IMPLEMENT HECTOR SLAM AND AMCL ALGORITHM FOR UNMANNED GROUND VEHICLE(UGV) IN GPS DENIED AND DYNAMIC ENVIRONMENT

A Project

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Electrical and Computer Engineering

By

Xianzhe Jin

2021

SIGNATURE PAGE

PROJECT: IMPLEMENT HECTOR SLAM AND AMCL ALGORITHM FOR UNMANNED GROUND VEHICLE(UGV) IN GPS DENIED AND DYNAMIC ENVIRONMENT

AUTHOR: Xianzhe Jin

DATE SUBMITTED: Spring 2021

Electrical and Computer Engineering Department

Dr. Zekeriya Aliyazicioglu
Project Committee Chair
Professor of Electrical and Computer Engineering _____

Dr. Tim H. Lin
Professor of Electrical and Computer Engineering _____

Dr. Subodh Bhandari
Professor of Aerospace Engineering _____

ABSTRACT

This project implements the Hector SLAM (Simultaneous Localization and Mapping) AMCL (Adaptive Monte Carlo Localization) algorithm on UGV in GPS denied and dynamic environment. A microcontroller controls the UGV with ultrasonic, IR, step counter sensor to avoid a collision, provide odometer information, and explore the unknown environment. At the same time, the SLAM system works on an onboard computer and maps an unknown indoor environment. Apply AMCL and path planning algorithm RRT*FN-D (Rapidly exploring Random Tree based memory efficient motion planning) to localization and path planning on the unknown environment map created by Hector SLAM before. Robotic Operating System (ROS) is used to integrating all the algorithms. The display interface program Rviz is used to display the testing result.

Finally, the project demonstrates the ability of a UGV to explore an unknown indoor environment without collisions and, at the same time, map the unknown environment. The performance of the developed system was tested in a built environment.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1:

INTRODUCTION

1.1. Overview

The autonomous driving UGV has been popular in many areas such as emergency response, military applications, and commercial applications. One of the major challenges for UGV designers is the use of data from different sensors to control vehicles. Typical sensors used on UGV are lidar, ultrasonic sensors, inertial measurement units (IMU), and stereo vision. The most applied algorithms are simultaneous localization and mapping (SLAM) algorithms to detect the surrounding environment, path planning algorithms to have an optimal path from a start point to the goal point in real-time at both static and dynamic environments, and obstacle avoidance algorithms to avoid a collision from obstacles.

Mobile robots and robot operating systems (ROS) are widely used in R&D phases. Some mobile robots can navigate without guidance, while others need the pre-defined route or some guidance devices. With the help of ROS, researchers and engineers can test more algorithms on mobile robots instead of starting from "reinventing the wheels." In this project, a mobile robot with lidar, ultrasonic sensor, IR, and wheel encoder are employed for Hector-SLAM, and RRT*FND algorithms in GPS denied environment. This project aims to prototype a UGV able to construct a 2D map for the unknown environment without GPS and, using this map, navigate UGV to the goal point using a real-time path planning algorithm in a dynamic environment. This project's motivation is that although the current 2D Lidar-based SLAM algorithm, including its application in the indoor rescue

environment, has achieved much success, the evaluation of SLAM algorithms combined with path planning for indoor rescue has rarely been studied.

This project makes use of open-source software ROS. Using ROS eliminates the need to use multiple micro-computer.

1.2. Research Objectives

- Sensor fusion: To integrate the data from the different sensors and to make the UGV have a comprehensive understanding of the surrounding area.
- SLAM: Map the unknown environment and localization itself without GPS.
- Path Planning: To calculate the best route to where they want to go.
- Control: To control the UGV to follow the planned trajectory, including avoiding obstacles.

1.3. Project Outline

The project is outlined as follows: Chapter 2 introduce all the theory background for the project. It contains explanations about Hector-SLAM, AMCL, RRT*FND algorithm, and the software used for integrating all the algorithm and systems. Chapter 3 takes a deeper look into the design and implementation of the system, including the mobile robot, lidar sensor, ultrasonic sensor, IR sensor, and wheel encoder. Chapter 4 evaluates the system performance with respect to the 2D mapping and obstacle detection and avoidance. This chapter also discusses the limitations of the system. Chapter 5 summarizes the project and suggests some future work.

