School of Engineering, Computer and Mathematical Science

Indoor localization Mobile Robot Project Report

Supervisor: Dr. Xuejun Li

Submitted by Sean Shi
Student ID: 16932688

Contents:

Acknowledgements

I would like to thank my project supervisor Dr Xuejun Li, for all the support and help he provided in the process of the project. He is kind and helpful to guide me through the project and give me suggestions when I needed to implement on the design.

Also, I would like to thank my group member Mr Oscar Aspite. His encouragement and assistance helped me through a challenging time when I extremely preoccupied. He helped me brainstorming about this industrial project idea in the first place and expressed his interest to work together in a team to design the mobile robot.

On a more personal note, I would like to thank all the friends, lectures, and technicians from the Auckland University of Technology for all their support and suggestions on my project. Thanks to Dr David Wilson who gave us critical advises about our project methodology and approaches in the mid-term presentation.

Abstract:

Artificial intelligence is the mainstream in both computer science and software engineering field nowadays, this also leads the rapid development in the robotic area covers human production and living. Intelligence robots are taking important roles in people’s daily life, such as a cleaning robot, UAV, a mobile robot for building recognition and disaster recovery.

This project aims to demonstrate the feasibility of building a mobile robot planted with indoor localization functionality using ROS and Kinect Camera approaches. The mobile robot model can localize itself and map the surrounding area, planning the path which allows them to perform specialised tasks.

The report will be structured as follows: First, the study of the ideal mechanical design of the mobile robot, the introduction and background. The second part demonstrates the literature review of the methods approaches, including, path planning, localization, and SLAM. The third part is about the software design prototype of the mobile robot, the comparison of the systems and tools. The final part is to discuss the experiments and results, analysis of the developed solution and future development can be made.

Introduction:

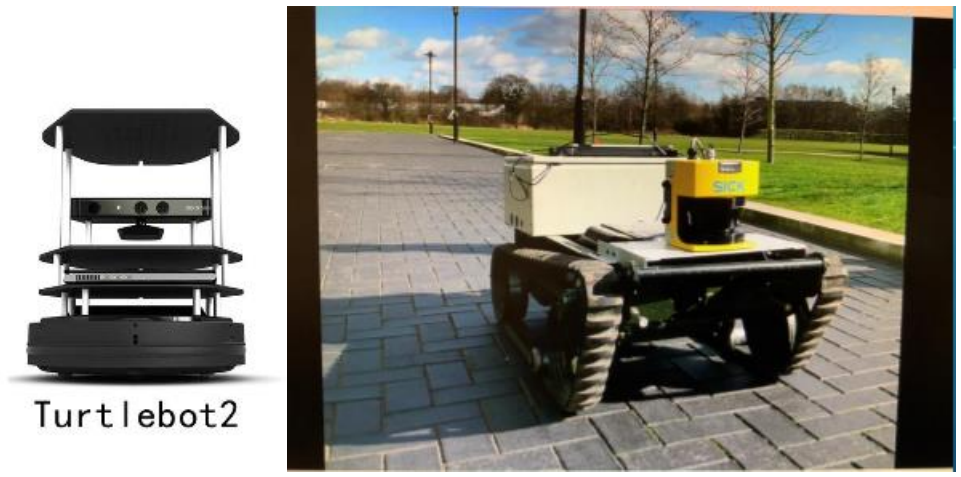


Figure 1 Mobile Robot

As technology rapidly advancing in development, there is an increase in interest for robotic and automation in our life. Now we can see that technology in our daily life, such as drones, Smart Home, IoT, self-driving cars like Tesla Model 3. Mobile robots play a significant role in fields like industrial, medical, military and disaster recovering, daily care and other areas.

However, there are issues the mobile robot need to be able to solve. First, the mobile robot can be able to know the location of it currently at, which is robot localization. Second, the mobile robot must know the surrounding, which is robot mapping. Lastly, the mobile robot needs to know where the destination is, which is robot navigation or path planning. The important task for a mobile robot is to achieve the same level that the human visual system can do.

Those issues can be called dead reckoning, which is solving the direction of the robot, SLAM can solve the localization and creating a map, path planning can lead the mobile robot to the destination.

Recent years, several robot companies have launched their wheeled mobile robot such as Turtlebot2 and PioneerLX. Turtlebot2 is the most popular mobile wheeled robot around the world. Nevertheless, the cost of the mobile wheeled robot is expensive, and the hardware and software come with it are not all open sourced.

As the SLAM problem is one of the trendy topics in the robotics area, as solving this issue, we can achieve the autonomous mobile robot in the end. SLAM problem has been raised since 1988, till now it has been over 30 years. In the early stage of the SLAM development, the research focused on using filter theory to minimize the noise of map landmark points and moving body poses. Later, researchers began to study the method in structure from motion (SfM) to solve SLAM problems. However, based on the various types of sensors used in SLAM and different method of installing them, the results can be varied.

SLAM can be divided into two categories: Laser SLAM and Vision SLAM. The laser sensor can collect direct distance information in the environment to achieve relative positioning. However, Vision SLAM is different. It is difficult for Vision SLAM to gather information relative to the environment, and it can estimate the pose and location by calculating the current position and post-change by comparing two or more frame of images.

There are several open-sourced SLAM algorithms available on the internet for researching and learning. The Laser SLAM has well-developed research and projects, but Vision SLAM is still under the early development stage. Now because of the technology got better and the rapid development of mobile computing, image processing and deep learning got popular in the industry, there is space for developing a high-performance computing and communication network at low cost, which is a huge benefit in development and implementation of mobile robot and SLAM technology.

The purpose of this report is to design a mobile robot which can localize itself in an indoor unknown environment and have a high-level understanding of the surrounding, position and mapping of the current location and path planning to the target destination.

Literature Review:

For indoor localization, the first thing we can do is to open our eyes, to recognize the environment around us, be able to tell the approximate location. It is same for our robot, we want to develop a computer vision program that can detect the objects around the robot, recognize the object's label, be able to tell the approximate location of the robot standing.

Localization means estimating the mobile robot's geometric feature position as it travels in the area. When it comes to localization, the term pose is also used as a reference to represent the X, Y, Z coordinates and 3D orientation (Roll, Pitch and Yaw) of the device.

Location determination system has been well known as using the Global Positioning System (GPS), GLONASS or another satellite provider, network positioning using the cell to determine the location, those solutions shared one thing in common: they are designed for outdoor usage. And currently, there is no indoor positioning system, the GPS does not work inside the building since there is not any direct light of sight with the satellite.

For applications where area does not have a network connection, GPS signal in the area, no pre-installed infrastructure, we use the embedded camera on a mobile device to estimate the location. This method allows us to develop a system that does not rely on the infrastructure installation. This problem also known as the SLAM problem, it stands for simultaneous localization and mapping, it computes the robot's poses and the map of the environment at the same time.

Dead Reckoning:

Dead Reckoning records the position vector of the mobile robot was at the previous time, combines with information of direction of pointing and current moving speed, it can simulate the current position based on mobile robot travel distance and trajectory.[2] However, the accuracy of the result can be affected by the environment, such as the surface roughness and the friction of the wheel when it travels.

This method can be used for estimating the mobile robot movement and travel path. It can be considered as the continuous accumulation of displacement vector. The estimation of the robot position and pose is based on the previous calculation in the sampling interval, and therefore the error may carry over.

Though the changing of the wheels at each movement, we can use this information to calculate the trajectory of the mobile robot and get a description of the current pose. There are many other approaches to dead reckonings, such as using gyroscopes, accelerometers etc, but they all follow the same calculation for estimation.

After research of the dead reckoning, to reduce and minimize the errors may occur in the calculation process of the wheel speed angular velocity and rotation angles, there is an approach where they added inertial motion information to Karman filter, uses speedometer data and inertial information to generate much accurate state estimation.

SLAM:

SLAM stands for Simultaneous Localization and Mapping, refer to establish a map of the unknown environment during mobile robot movement and mapping process using SLAM sensors onboards and localized its position in the generated map simultaneously. There are many algorithm approaches to SLAM, including gmapping, hector_slam and tinySLAM.

Gmapping is the most widely used 2d SLAM method, by using the RBPF method so it can understand the particle filter algorithm. The particle filtering method generally requires a large number of particles to get good looking results, but this will potentially increase the complexity of the computational assumption. The resampling process of the particles will substitute for particle dissipation problem, which means the large particle with heavy weight are significant in the process, but the small light weight particles will be ignored and disappeared during this process. The introduction of adaptive resampling technology can reduce this particle dissipation issue. When calculating the particle distribution, it not only depends on the movement of the mobile robot (robot odometer) , but also takes the current observation into account, therefore it reducing the uncertainty of the robot position in the particle filtering step. [5]

The advantage of gmapping is it can construct good and reliable map in corridors and low-features scenes due to the odometer on the mobile robot. Also, the odometer limits the application of mobile robot when it comes to uneven areas or drones. Another disadvantage is it cannot be used in large scale scenes, especially when there are many particles, because it consumes lots of computation power.

