath(startNode.position, endNode.position);
        if (path != null)
        {
            foreach (Node node in path)
            {
                Debug.Log(node.position);
            }
        }
    }


    private List<Node> FindPath(Vector3 startPosition, Vector3 targetPosition)
    {
        Node startNode = GetNodeFromWorldPosition(startPosition);
        Node targetNode = GetNodeFromWorldPosition(targetPosition);


        openSet.Clear();
        closedSet.Clear();
        openSet.Add(startNode);


        while (openSet.Count > 0)
        {
            Node currentNode = openSet[0];
            for (int i = 1; i < openSet.Count; i++)
            {
                if (openSet[i].fCost < currentNode.fCost || openSet[i].fCost
== currentNode.fCost && openSet[i].hCost < currentNode.hCost)
                {
                    currentNode = openSet[i];
                }
            }


            openSet.Remove(currentNode);
            closedSet.Add(currentNode);


            if (currentNode == targetNode)
            {
```

Figure 22. A* algorithm

Figure 23. A* algorithm

```
 }


                int checkX = node.gridX + x;
                int checkY = node.gridY + y;
                if (checkX >= 0 && checkX < grid.GetLength(0) && checkY >= 0
&& checkY < grid.GetLength(1))
                {
                    neighbors.Add(grid[checkX, checkY]);
                }
            }
        }


        return neighbors;
    }


    private List<Node> RetracePath(Node startNode, Node endNode)
    {
        List<Node> path = new List<Node>();
        Node currentNode = endNode;


        while (currentNode != startNode)
        {
            path.Add(currentNode);
            currentNode = currentNode.parent;
        }


        path.Reverse();
        return path;
    }


    private Node GetNodeFromWorldPosition(Vector3 worldPosition)
    {
        int x = Mathf.RoundToInt((worldPosition.x - startNode.position.x) /
nodeSize);
        int y = Mathf.RoundToInt((worldPosition.z - startNode.position.z) /
nodeSize);
        return grid[x, y];
    }


    private float GetDistance(Node nodeA, Node nodeB)
    {
```

```
            return RetracePath(startNode, targetNode);
        }

        foreach (Node neighbor in GetNeighbors(currentNode))
        {
            if (closedSet.Contains(neighbor) || neighbor.isObstacle)
            {
                continue;
            }


            float newMovementCostToNeighbor = currentNode.gCost +
GetDistance(currentNode, neighbor);
            if (newMovementCostToNeighbor < neighbor.gCost ||
!openSet.Contains(neighbor))
            {
                neighbor.gCost = newMovementCostToNeighbor;
                neighbor.hCost = GetDistance(neighbor, targetNode);
                neighbor.parent = currentNode;


                if (!openSet.Contains(neighbor))
                {
                    openSet.Add(neighbor);
                }
            }
        }
    }

    // No path found
    return null;
}


private List<Node> GetNeighbors(Node node)
{
    List<Node> neighbors = new List<Node>();
    for (int x = -1; x <= 1; x++)
    {
        for (int y = -1; y <= 1; y++)
        {
            if (x == 0 && y == 0)
            {
                continue;
```

Figure 24. A* algorithm

```
        int distX = Mathf.Abs(nodeA.gridX - nodeB.gridX);
        int distY = Mathf.Abs(nodeA.gridY - nodeB.gridY);
        if (distX > distY)
        {
            return 14f * distY + 10f * (distX - distY);
        }
        else
        {
            return 14f * distX + 10f * (distY - distX);
        }
    }


    public class Node
    {
        public Vector3 position;
        public int gridX;
        public int gridY;
        public bool isObstacle;
        public float gCost;
        public float hCost;
        public Node parent;