CHAPTER 2:

THEORY BACKGROUND

2.1. Simultaneous Localization and Mapping (SLAM)

The SLAM problem estimates the robot's location and a map of the environment with various measurements from the onboard sensors over discrete time steps. Statistical techniques, including Kalman filters and particle filters, estimate the posterior probability function for the pose of the robot and the map's parameters.

Many SLAM algorithms are implemented in ROS libraries; three SLAM algorithms are most used in recent research.

• Gmapping: Gmapping is a laser-based SLAM. It is substantially a particle filter. It needs odometry and laser scan data to produce a 2D occupancy grid map.

• Hector SLAM: Hector SLAM is laser-based and based on a scan matching algorithm. It can be used without odometry and on platforms that can roll or pitch. It benefits from the high rate lidar systems and provides 2D pose estimation at the sensors' scan rate. It is accurate enough, although it does not provide explicit loop closing ability. It is sufficiently exactly for many real-world scenarios [1].

• Cartographer: Cartographer is an algorithm developed by Google in 2016. It is a depth camera-based SLAM. It achieves real-time loop closure by using a branch-and-bound approach to compute scan-to submap matches as constraints. It works well with a portable platform and limited computational resources [2].

In this project, the performance of Hector SLAM was evaluated. There are stairs and rugged surfaces in a search and rescue environment, GMapping and Cartographer are not proper for this project because it relies on an odometer and IMU. A search and rescue

environment cause the odometer and IMU inaccurate. So, Hector-SLAM is choose to use in this project.

2.2. AMCL (Adaptive Monte Carlo Localization)

AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive Monte Carlo localization approach, which uses a particle filter to track a robot's pose against a known map [3].

Estimating a dynamic system based on noisy sensor measurements is extremely important in areas of mobile robot navigation. Last decade, many of the state estimation problems are using particle filters. AMCL presents a statistical approach to increasing the efficiency of particle filters by adapting the size of sample sets on the fly. The key point of the adaptive Monte Carlo localization method is to bound the approximation error introduced by the sample-based representation of the particle filter. If the density is high on a small part of the state space, the adaptive particle filter algorithm choose a small number of samples and it determines a large number of samples if the state uncertainty is high. Both the implementation and computation overhead of this approach is small.

In this project, the performance of AMCL was evaluated.

2.3. RRT*FND algorithms for Path Planning

The effective path planning in complex environments with dynamic obstacles is becoming essential. Path planning aims to find a sequence of discrete UGV configurations from the initial to the goal state complying with the system's environment and internal dynamics constraints. Preliminary research on the topic included cell decomposition-based

methods [4], artificial potential fields [5], and visibility graphs [6]. However, these methods have limitations in higher dimensional configuration spaces due to the need for explicit representation of the configuration space's obstacles.

Currently, sampling-based planners lead the path planning for higher dimensional configuration spaces. Sampling-based path planning algorithms offer significant computational savings over complete path-planning because an explicit representation of the environmental constraints (i.e., obstacles) is unnecessary. The sampling-based path planning method uses collision checking to verify a candidate path's feasibility and connect a set of collision-free points to create a graph of the feasible path. Most popular sampling-based path planning algorithm is based on Rapidly-Exploring Random Trees (RRT) [7]. As first generation of the algorithm, RRT returns a solution when one of the collision-free trajectories reaches the goal region. RRT does not use a metric to measure the path optimality between the initial state and the other nodes due to its emphasis on to find feasible solution as quickly as possible. The next generation is RRT* that extends RRT by performing local optimization to improve the results of the global path planning problem. To curb the memory need and speed up the search of neighborhood nodes in asymptotically optimal planning, RRT*FN [8] limits the maximum number of nodes in the tree and removes a node to add a new node in its incremental sampling procedure. In real-world scenarios, information about the environment is often incomplete, fragmented, and only updated iteratively over time, occasionally with false information. The most straightforward approach to tackling the robot path planning problem is to treat all obstacles as static and re-run the path planner for static environments with the updated information. Depending on the planning algorithm, this approach provide a feasible or even

an optimal solution; however, the computation of the whole solution from scratch take a considerable amount of time, which might not be available once the motion is started. Therefore, there is an interest in developing computationally efficient motion planners tailored for problems with unknown and dynamic obstacles. For this problem, RRT*FN-Dynamic (RRT*FND) [9] is introduced.