Hector_slam algorithm uses the obtained map to optimize the laser beam lattice, estimate the representation of the laser point on the map. Furthermore, the scan matching method uses Gauss-Newton method to solve where scan-match method in gmapping is using the gradient descent method to estimate the position and pose of the mobile robot. The scan matching method in hector_slam finds the projected laser spot into the existing conversion map (x, y, theta) in order to avoid a local minimum rather than the global optimum occurs. [6]

tinySLAM, also known as coreslam, is a SLAM implementation method based on the Monte Carlo positioning algorithm which simplifies SLAM into two parts: distance calculation and map update. In the first part of this algorithm, each scan input uses a particle filter algorithm to calculate the distance. The particle filter matcher is used to match the laser with the current map. Each filter particle represents the possible pose description of the mobile robot and the corresponding probability weight, which depends on the distance calculation. The second part of the tinySLAM add the scanned line into the current map and update it. When the obstacles appear, it will draw a set of adjustment point around the obstacle points. [7]

hector_slam is a very good algorithm and does not require odometer data of the mobile robot, but because hector_slam matches the scan points by relying on high-precision lidar data, the Kinect camera v2 sensor with a small scan angle and a large particle noise therefore the constructed map is not reliable. The gmapping algorithm is currently the most widely used method for laser 2d slam. Due to its simple calculation, it is more suitable for application in the field of mobile robots, but it highly requires flatness surface. The tinySLAM algorithm is easy to code in the program, but there is no loopback detection, once the calculation has a large error, it will carried through the process and the map will update the incorrect position of the mobile robot.

Kinect based SLAM, which it came up in the last few years since the Kinect sensor is available that a camera and a depth sensor, so we can get visual information and depth information available at the same time, and this allows to build accuracy high-resolution model of the surrounding environment from the mobile robot.

Path Planning:

In the mobile robot path planning algorithm, it uses the A* method. The A* algorithm is an efficient path search algorithm for mobile robot. It uses a heuristic function to estimate the distance between the current position of the robot on the map and the target destination. The user can choose the direction to search, if it fails, it will choose another path to continue searching until the path is found. [8]

Hardware Design:

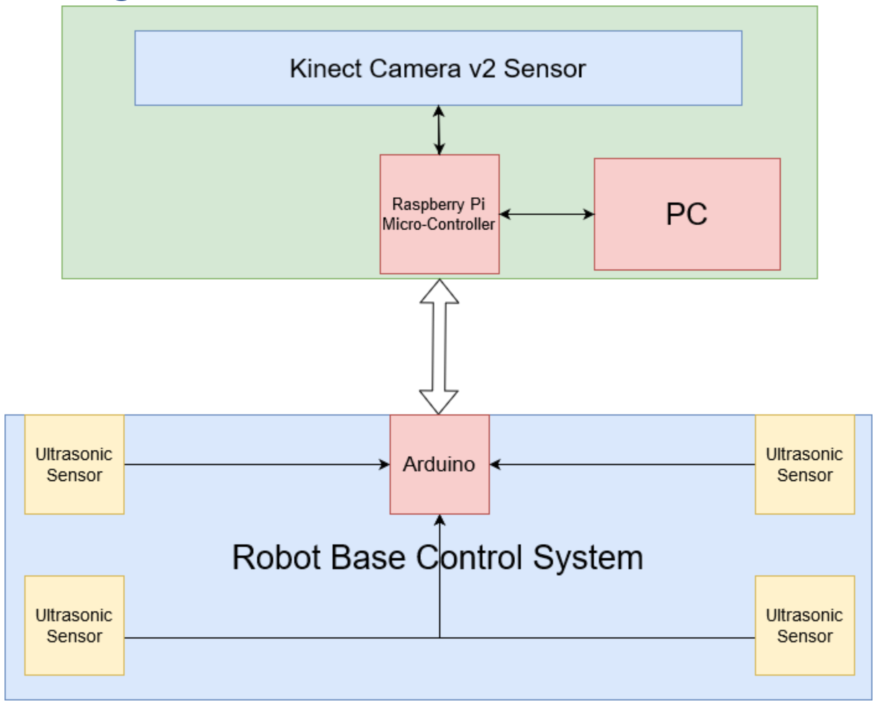


Figure 2 Hardware Design Diagram

The hardware design of the mobile robot divided to two parts: upper system equipped with ubuntu system on the Raspberry Pi with the ROS system installed for the main algorithm of the mobile robot (localization and mapping). The lower system equipped with the Arduino board with the ultrasonic sensor which avoiding the obstacle.

I used the Microsoft Kinect camera version 2 as the eye of the mobile robot. The sensors on the camera can generate a point could image if the surrounding environment which can be used in the ROS system for mapping and localization.

Microcontroller:

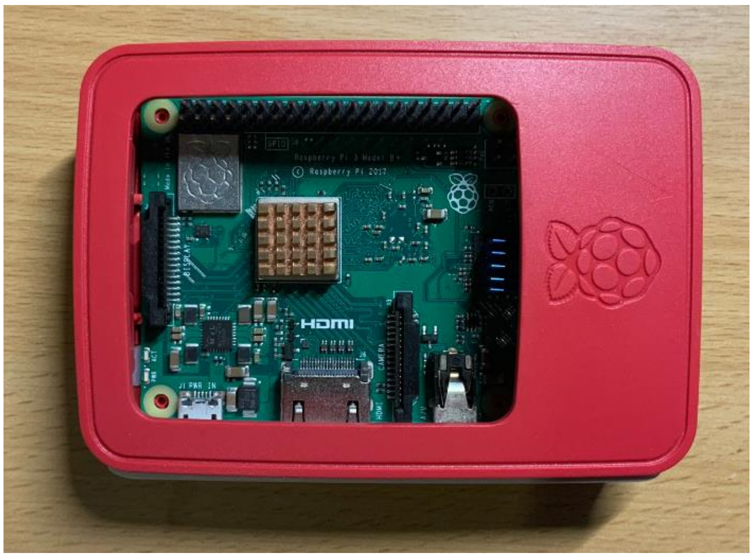
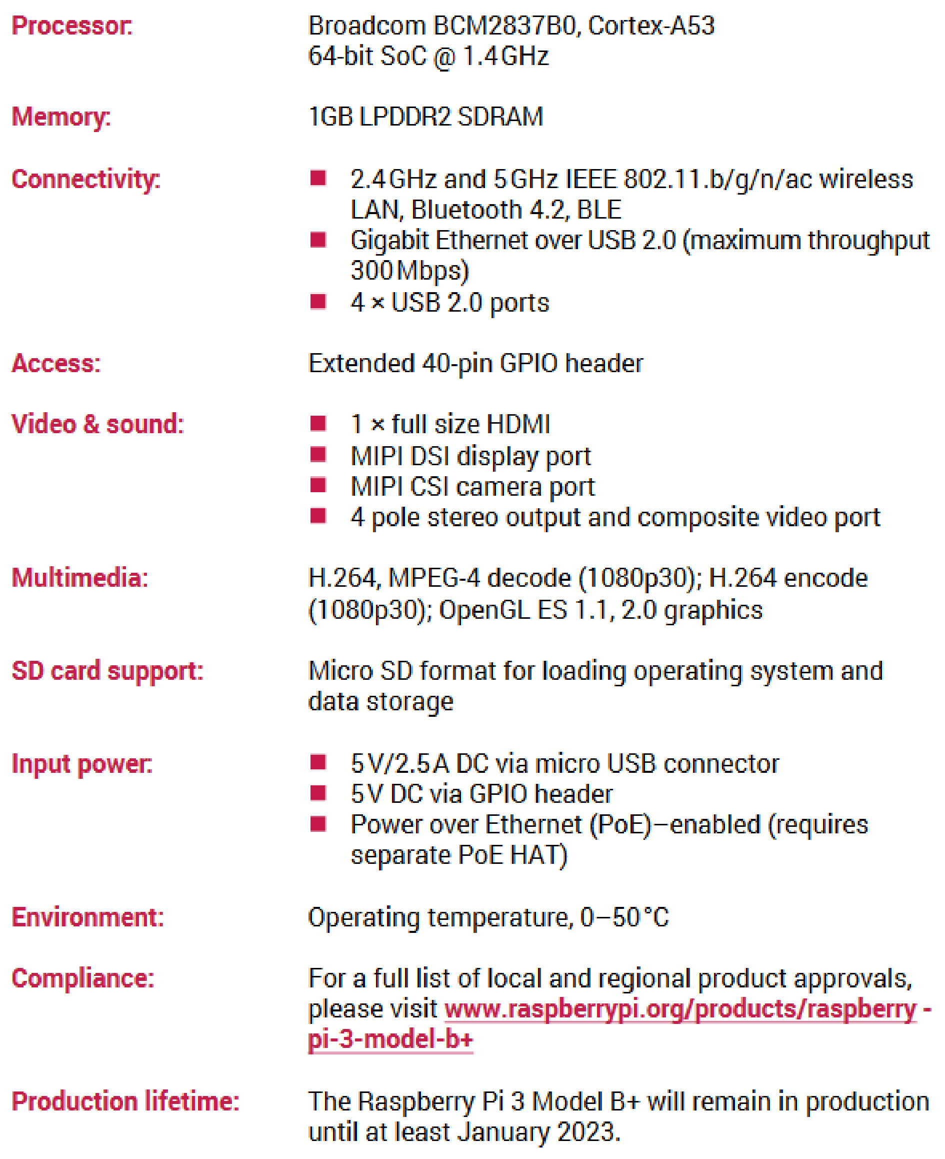


Figure 3 Raspberry Pi 3B+

Raspberry Pi is a cheap ARM architecture microcontroller board which have low power cost and high compatibility features. It commonly is used as installing ubuntu system for IoTs and minicomputer with excellence computing ability. [2]



Processor:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memory:	1GB LPDDR2 SDRAM
Connectivity:	■ 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE ■ Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps) ■ 4 × USB 2.0 ports
Access:	Extended 40-pin GPIO header
Video & sound:	■ 1 × full size HDMI ■ MIPI DSI display port ■ MIPI CSI camera port ■ 4 pole stereo output and composite video port
Multimedia:	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support:	Micro SD format for loading operating system and data storage
Input power:	■ 5V/2.5A DC via micro USB connector ■ 5V DC via GPIO header ■ Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature, 0–50°C
Compliance:	For a full list of local and regional product approvals, please visit www.raspberrypi.org/products/raspberry -pi-3-model-b+
Production lifetime:	The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.