        public float fCost { get { return gCost + hCost; } }


        public Node(Vector3 position, int gridX, int gridY, bool isObstacle)
        {
            this.position = position;
            this.gridX = gridX;
            this.gridY = gridY;
            this.isObstacle = isObstacle;
        }
    }
}
```

Figure 25. A* algorithm

3.4 Testing

Testing is important because it makes sure that a system or product functions as expected and meets all of its requirements. It can spot errors or issues early on during the development cycle, which may save time and money. Furthermore, testing assists with improving the system's or product's overall quality.

The individual component testing involves checking each individual component, such as the GPS module and ultrasonic sensor, to make sure they are operating correctly and within the specified limits. Through testing, any flaws or limitations in each component can be found and fixed.

The subsystem testing involves testing the integration of multiple components, such as the location detection subsystem and obstacle detection subsystem, to ensure they are working together effectively and as intended. This testing helps to identify any issues or limitations in the interaction of the components and allows for improvements to be made to the overall system performance.

3.4.1 Individual components test

Ultrasonic sensor test

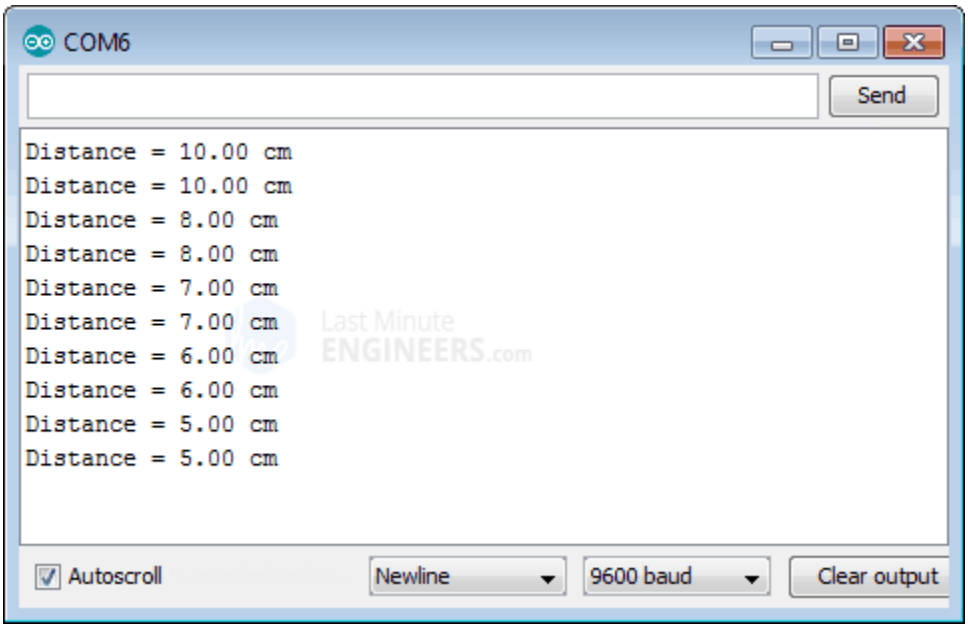

Figure 26. Test of ultrasonic

The experiment measured the distance between the obstacle and the sensor. Typically, ultrasonic sensors have a measurement range of 20mm-4000mm. However, during some testing, it was observed that the ultrasonic sensor was working in a range of around seven meters without any error, which was well beyond its advertised range. Moreover, the measurement was found to be quite stable, indicating the accuracy of the sensor even at such a long range.

GPS sensor test

Our testing of the NEO-6M GPS module highlighted some challenges that we had to overcome in order to obtain reliable location data. One of the main issues we encountered was that the module did not always provide stable latitude and longitude coordinates. We found that at times, it would give us readings that were significantly different from its actual location, with deviations of up to 20 meters being observed. To address this issue, we modified our code to ensure that the module did not provide any location data that differed from the previous reading by more than 5 meters. This helped to stabilize the readings, although it did result in some loss of data.
(In addition, the robot is most certainly inside the operating area when in operation. Therefore, if a robot is located outside the operation area due to GPS inaccuracy we are getting the closest point in the operable area)

Another problem we faced was that the NEO-6M GPS module had difficulty connecting with the satellites when it was inside buildings or in areas with limited visibility to the sky. This is a common limitation of GPS technology, which relies on direct line-of-sight communication with the satellites to determine location.

We found that the NEO-6M GPS module was not able to provide us with the exact location of our module due to its limited accuracy range of around 2.5 meters. While this level of accuracy may be sufficient for some applications, it may not be precise enough for others that require highly accurate location data.

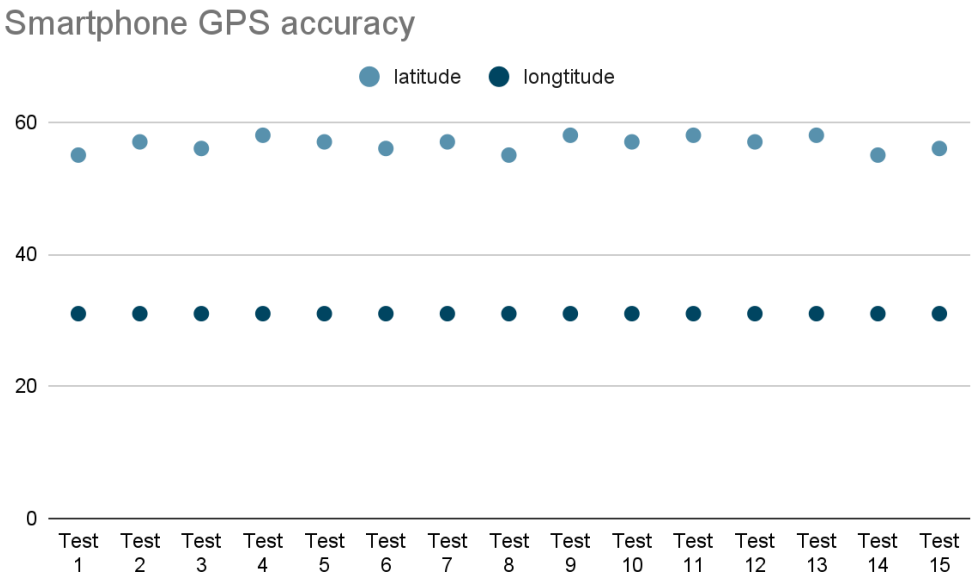

Figure 27. Smartphone GPS test

Figure 27 presents the results of our experiment, in which we conducted 15 tests to observe changes in longitude and latitude. The collected data shows that the latitude values were changing every 10 to 20 seconds with an accuracy ranging from 0.00001 to 0.00002, equivalent to a change in position of 1-2 meters[60]. In contrast, the longitude accuracy was remarkably stable and did not show significant variations throughout the tests.

As a result, we decided to use smartphone GPS for our prototype and Unity simulation.While smartphone GPS may not be accurate either, we found that it was stable and reliable even inside buildings, which made it a suitable alternative for our needs.