This project implemented the RRT*FND algorithm for our UGV for search and rescue missions. RRT*FND is introduced for path planning in the presence of dynamic obstacles. Once a dynamic obstacle invalidates a part of the tree in our algorithm, those nodes are removed. However, the nodes on the solution path not colliding with the obstacle are kept intact. Then RRT*FND algorithm tries to reconnect the tree to one of the solution path nodes disconnected from the main tree by the obstacle. Before RRT*FND is introduced, there is algorithm DRRT, which rapidly removes the invalidated nodes and regrows the tree until a new solution is found. Even though our algorithm has similarities and is inspired by the DRRT, it differs in three main areas: First, the RRT*FND algorithm is based on the RRT*FN, which has a limit on the maximum number of nodes. This allows RRT*FND to rapidly remove the invalidated nodes, even in high dimensional and complex problems. Secondly, RRT*FND solves an optimal path planning problem instead of the feasible one solved by DRRT. Thirdly, RRT*FND considers the nodes on the solution path and solve the smaller problem of reconnecting the intact tree to the solution path. The source code for RRT*FND presents at Appendix.

2.4. Robot Operating System (ROS)

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more [10]. ROS community is where ROS users can exchange software and knowledge, including different distributions, repositories are for other institutions to develop and release their software. Language-independent tools and client libraries are released under the terms of the Berkeley Software Distribution (BSD) license, so they are open-source software and can be used for free both commercially and in research. Most of the packages are also licensed under open source licenses. These packages implement the commonly used functionality like hardware drivers, robot models, data types, route planning, environment perception, simultaneous localization and mapping (SLAM), simulation tools, and other algorithms. The main ROS client libraries (C++, Python, and Lisp) are for a Unix-like system. Client libraries are supported in Ubuntu Linux. ROS-Industrial is an open-source project. It applies the capabilities of ROS to manufacturing automation and robotics. It provides interfaces for common industrial manipulators, grippers, sensors, and device networks. It also includes software libraries for automatic 2D/3D sensor calibration, route planning, an application called Scan-N-Plan, developer tools like the Qt Creator ROS Plugin, and a training curriculum specially designed for the needs of manufacturers. ROS releases many versions that are called distribution. Each distribution includes a different set of ROS packages. Each distribution has its name. Indigo Igloo is more friendly to beginners of ROS for its stability and great community support. Catkin, which is a build system, is only supported by Indigo and later. Kinetic Kame is the most used distribution with new capabilities. ROS

is picked for this project because ROS can compile various software languages, so multiple software language script can be used in this project when integrate all the system together.

CHAPTER 3:

DESIGN AND IMPLEMENTATION

In this chapter, the design and implementation of the low-cost UGV prototype with autonomous mapping functionality is presented. The prototype is capable of building 2D map of unknown environment simultaneously, and it can avoid collision. The UGV can localizing itself in created map and moving according to the path planned by path planning algorithm.

3.1. System Architecture

This project's system is a low-cost UGV prototype capable of localization and mapping, and path planning using a lidar sensor. As is illustrated in Figure 1, the architecture of the system consists of four components:

• UGV: The mobile platform is used to carry the lidar scanner to build the map and travel around to test the obstacle detection and avoidance function.

• Jetson Nano: it is an onboard computer to run the ROS.

• Sensors: RPLIDAR, Ultrasonic sensor, Speed Measuring sensor, and Infrared sensor for scanning the environment and navigation.

• Laptop/Desktop: It is used as a screen for onboard computer also as a remote-control station.