Figure 4 Raspberry Pi Datasheet

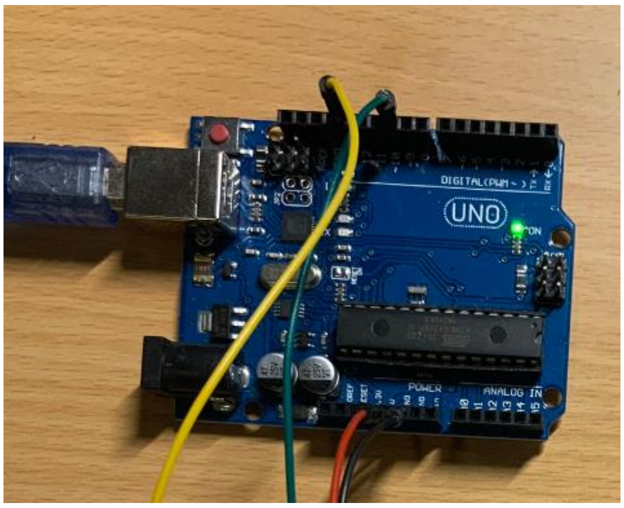


Figure 5 Arduino Uno Board

For using the ultrasonic sensor on the mobile robot base, the Arduino uno board is most common used for this application. It is light weight and do not have much power consumption, which compare to Raspberry pi it is a better opinion. Also, it can only be coded in Arduino program using computer and it does not have its own OS, it reduces the complexity of this project.

Kinect Camera v2 sensor:

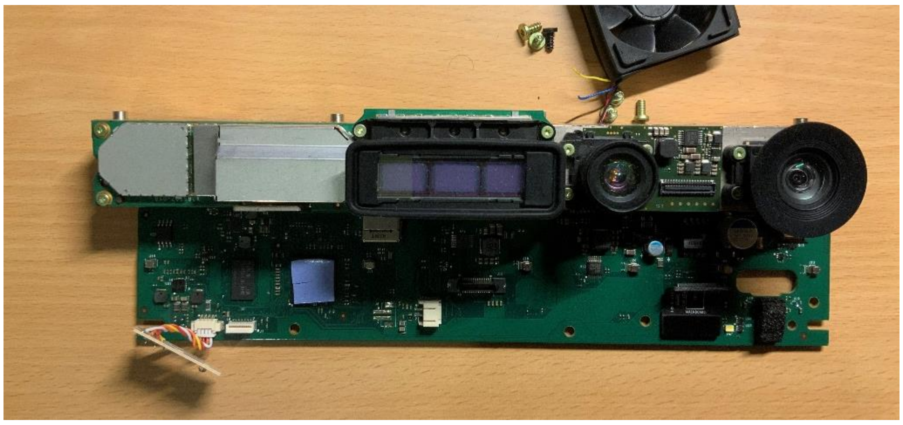


Figure 6 Kinect Camera v2

Microsoft Kinect Camera has RGB sensors which can help to collect the colourful information of the scene, the left and right sensors are the infrared emitter and infrared depth sensor that collect depth information to construct the distance from the object in the scene to the camera. The RGB camera on the Kinect supports 1920*1080 resolution and the IR sensor can support 640*480 image. Those feathers ensure we collect clear image and enough data to construct a detailed scene, produces less blurry objects and less error. Also, the Kinect camera equipped with 4 microphones at a different place, which they compare the collected audio information and denoise the channel, which can be used to collect audio information to able voice recognition and sound localization. [3]

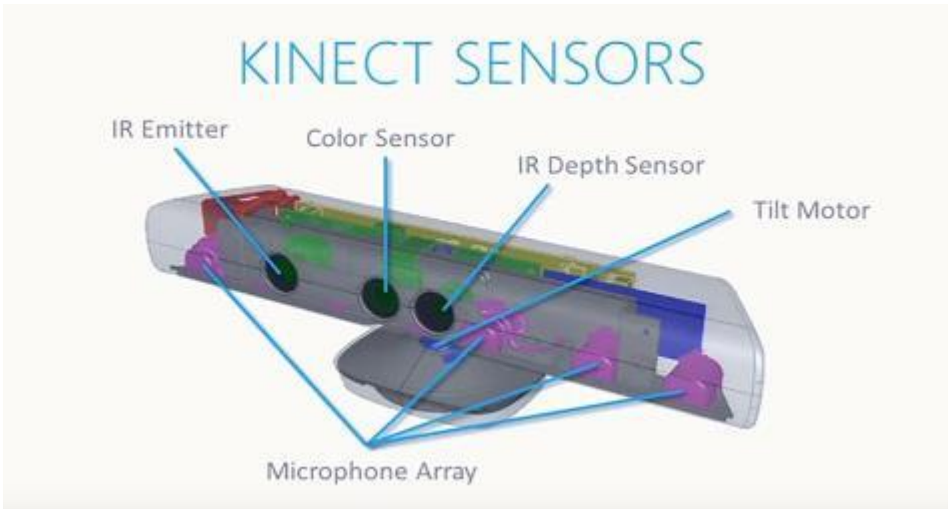


Figure 7 Kinect camera internal Structure diagram

The infrared emitter and infrared SMOS camera combine to perform like 3D structure depth sensor to collect the depth data of the environment. Those camera sensors can read the projected infrared spectrum and calculate the depth information from the reflection spot.

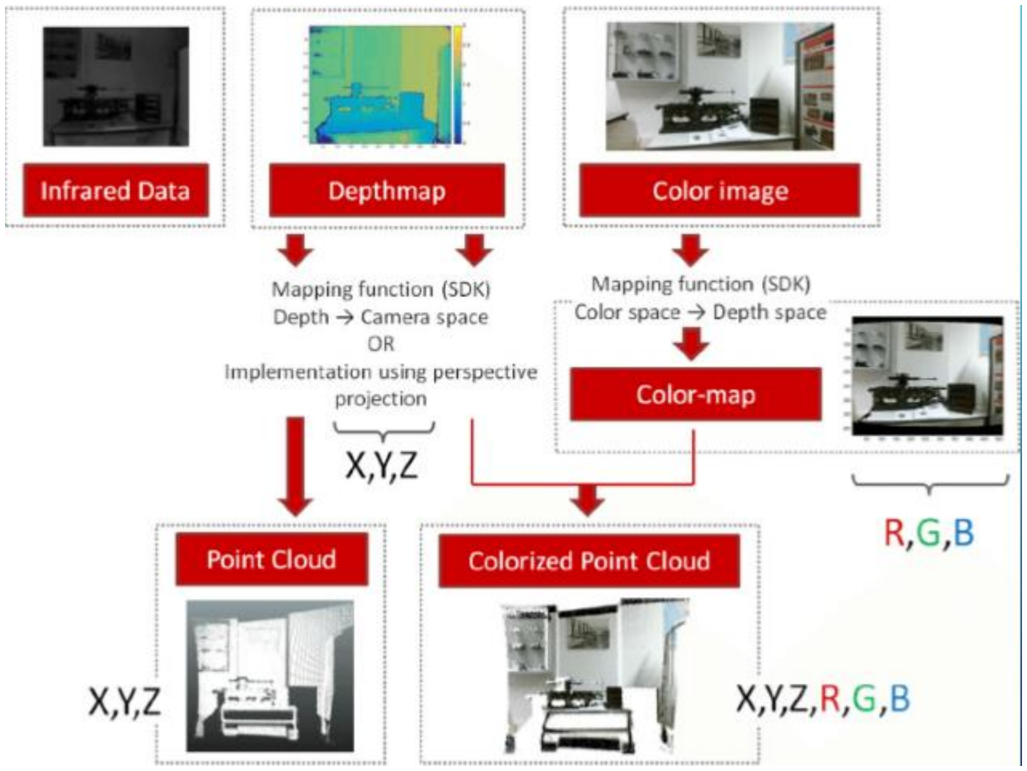


Figure 8 Kinect Camera Image Processing

Ultrasonic Sensor:

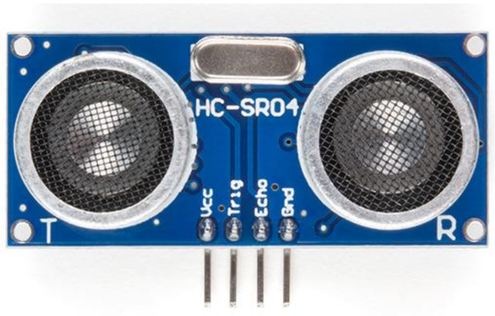


Figure 9 HC-SR04 Ultrasonic Sensor

Above graph shows the HC-SR04 ultrasonic sensor module used in this project. This ultrasonic sensor has high resolution within 1mm range to a maximum of 0.5 mm. After field testing in a real-life scenario, this can be reliable to achieve 2cm to 400 cm range. This sensor can work under 5V and 15mA current which does not require to drain more power from the system. **[1]**