3.4.2 Subsystem test

Location detection subsystem

In order to test the functionality of a prototype, a subsystem test was conducted using both GPS and odometry. The prototype was able to determine its direction through the use of odometry, while GPS provided information on its latitude and longitude. Using this information, the prototype was able to create a map of its surroundings through the use of an Arduino microcontroller.

Despite initial successes, some issues were encountered during testing. One major issue was the inaccuracy of the GPS data, which caused the prototype to lose its navigation when it was located outside of the software's operation area. This meant that when the prototype was close to a curb or other obstacles, it would sometimes lose track of its position, leading to navigation errors.

Another issue encountered during testing was the prototype's tendency to collide with obstacles or curbs, causing it to stop. These collisions occurred again due to the inaccuracy of the GPS module.
Overall, the subsystem test of the prototype using GPS and odometry revealed both successes and challenges. While the prototype was able to use this data to create a map of its surroundings and navigate its environment, it also experienced issues with accuracy which resulted in the need to improve the GPS accuracy.

Obstacle detection subsystem

This subsystem is to detect obstacles in the robot's path using an ultrasonic sensor, and to place these obstacles dynamically on a high-level environmental map.

The process of dynamic obstacle placement involves detecting objects within a range of 1-2 meters using the ultrasonic sensor. When an obstacle is detected, the system places the weight on the map in real time. This allows the robot to build a comprehensive map of its environment, including both static and dynamic obstacles.

Once an obstacle has been placed on the map, the system continually monitors the ultrasonic sensor to determine whether the obstacle is still present. If the sensor no longer detects the obstacle, the system removes it from the map, indicating that the path is now clear.

If a new dynamic obstacle is found, the robot will stop and reevaluate its path. This ensures that the robot can safely avoid colliding with the obstacle and move around it. Every 0.2 seconds, the system checks for dynamic obstacles to provide the robot real-time environmental information.

Path planning subsystem

The system was able to update the path based on dynamic obstacles in real time, with updates occurring every 0.2 seconds.

Once the dynamic obstacles were placed on the map, the path planning system used this information to update the robot's path. The system ensured that there was a direct and clear path for the robot to move forward towards the next cell in the path. This ensured that the robot was able to navigate around obstacles in real-time, without getting stuck or colliding with any obstacles.

One key aspect of the path planning subsystem that was tested during the testing phase was the ability of the robot to rotate itself in the correct direction using information provided by the odometer sensor. This made sure the robot was always pointed in the right direction as it traveled to its objective and was always correctly orientated.

Full system testing

1. At this point, each individual subsystem’s have been tested and updated to be working
2. The whole system is being tested
    - Controlled environment testing
        - Given clear path, going to point A to point B
    - Controlled environment dynamic obstacles
        - With certain dynamic obstacles, going from point A to point B
        - Here we have discovered that if a dynamic object is placed by the sensor, sometimes it never gets removed as the sensor never gets pointed to the added obstacle afterwards. This results in a clustered environment map, and non optimal routes or unnavigable points on the map.
            - Implemented timeout duration. Once a dynamic object is placed, then after a certain amount of time (10 seconds, in our case is a good amount), remove the obstacle.

3.5 Result and discussion