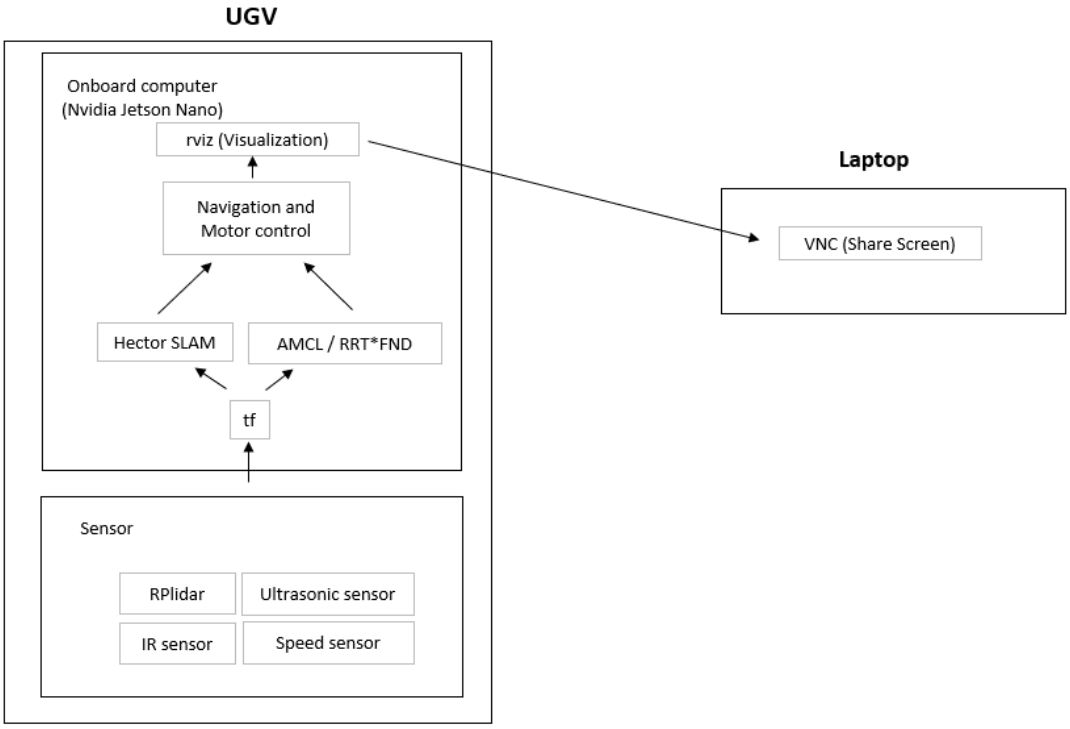

Figure 1: Overall system block diagram

3.2. Hardware

This section describes the hardware employed in this project (see from Figure 2 to Figure 8) and their configurations, including the Maker Focus DIY Robot Car Smart Chassis Kit for mobile platform, a microcontroller (Arduino UNO), a Slamtec RPLIDAR A1M8 sensor, a SongHe Speed Measuring Sensor, L298N Motor Drive Controller, INIU 10000mAh Power Bank, 3.7V 5000mAh Rechargeable Batteries, Gikfun Obstacle avoidance IR Infrared Sensor, Yeeco 4PCS DC Electric Motor 3V-6V Dual Shaft Geared TT Magnetic Gearbox Engine 1:120 Reduction Ratio, and 20 line step encoder disk. Figure 2 shows the prototype UGV. The UGV consists of 4 layers. The UGV dimensions are 30cm for the length, 17cm for the width, and 21cm for the height.

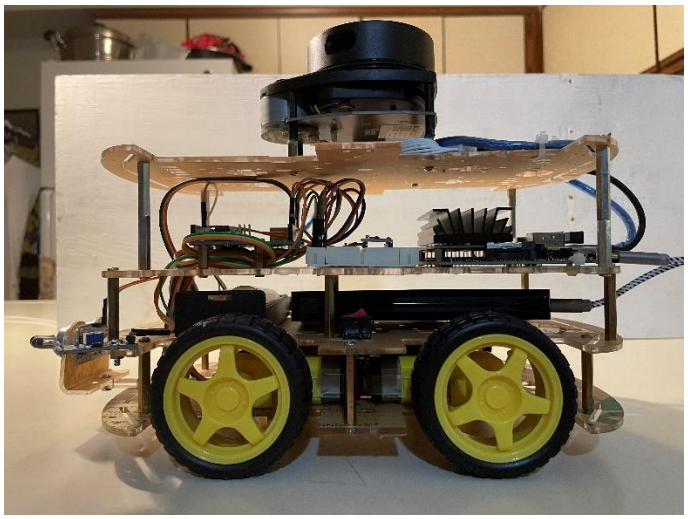

Figure 2: The Prototype UGV

Figure 3 shows the upper side of first layer. This layer has 4 DC motors with two step encoder disks and one L298N motor driver. Figure 4 shows the bottom side of the UGV and step encoder sensor.