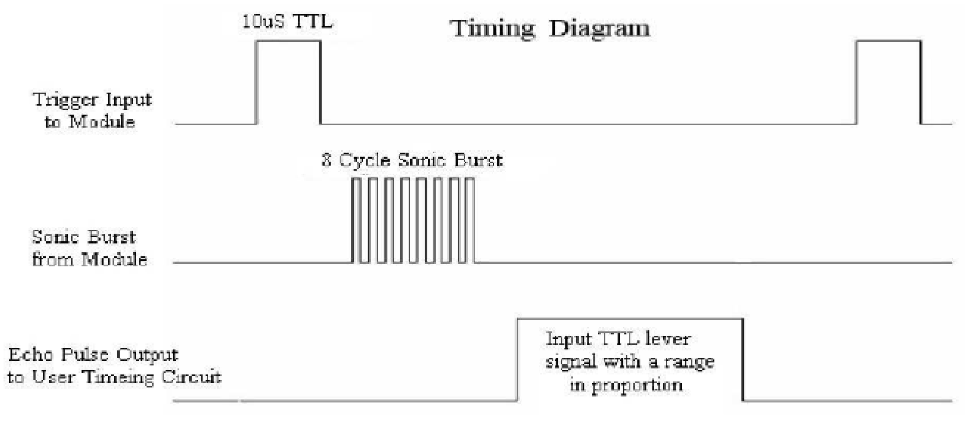

10uS TTL
Timing Diagram
Trigger Input to Module
8 Cycle Sonic Burst
Sonic Burst from Module
Echo Pulse Output to User Timeing Circuit
Input TTL lever signal with a range in proportion

Figure 10 Ultrasonic sensor timing diagram

Ultrasonic sensor required the microcontroller to send a 10us high pulse signal to the trigger input, and the sensor starts ranging and constantly send an 8-cycle burst of 40 kHz ultrasound and raise its echo. **[1]** The echo is the range and pulse width in proportion, and the formulae to calculate the range is the time difference between the sending signal and receiving echo signal.

$$Range = \frac{high\ level\ time * Velocity}{2} \qquad Velocity = 343m/s$$

The microcontroller gives a short low pulse before sending the 10us high pulse signal to ensure a clean high pulse. High pulse represents the time interval from the sending of the ping to the reception of its echo off an object.

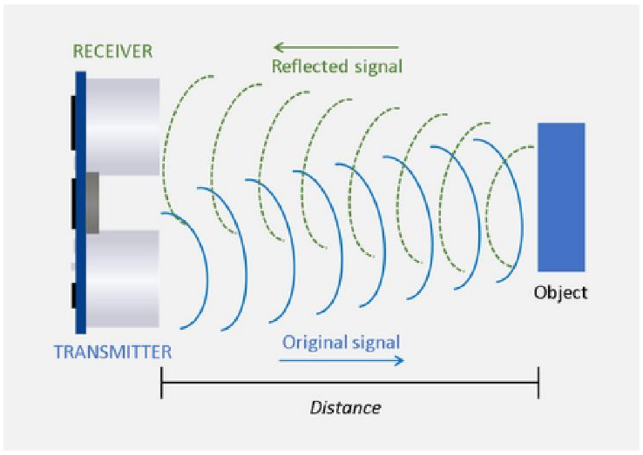


*Figure 11 Calculating the distance of the ultrasound **[1]***

To avoid the interrupt of multichannel ultrasound, we need to set the time interval for those two channels sending and receiving the signal. Due to the decay of the ultrasound signal, the range of detection is limited to 5m, which means the distance of the ultrasound travelling is no more than 10m. The time for the ultrasound travels 8m is 29.15ms, which means to avoid the interrupt of two channels on the sensor, time interval between them should be bigger than 29.15ms.

Software Design:

Overview:

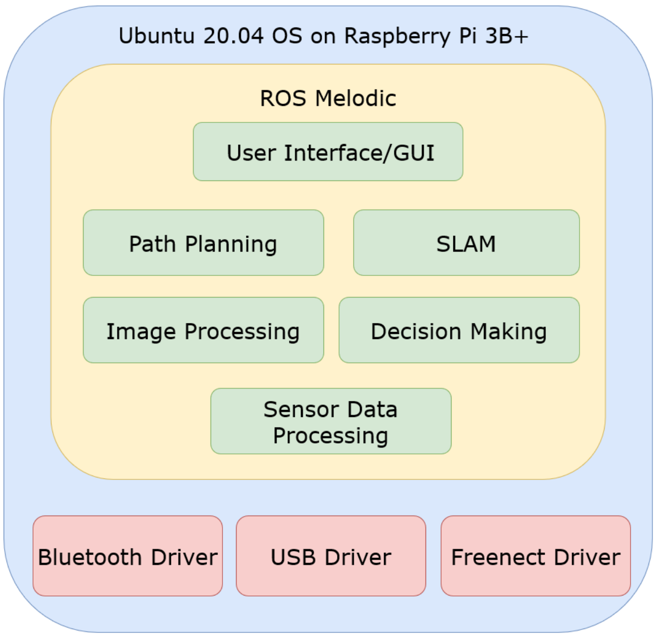


Figure 12 Software System Diagram Design

As shown in the graph above, the main objective for the software algorithm has good user interface design, path planning and SLAM. The ubuntu system can provide the Kinect camera sensor driver, and with installed ROS system on ubuntu, the ROS can read the data from the Kinect camera in the simulation.

In the ROS community, there are many open-source packages for the user to download and import to the system, which reduce the cost of development and easier to build the algorithm. This allows me to add custom program into the algorithm for the mobile robot such as image processing, sensor data processing, path planning and SLAM. Those are organized in the form of ROS nodes.

Ubuntu OS:

Ubuntu OS is based on Debian and GNOME environment and Linux OS. The purpose of the Ubuntu OS is to deliver to the user with a stable and reliable operating system that is based on software build. Now ubuntu OS is popular around open source community and mainly used python language to able IoTs and other features.

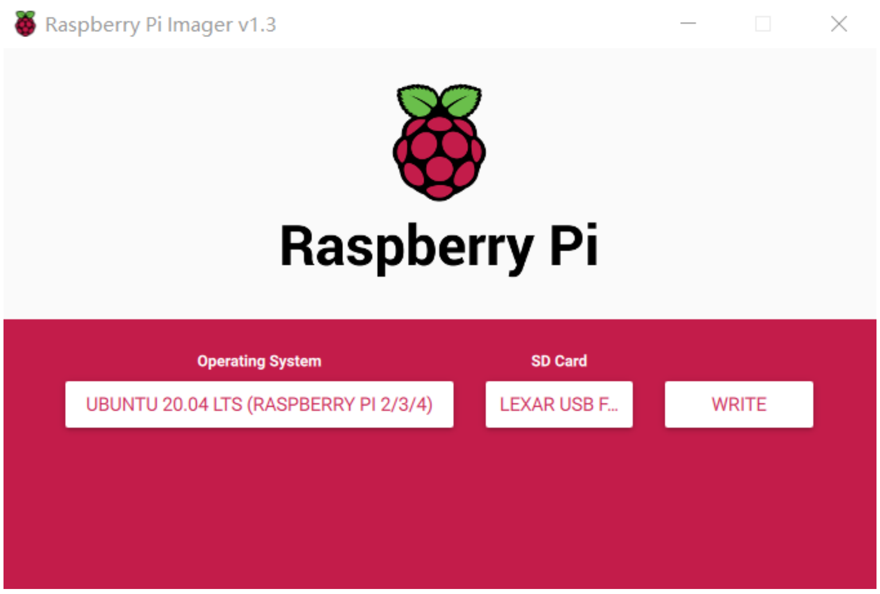


Figure 13 official firmware writing software

Raspberry Pi imager allows the user to burn the ubuntu system image into a microSD card, and be able to install the system when booting the Raspberry Pi. For this project, we stuck with the latest firmware which is Ubuntu 20.04 LTS and burned it on Raspberry Pi 3B+.

ROS

ROS stands for Robot Operating System; it is a set of computer operating system architecture specially designed for robot software development. It is an open source operating system (post-operating system) that provides services similar to the operating system, including hardware description, low-level driver management, execution of common functions, program distribution package management, it also provide some tools and libraries for acquiring, establishing, writing and executing multi-machine fusion programs, organize multiple codes responsible for different tasks in the form of nodes, and send data based on TCP/IP protocol transmission between nodes in the form of topics and receive.

ROS provides a loosely coupled program management mechanism based on the subscription publishing model on ROS website. Therefore, this can improve the code reusability in the field of robot research and development.

ROS Architecture has three levels: Computation Graph Level, FileSystem Level and Community Level. [4]

The navigation function package set used in ROS has restrictions which the navigation function package set can only handle two-wheel differential drive and fully driven robots. The robot needs to publish the sensor position relationship about all joints. The robot must be able to publish linear velocity and angular velocity. The robot must have a flat lidar to complete the map construction and positioning.