The objective of this thesis was to develop a cleaning robot algorithm capable of working in any given environment, irrespective of the presence of static and dynamic obstacles. To achieve this, the algorithm was designed to be adaptable to different environments and to allow users to set the robot's starting static map and cleaning path points according to their requirements. By doing so, the algorithm enables the robot to navigate through various outside environments with ease, while detecting and avoiding both static and dynamic obstacles, resulting in a thorough and efficient cleaning performance.

The prototype of the street cleaning robot was designed and developed using GPS, compass, and ultrasonic sensors. The GPS and compass sensors were used to navigate the robot, while the ultrasonic sensor was used to detect obstacles in its path. The prototype was tested using a Unity simulation, which demonstrated that the A* algorithm performed well in a dynamic environment.

However, during testing, it was discovered that the NEO-6M GPS module was not sufficient for navigating the prototype due to its limited accuracy range of 2.5 meters and its inability to work inside buildings. Furthermore, compass sensors were not available in the Mongolian markets, which led to the use of a smartphone's GPS and compass sensors. Although the accuracy of the smartphone's GPS was the same as NEO-6M GPS, it worked well indoors, which was a significant advantage for navigating the robot through buildings.

The ultrasonic sensor was used to detect obstacles in the robot's path. The sensor was mounted on the front of the robot and could detect obstacles up to 7 meters away. The sensor's accuracy was tested by placing various obstacles in the robot's path, and the sensor was able to detect all obstacles accurately.

The A* algorithm was tested extensively in the Unity simulation, and the results were promising. The algorithm was able to find the most efficient and shortest path between two points, while also avoiding obstacles in the robot's path. The algorithm's ability to adapt to dynamic obstacles was also tested, and it was able to adjust its path in real-time based on the sensor's input.

Overall, the results of the simulation and prototype tests were positive. The proposed algorithm performed well in a dynamic environment, and the prototype was able to navigate through obstacles effectively. The use of a smartphone's GPS and compass sensors proved to be a practical solution for navigating along the path, where the accuracy of the GPS signal was limited. In conclusion, the proposed algorithm and prototype have significant potential for enhancing the navigation capabilities of autonomous street cleaning robots in various environments.

Chapter 4. Conclusion

In conclusion, the thesis aimed to implement algorithms for a street cleaning robot that can operate autonomously and efficiently navigate while avoiding obstacles. The research conducted focused on both hardware and software components. In terms of hardware, the study looked into location determination, data processing, electric motors, kits, and batteries. The software component included path planning algorithms, with the A* algorithm being the most effective during simulation testing conducted on Unity.

The simulation testing demonstrated that the proposed algorithm is capable of handling complex environments and dynamic obstacles. The algorithm was able to generate efficient paths for the robot to follow, even in the presence of obstacles added to the environment during testing. The positive results of the simulation testing allowed for the exploration of two possible solutions for the prototype. After careful consideration, GPS, compass, and ultrasonic sensors were implemented.

The prototype component selections were tested individually, such as motors and sensors, by using low-level software codes. The subsystem testing followed by connecting the individual component codes to develop the high-level A* algorithm.