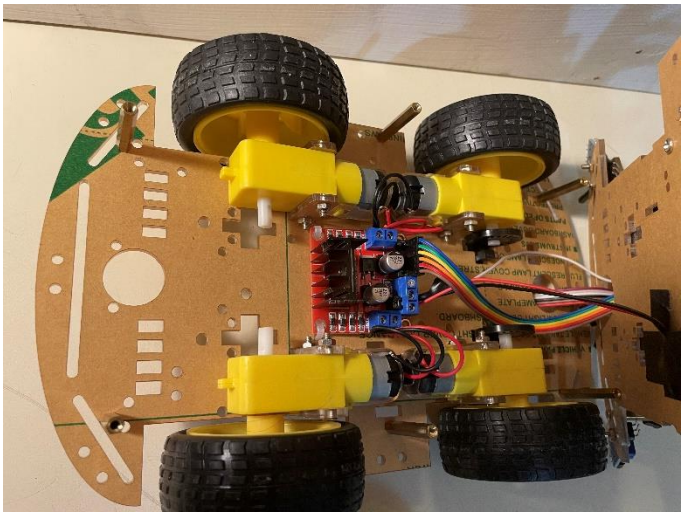

Figure 3: The upper side of first layer

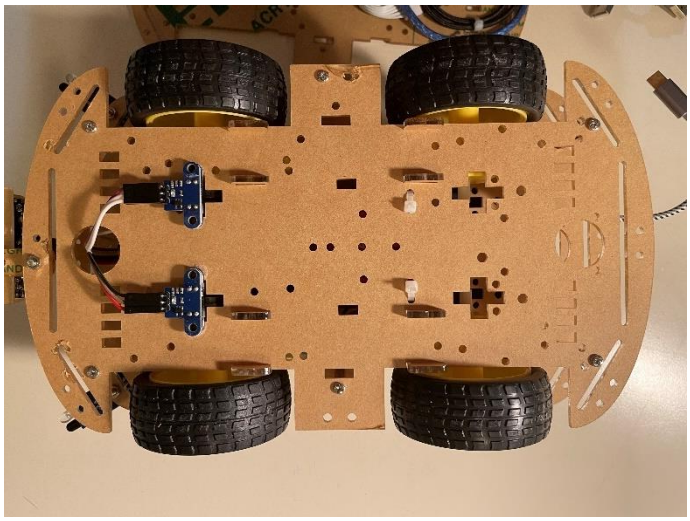

Figure 4: The bottom side of first layer

Figure 5 shows the second layer of the UGV. This layer has two sets of battery. Two 18650 5000mAh 3.7 Li-ion batteries provide power for motor. One 10000mAh Portable Charger provides power for Jetson Nano onboard computer, Arduino UNO, and RPLIDAR. Two Infrared sensor were mounted at front corner.

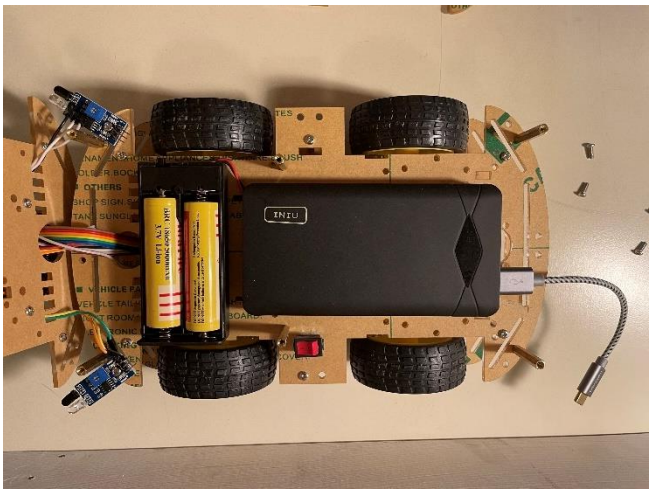

Figure 5: Second layer of the UGV

Figure 6 shows the third layer of the UGV. This layer has onboard computer Jetson Nano, microcontroller Arduino UNO, and bread board for cable connection purposes. The bread board also for mount IMU for future research.