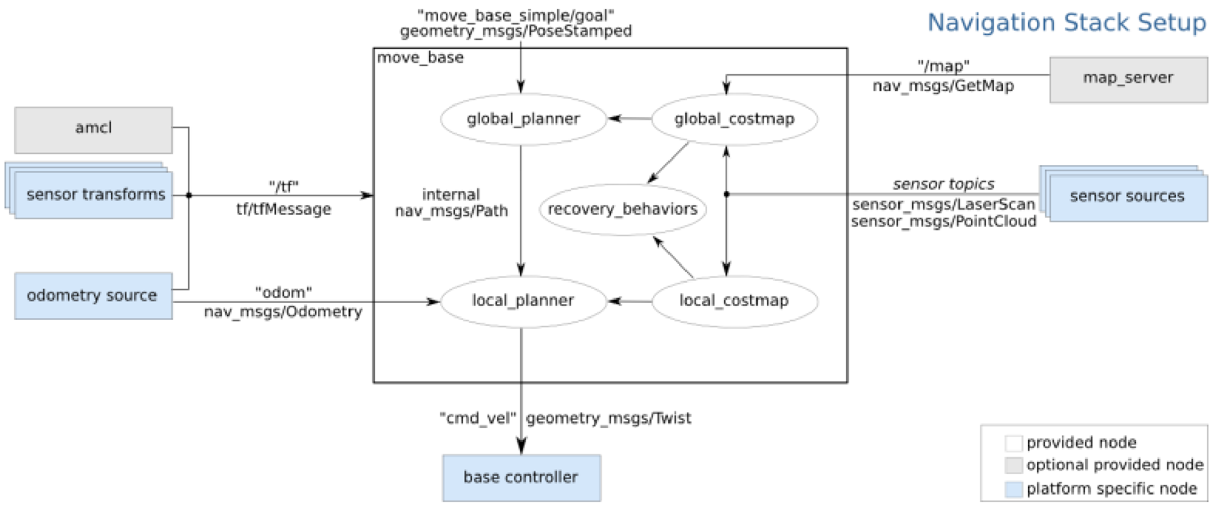


Figure 14 Overview of TF tree

According to the ROS official navigation function package set configuration, nodes can be used for mobile robot SLAM are:

Nodes	Functionality
base_controller	Base controller, responsible for data interaction with the robot chassis.
pointcloud_to_laserscan	A virtual 2D laser data publishing node that converts Kinect's point cloud data into 2D laser data.
robot_pose_ekf	Pose filter based on extended Kalman filtering, integrating mobile robot wheel speedometer track estimation data and inertial data.

camera	Data acquisition and release of Kinect based on freenect, including the relationship between point cloud data and tf tree.
slam_gmapping	2D laser slam algorithm based on gmapping to get map.
rviz	Robot status visualization tool node
tf	Release real-time poses and dependencies of each connection of the robot

3D modelling of the mobile robot in ROS:

The robot 3D model is mainly used to simulate the robot or help the developer to simplify the debugging process. In the ROS, the URDF file can be used to describe the robot 3D model.

Urdf is based on the xml format and describes the basic fields of the robot's geometric structure: links and joints. The connection describes the basic characteristics of the element, including shape, volume, color, material texture, and the joint describes the dependence of the two elements and relationships between elements including motion type (rotation, fixation, rotation, plane motion), relative position, relative angle, etc. [9]

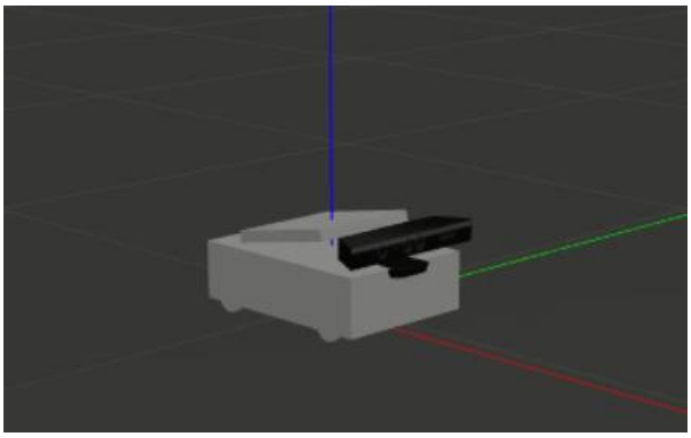


Figure 15 3D model of the design mobile robot in ROS

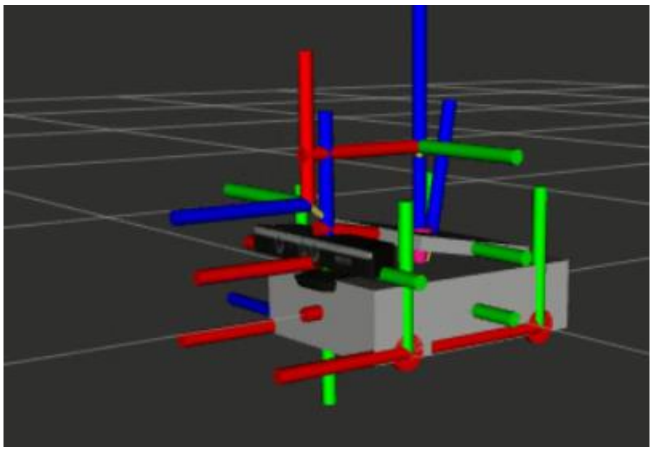


Figure 16 3D model of the design mobile robot in ROS

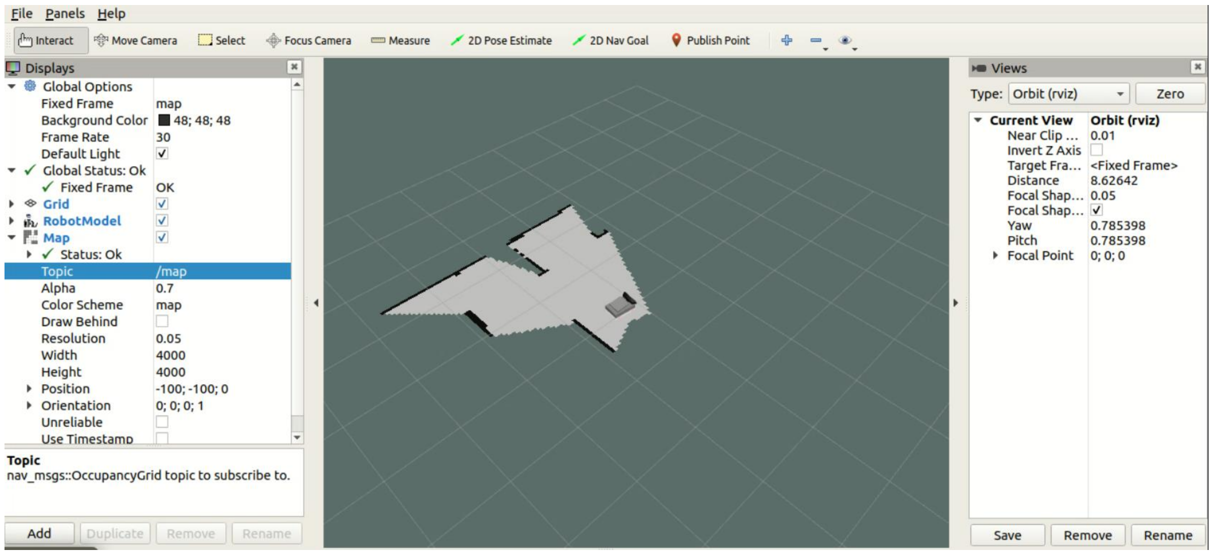
Kinect laser data in ROS:

For this project design of the mobile robot, I installed the open source freenect driver on ROS to obtain Kinect sensor data from the Kinect camera v2.

The rqt_rviz in ROS to display the point cloud image obtained by Kinect camera v2. Since Kinect camera v2 cannot directly output the laser data, it can only output point cloud data. I used the open source pointcloud_to_laserscan function package so the 3D point cloud data of Kinect camera will be converted into 2D laser data and published under the topic of scan. [9] This function package uses the pixel closest to the Kinect camera in each column of point clouds in the vertical direction as the obstacle distance in the 2D laser, and projects the minimum value in all columns to the plane where the robot coordinate system is located. [9]

Navigation and SLAM in ROS:

This part mainly uses open source algorithm function package and achieving SLAM by setting some parameters for the characteristics of the robot which reduces the difficulty of development significantly.



Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
Map	✓
✓ Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	
Resolution	0.05
Width	4000
Height	4000
Position	-100; -100; 0
Orientation	0; 0; 0; 1
Unreliable	
Use Timestamp	

Topic
nav_msgs::OccupancyGrid topic to subscribe to.

Type: Orbit (rviz) Zero

Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	
Target Fra...	<Fixed Frame>
Distance	8.62642
Focal Shap...	0.05
Focal Shap...	✓
Yaw	0.785398
Pitch	0.785398
Focal Point	0; 0; 0

Figure 17 SLAM in ROS

Above figure shows using the slam_gmapping open source function package for map construction, the node needs /tf (coordinate transformation information) and /scan topic (laser scan data), and then output to /map topic (raster map).

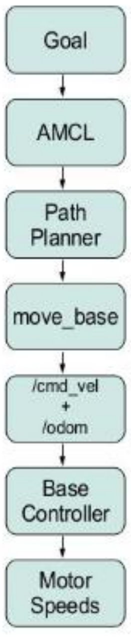


Figure 18 navigation control level diagram

Experiments and Program Debugging:

Ultrasonic Sensor Program debugging:

As shown below are debugging the Arduino program for the ultrasonic sensors.