To sum up, the prototype testing was a success, as the proposed algorithm performed efficiently in a dynamic setting, and the robot prototype demonstrated effective obstacle avoidance capabilities. The practical solution of using GPS and compass sensors from a smartphone proved helpful in navigating the robot through buildings with limited GPS signal accuracy. This algorithm and prototype have significant potential for enhancing the navigation capabilities of autonomous street cleaning robots in various environments.

4.1 Limitation and future improvements

While the proposed algorithm and prototype showed promising results, there are still limitations to be addressed. The first limitation is related to GPS accuracy. Currently, the GPS accuracy is limited to around 2.5 meters, which is not sufficient for accurate navigation in dynamic environments. In the future, an improvement can be the use of RTK GPS, which has an accuracy of 1cm. However, due to the lack of supply and the high cost of around $300, we were not able to implement this solution in our prototype.[58]

Another limitation is related to the use of ultrasonic sensors for obstacle detection. While the sensors were effective in detecting obstacles, they have limitations in accurately determining the size of the obstacle. Therefore, a potential improvement could be to use lidar sensors instead of ultrasonic sensors. The average price of a lidar sensor is around $200, which is more expensive than the ultrasonic sensors used in our prototype.[59]

Lastly, the limitations of economics and the Mongolian market were also a challenge in the development of our prototype. The cost of implementing advanced technologies and sensors is relatively high, and the availability of such equipment is limited in Mongolia. Therefore, there is a need for further investment in research and development in this area to overcome these limitations and to advance the capabilities of autonomous street cleaning robots in various environments.

Reference

1. Moonpreneur. Arduino Used in Robotics. [Internet]. Moonpreneur; 2022 [cited 2023 Apr 30].
2. ElProCus. How GPS System Works? [Internet]. ElProCus; 2022 [cited 2023 Apr 30].
3. Encyclopedia Britannica. Global Positioning System (GPS) [Internet]. Encyclopedia Britannica; 2022 [cited 2023 Apr 30].
4. Schaumann G, Russer P, Weigel R. Microcontroller-Based TDR System for Detection and Localization of Leaks in Plastic Pipelines. Graz: Graz University of Technology; 2005.
5. u-blox. NEO-M6 Series. [Internet]. u-blox; 2023 [cited 2023 Apr 30].
6. MediaTek. MT3339. [Internet]. MediaTek; 2023 [cited 2023 Apr 30].
7. SkyTraq. Venus838LPx-T. [Internet]. SkyTraq; 2023 [cited 2023 Apr 30].

8. Quectel. L80. [Internet]. Quectel; 2022 [cited 2023 Apr 30].
9. Circuit Digest. Best GPS GSM Module Selection Guide. [Internet]. Circuit Digest; 2022 [cited 2023 Apr 30].
10. Chen CH, Liang LW. Ultrasonic Sensors for Nondestructive Testing and Evaluation. In: Encyclopedia of Structural Health Monitoring. Cham: Springer; 2019. p. 1-7.
11. Huang JH. Ultrasonic Sensors: Technologies and Applications. London: Academic Press; 2011.
12. Nakamura K, Yamada H. Ultrasonic Transducers: Materials and Design for Sensors, Actuators and Medical Applications. Oxford: Woodhead Publishing; 2012.
13. Patel KD, Patel MD. Object Detection and Obstacle Avoidance Robot using Ultrasonic Sensor. International Journal of Scientific Research. 2017;6(6):276-9.
14. Kamath AG, Kanal LN. Lidar Sensor in Autonomous Vehicles. International Journal of Engineering Research and Technology. 2021;10(4):450-7.
15. Wikipedia. Inertial Measurement Unit. [Internet]. Wikimedia Foundation; 2023 [cited 2023 Apr 30]
16. Davies JH. Microcontroller Basics. Oxford: Newnes; 2004.
17. Barr M, Massa A. Programming Embedded Systems: With C and GNU Development Tools. Sebastopol: O'Reilly Media; 2006.
18. Yiu, J. (2013). The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors. Newnes.
19. Gadre, D. V. (2000). Programming and Customizing the AVR Microcontroller. McGraw-Hill Education.
20. Reese, R. B. (2009). Microcontrollers: From Assembly Language to C Using the PIC24 Family. Cengage Learning.
21. Noergaard, T. (2005). Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers. Newnes.
22. Fisher, M. (2016). ARM Cortex-M4 Cookbook. Packt Publishing.
23. Evans, B. (2011). Beginning Arduino Programming. Apress.