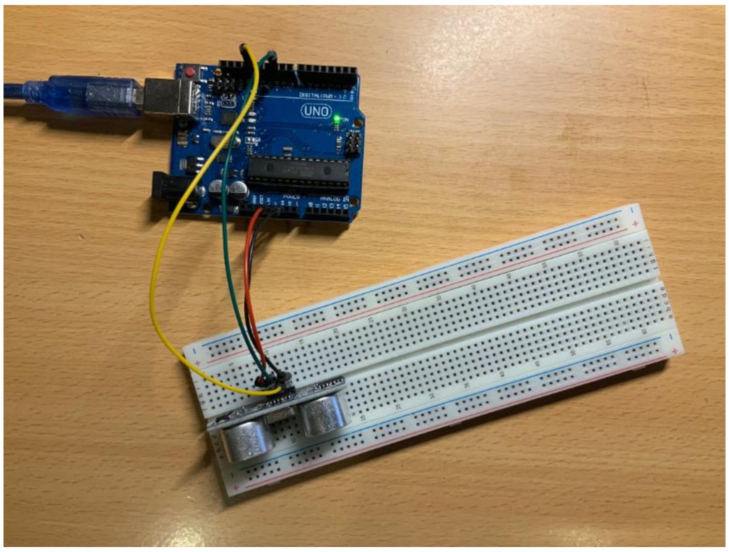


Figure 19 Programming and Debugging of the Ultrasonic Sensor

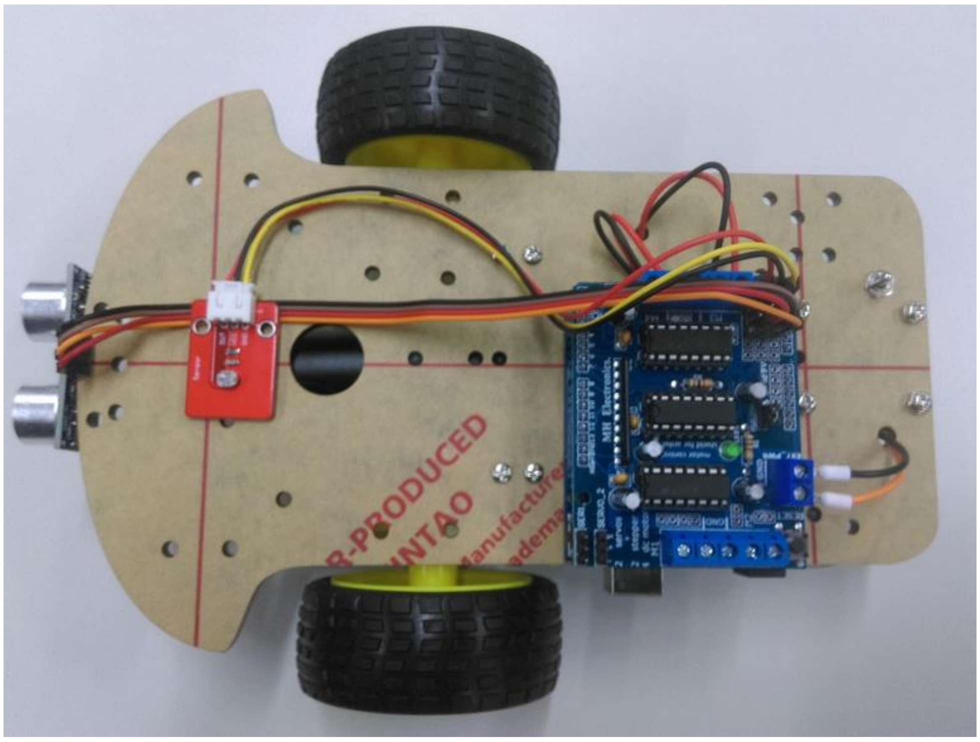
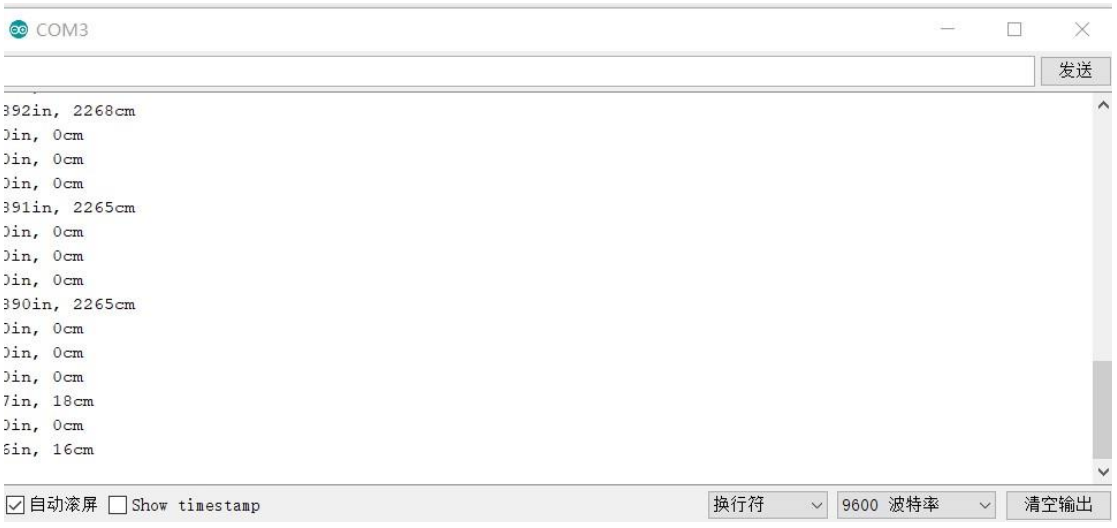


Figure 20 Attach the Ultrasonic Sensor on the motor control board base

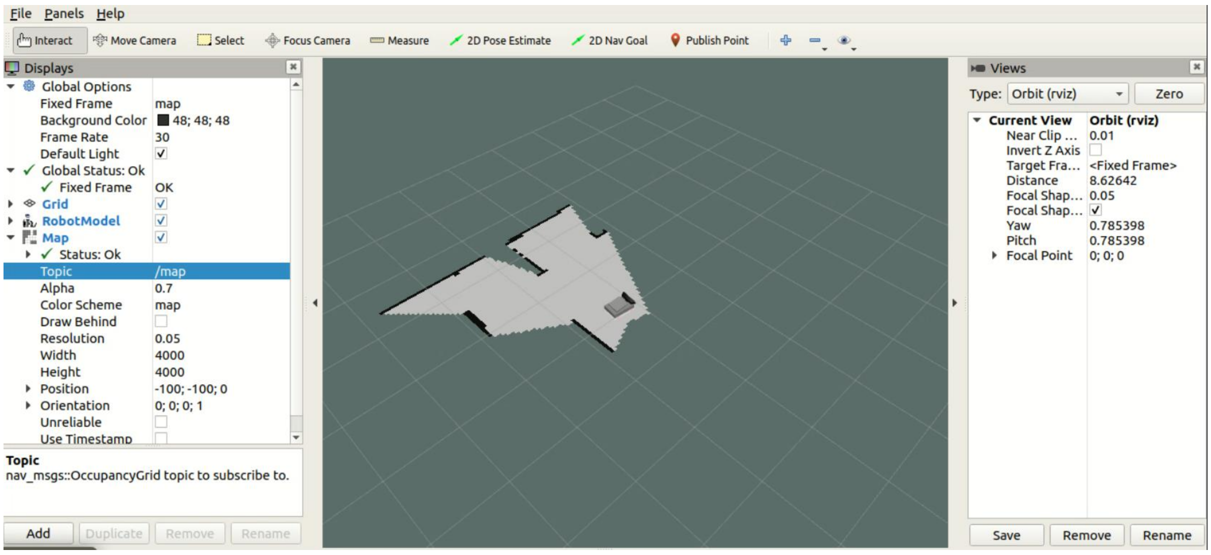


```
COM3
发送
392in, 2268cm
0in, 0cm
0in, 0cm
0in, 0cm
391in, 2265cm
0in, 0cm
0in, 0cm
0in, 0cm
390in, 2265cm
0in, 0cm
0in, 0cm
0in, 0cm
7in, 18cm
0in, 0cm
6in, 16cm
☑自动滚屏 ☐Show timestamp    换行符    9600 波特率    清空输出
```

Figure 21 Program Debugging

ROS:

Catkin is ROS official software packages compilation system, which it combines the Cmake file and Python script to provide the flow of Cmake.



```
Displays
▾ Global Options
    Fixed Frame          map
    Background Color     ■ 48; 48; 48
    Frame Rate           30
    Default Light        ✓
▾ ✓ Global Status: Ok
    ✓ Fixed Frame        OK
▸ Grid                   ✓
▸ RobotModel             ✓
▾ Map                    ✓
  ▸ ✓ Status: Ok
    Topic                /map
    Alpha                0.7
    Color Scheme         map
    Draw Behind          ☐
    Resolution           0.05
    Width                4000
    Height               4000
  ▸ Position             -100; -100; 0
  ▸ Orientation          0; 0; 0; 1
    Unreliable           ☐
    Use Timestamp        ☐
Topic
nav_msgs::OccupancyGrid topic to subscribe to.

Views
Type: Orbit (rviz)
▾ Current View      Orbit (rviz)
    Near Clip ...   0.01
    Invert Z Axis   ☐
    Target Fra...   <Fixed Frame>
    Distance        8.62642
    Focal Shap...   0.05
    Focal Shap...   ✓
    Yaw             0.785398
    Pitch           0.785398
  ▸ Focal Point     0; 0; 0
```

Figure 22 SLAM in ROS

When the robot system is complicated, there will be dozens or even hundreds of nodes, so when starting these nodes, if each node needs to manually enter a command to start, it will be very time-consuming, so ROS provides a way to start in batches Node, the launch description file that sets the node startup parameters. [9] The typical slam startup file starts the following nodes: map server, keyboard remote control, pose filter, base controller, slam_gmapping, rviz visual interface, point cloud to 2D laser data, move_base node, etc. In contrast, only the gmapping node is replaced by the amcl node in the navigation startup file, and the rest is basically the same.

Plugin used: Rqt_rviz, Rqt_plot, Rosbag, Rqt_graph, Rqt_console, Rqt_reconfigure.

SLAM:

After launching the gmapping launch file, the corresponding nodes such as base controller, pose filter, Kinect driver, point cloud conversion laser, visual interface, etc. are started. The robot will immediately estimate its pose and build the increase during walking Quantitative map, this process is called SLAM.

At the beginning of the design, I tried to use the hector_slam algorithm. This algorithm relied on high precision lidar at the beginning of the design. It was soon discovered that because the Kinect's viewing angle was too small and the noise was too much, the resulting map was almost unusable. So, the SLAM algorithm was replaced with gmapping, and the effect was improved significantly.

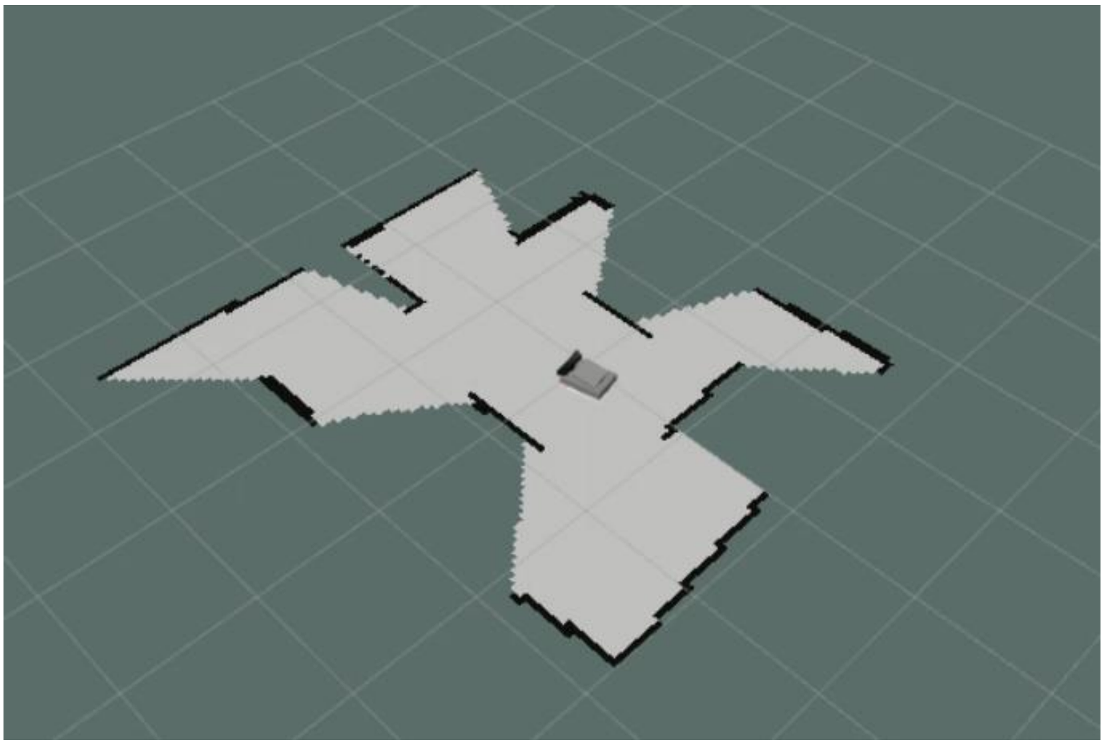


Figure 23 SLAM in ROS

In the figure, the gray area indicates unknown, the white area indicates a known passage area, and the black area is a fixed obstacle detected.

It can be seen that the global map of the robot is basically consistent with the actual experimental environment outline, but there are still defects in the corners or details. The main reason is that the accuracy of the wheel speedometer navigation is not high enough, the visual angle of Kinect is small, and the noise is caused. If you improve the accuracy of the wheel speedometer and use a high-precision, wide-angle lidar, this problem will be effectively solved.

Conclusion and future work:

This project report reflects creating a mobile robot with localization functionality and the analysis and designs the algorithm for trajectory calculation, positioning, path planning and SLAM.

Project design including Arduino Uno board with ultrasonic sensors for obstacle avoidance. Raspberry pi 3B+ with ubuntu 20.04 LTS OS with ROS software implements the framework and user interface. ROS for 3D modelling of the mobile robot, debugging of the ROS robot packages, to be able to achieve SLAM, positioning and mapping functionality.

Nevertheless, this design can be improved with better accuracy and stability of the system for future study associated with UAV, self-driving vehicles and other robotic purposes.

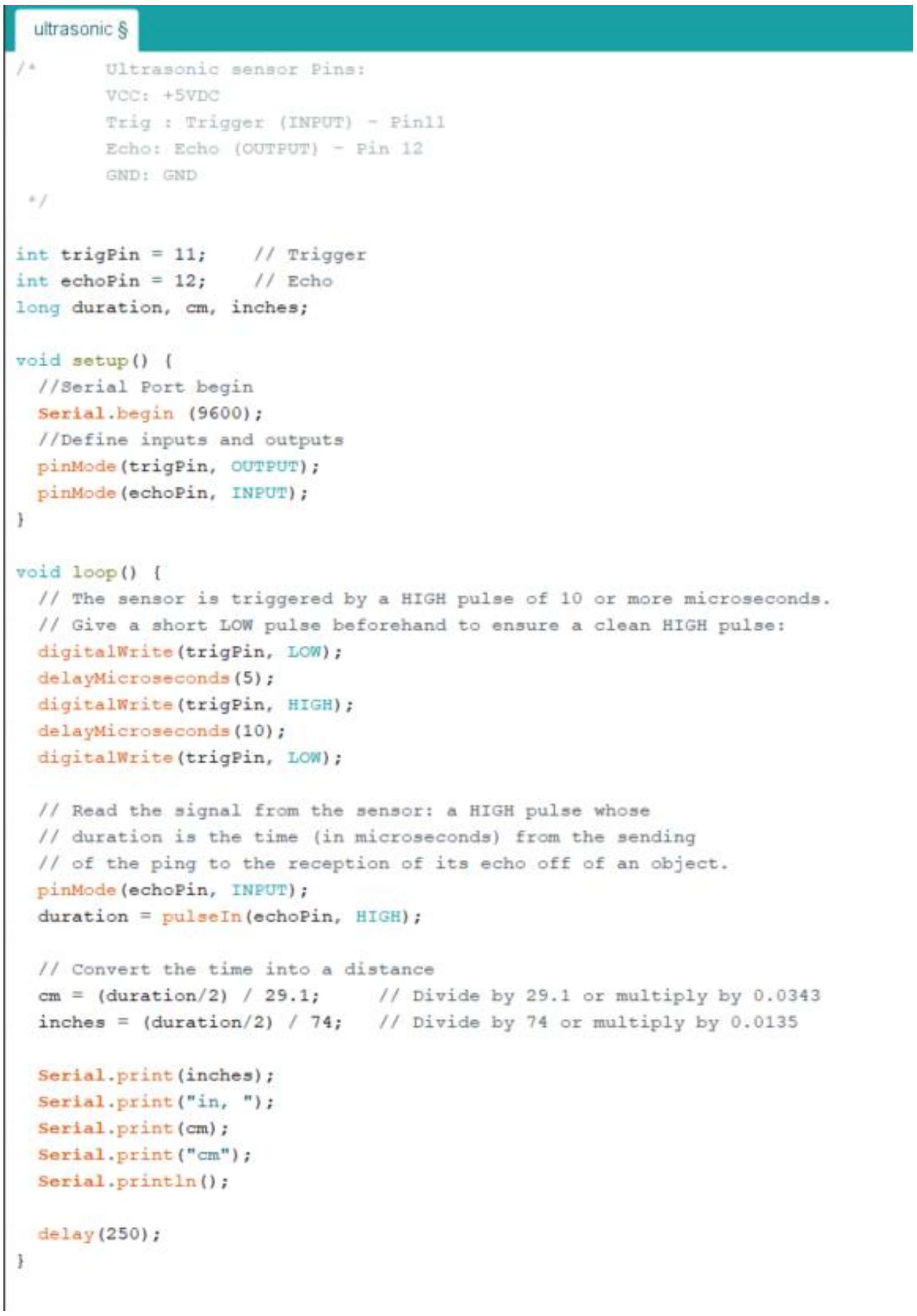
There are possible functionality can be attached to the mobile robot algorithm in the future work, such as voice command to control the movement of the mobile robot, face recognition to communicate with the user, deep learning and machine learning approach in the algorithm and better and reliable 3d reconstruction of the environment, especially useful for building and construction site.