24. Halfacree, G. (2021). Getting Started with Raspberry Pi Pico. Raspberry Pi Press.
25. Mazidi, M. A., McKinlay, R. D., & Causey, D. (2009). The 8051 Microcontroller and Embedded Systems: Using Assembly and C for PIC18. Pearson Education.
26. Mehta, A., & Singh, R. (2014). Microcontroller basics, architecture and applications: A review. International Journal of Electronics and Communication Engineering & Technology, 5(6), 23-29.

27. Khan, A. M., & Islam, M. S. (2013). A review on microcontroller based control of switched reluctance motor. International Journal of Scientific & Engineering Research, 4(6),
28. Kallappa, N. B., & Biradar, M. V. (2015). From apposite preference to specific application: An overview of microcontroller unit. International Journal of Technical Research and Science, 1(2), 108-112.
29. OurPCB. (n.d.). What is Motor Driver? Retrieved April 30, 2023
30. Robocraze. (n.d.). What is Motor Driver? Retrieved April 30, 2023
31. Dijkstra's algorithm. (2022, March 2). In Wikipedia. Retrieved April 30, 2023
32. Gulzar, M. A., Khan, A. A., Din, M. U., & Khan, M. U. (2018). A Review on the Implementation of Dijkstra's Algorithm. Symmetry, 10(10), 450. doi: 10.3390/sym10100450
33. Singh, D., & Singh, A. (2021). Improved Dijkstra's Algorithm Using A* Algorithm for Shortest Path Computation in Real-Time Traffic Control System. International Journal of Advanced Computer Science and Applications, 12(1), 97-105. doi: 10.14569/ijacsa.2021.012011
34. u-blox. (2010). NEO-6 Data Sheet (GPS.G6-HW-09005). Retrieved April 30, 2023,
35. Last Minute Engineers. (n.d.). NEO-6M GPS Module with Arduino Tutorial. Retrieved April 30, 2023
36. u-blox. (2010). NEO-6 Data Sheet (GPS.G6-HW-09005). Retrieved April 30, 2023
37. Shenzhen H.C Starck Electronic Co., Ltd. (2015). Ultrasonic Distance Sensor HCSR04 Datasheet. Retrieved April 30, 2023,
38. Arduino. (n.d.). Arduino Uno Rev3. Retrieved April 30, 2023
39. JavaTpoint. (n.d.). Arduino UNO. Retrieved April 30, 2023
40. Adafruit Industries. (n.d.). Lithium Ion Polymer Battery - 3.7v 2500mAh. Retrieved April 30, 2023
41. SparkFun Electronics. (n.d.). GPS-RTK Board - NEO-M8P-2 (Qwiic). Retrieved April 30, 2023
42. Electric motor. (2022, March 22). In Wikipedia. Retrieved April 30, 2023
43. Rozum Robotics. (n.d.). How to Choose a Robot Motor: Types and Technical Characteristics. Retrieved April 30, 2023
44. Robocraze. (n.d.). Types of Motors used in Robotics. Retrieved April 30, 2023
45. 3D Insider. (n.d.). 3D Printing and 3D Printer Reviews. Retrieved April 30, 2023
46. Hai Prasaath K. Choosing Batteries for Robots [Internet]. Engineers Garage. 2019.
47. Trakkit GPS. How GPS Works [Internet]. Trakkit GPS. [cited 2023 Apr 30].

48. Osoyoo [Internet]. Osoyoo. 2018 Sep 18 [cited 2023 Apr 30]. Micro:bit Lesson Using the Ultrasonic Module.
49. Shekar P. LIDAR and Time-of-Flight: Part 2 - Operation [Internet]. Microcontroller Tips. 2018 Oct 12 [cited 2023 Apr 30].
50. CircuitBread. How do Accelerometers and Gyroscopes Work? [Internet]. CircuitBread. [cited 2023 Apr 30].
51. Circuit Basics. Introduction to Microcontrollers [Internet]. Circuit Basics. [cited 2023 Apr 30].
52. Gaur V, Gupta R, Soni SK. Recent Developments in the Field of Computer Vision [Internet]. Symmetry. 2018 Oct 15 [cited 2023 Apr 30].
53. Wikipedia. Dijkstra's Algorithm [Internet]. Wikimedia Foundation; 2022 Nov 12, 21:31 UTC [cited 2023 Apr 30].
54. Wikipedia. A* Search Algorithm [Internet]. Wikimedia Foundation; 2023 Apr 15, 21:20 UTC [cited 2023 Apr 30].
55. Last Minute Engineers. L298N DC Stepper Motor Driver Arduino Tutorial [Internet]. Last Minute Engineers. [cited 2023 Apr 30].
56. Jain P, Kaur M. A Review on Compass Sensor Technologies for Robotics Applications. Int J Innov Technol Explor Eng. 2018 Mar;7(5):98-101.
57. Science ABC [Internet]. Science ABC. What is the Index of Refraction? [cited 2023 Apr 30].
58. SparkFun. What is GPS RTK? [Internet]. SparkFun. [cited 2023 Apr 30]. Available from: https://learn.sparkfun.com/tutorials/what-is-gps-rtk/all
59. ROBOTIS [Internet]. ROBOTIS. 360 Laser Distance Sensor LDS-01 (LiDAR) [cited 2023 Apr 30].
60. National Geographic Education. Latitude [Internet]. [date unknown; cited 2023 May 6].