Reference:

1. n.d. *Ultrasonic Ranging Module HC - SR04*. [PDF] Elecfreaks.com, p.1. Available at: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.
2. n.d. Raspberry Pi 3 Model B+ Datasheet. [online] pp.1-5. Available at: <https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf>.
3. Julian Seßner, M. and Systems, I., 2020. *Microsoft Kinect V2 3D-Camera – FAPS – Institute For Factory Automation And Production Systems*. [online] Faps.fau.eu. Available at: <https://www.faps.fau.eu/ausbio/microsoft-kinect-v2-3d-camera/>.
4. Robot operating system (ROS). KOUBAA A. Studies in computational intelligence. 2016
5. (2020). Retrieved 21 June 2020, from https://www.researchgate.net/publication/314187567_3D_modelling_and_simulation_of_a_crawler_robot_in_ROSGazebo
6. Saat, S., Airini, A., Salihin Saealal, M., Wan Norhisyam, A., & Farees Ezwan, M. (2019). Hector SLAM 2D Mapping for Simultaneous Localization and Mapping (SLAM). *Journal Of Engineering And Applied Sciences*, *14*(16), 5610-5615. doi: 10.36478/jeasci.2019.5610.5615
7. Krinkin, K. (2016). TinySLAM improvements for indoor navigation. *2016 IEEE International Conference On Multisensor Fusion And Integration For Intelligent Systems (MFI)*, *6*. doi: 10.1109/MFI.2016.7849536
8. YI, Y., & LIU, D. (2010). Robot Simultaneous Localization and Mapping Based on Path Planning in Dynamic Environments. *ROBOT*, *32*(1), 83-90. doi: 10.3724/sp.j.1218.2010.00083
9. (2020). Retrieved 21 June 2020, from https://www.researchgate.net/publication/314187567_3D_modelling_and_simulation_of_a_crawler_robot_in_ROSGazebo

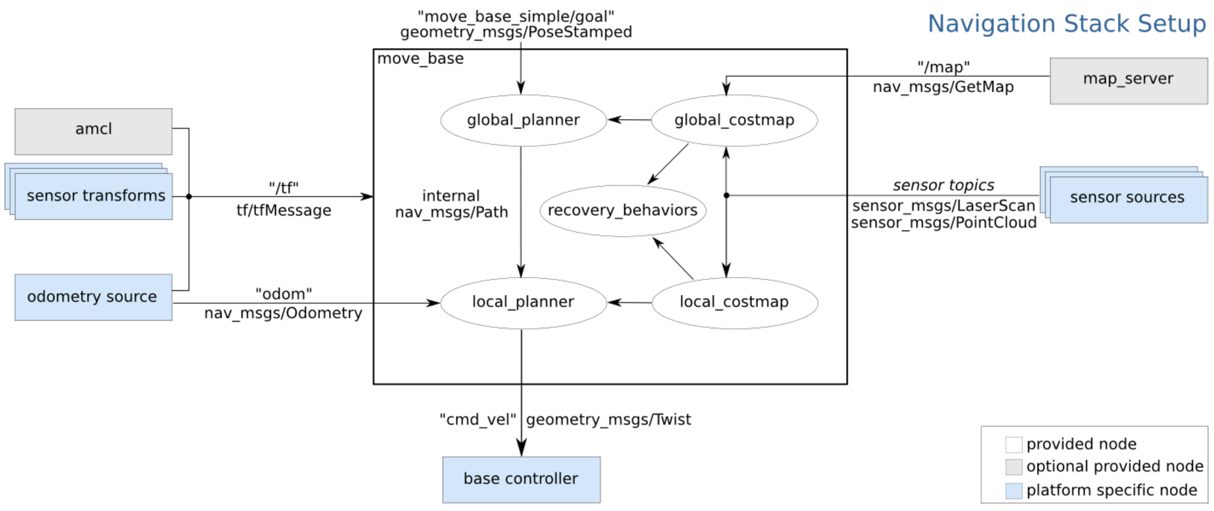
Appendix:

Ultrasonic Sensor Arduino Program:

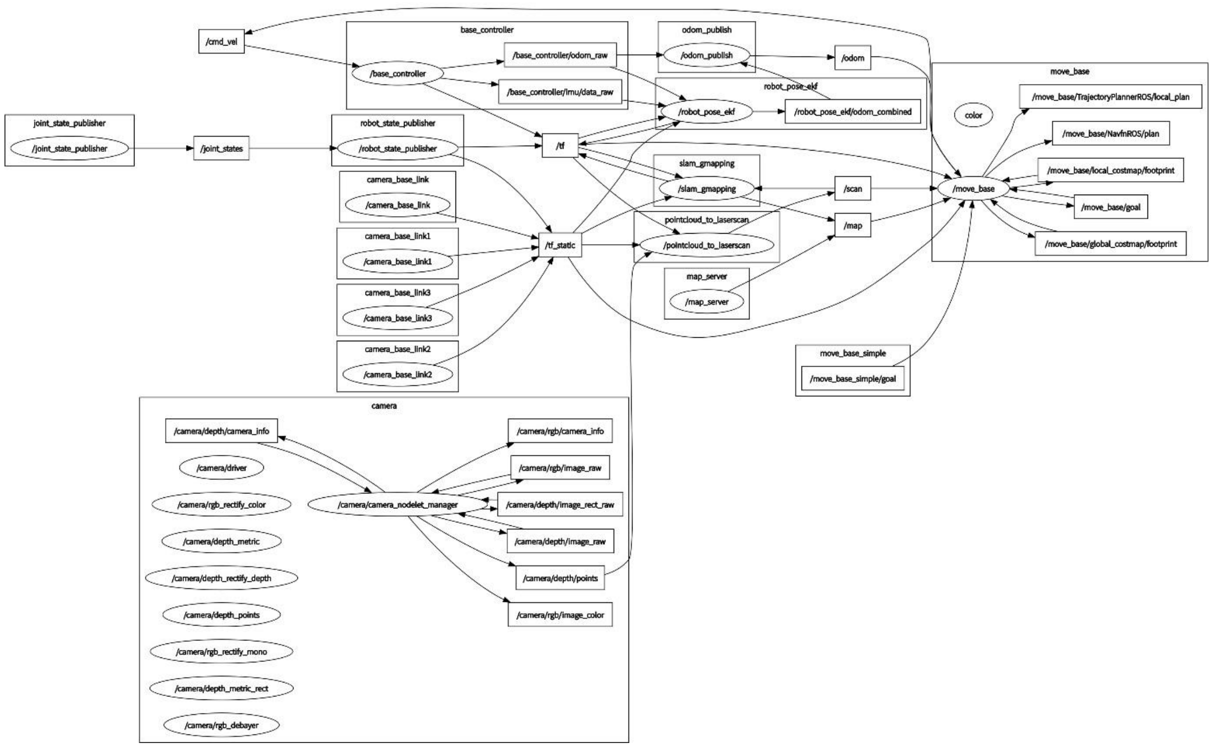


```
/*      Ultrasonic sensor Pins:
        VCC: +5VDC
        Trig : Trigger (INPUT) - Pin11
        Echo: Echo (OUTPUT) - Pin 12
        GND: GND
 */

int trigPin = 11;    // Trigger
int echoPin = 12;    // Echo
long duration, cm, inches;

void setup() {
  //Serial Port begin
  Serial.begin (9600);
  //Define inputs and outputs
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the signal from the sensor: a HIGH pulse whose
  // duration is the time (in microseconds) from the sending
  // of the ping to the reception of its echo off of an object.
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);

  // Convert the time into a distance
  cm = (duration/2) / 29.1;     // Divide by 29.1 or multiply by 0.0343
  inches = (duration/2) / 74;   // Divide by 74 or multiply by 0.0135

  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(250);
}
```

ROS overview TF tree:



ROS Nodes:


/cmd_vel
base_controller
/base_controller
/base_controller/odom_raw
/base_controller/imu/data_raw
odom_publish
/odom_publish
/odom
robot_pose_ekf
/robot_pose_ekf
/robot_pose_ekf/odom_combined
move_base
color
/move_base
/move_base/TrajectoryPlannerROS/local_plan
/move_base/NavfnROS/plan
/move_base/local_costmap/footprint
/move_base/goal
/move_base/global_costmap/footprint
joint_state_publisher
/joint_state_publisher
/joint_states
robot_state_publisher
/robot_state_publisher
/tf
/tf_static
slam_gmapping
/slam_gmapping
/scan
/map
pointcloud_to_laserscan
/pointcloud_to_laserscan
map_server
/map_server
camera_base_link
/camera_base_link
camera_base_link1
/camera_base_link1
camera_base_link3
/camera_base_link3
camera_base_link2
/camera_base_link2
move_base_simple
/move_base_simple/goal
camera
/camera/depth/camera_info
/camera/driver
/camera/rgb_rectify_color
/camera/depth_metric
/camera/depth_rectify_depth
/camera/depth_points
/camera/rgb_rectify_mono
/camera/depth_metric_rect
/camera/rgb_debayer
/camera/camera_nodelet_manager
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/depth/image_rect_raw
/camera/depth/image_raw
/camera/depth/points
/camera/rgb/image_color