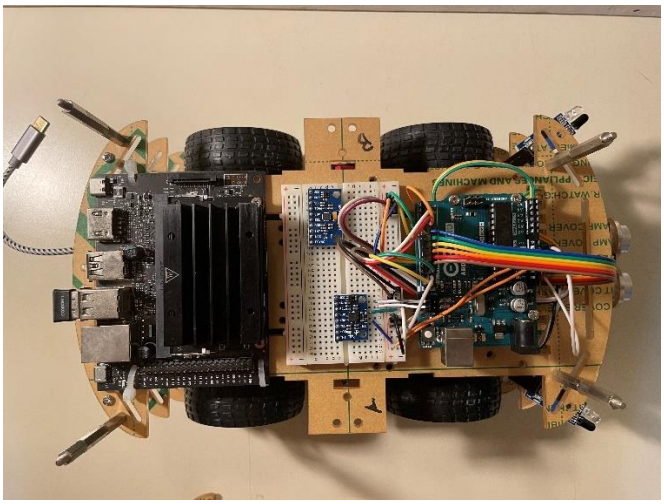

Figure 6: Third layer of the UGV

Figure 7 shows the top layer of the UGV. This layer has the RPLIDAR on the center of the UGV.

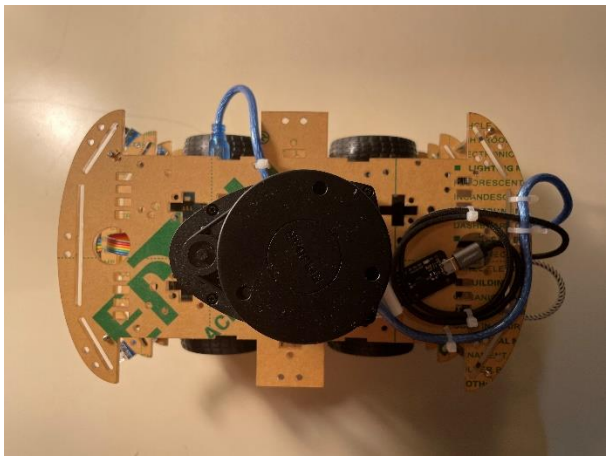

Figure 7: The top layer of the UGV

Figure 8 shows the front of the UGV. The Ultrasonic and IR sensor were mounted front of the UGV.

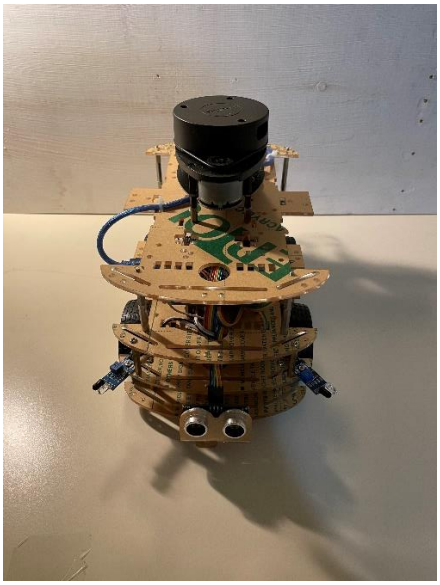

Figure 8: The front of the UGV

As is shown in Figure 9, the Maker Focus DIY Robot Car Smart Chassis Kit is composed of many pieces. Each component is described below.

Figure 9: Robot car chassis kit

The kit comes with four rubble wheels, plastic chassis, 4PCS 3-6VDC Electric Motor Dual Shaft Geared TT Magnetic Gearbox Engine (1:120 Reduction Ratio), and two 20-line

wheel encoders disks. In this project, the two sets of chassis are used to make a multiple-layer mobile robot that carries all necessary components, as shown in Figure 2.

Figure 10 shows a typical Arduino board in the market. Arduino has some advantages, such as a large user community, free and broad ranges of libraries of codes, relatively low-cost components, a wide array of sensors, and many projects and resources that exist for free (online). In addition, Arduino boards can work as ROS nodes directly using the rosserial arduino package. In this project, Arduino collect sensor data for obstacle avoidance, control motor for movement. In addition, Arduino communicate with ROS that runs on the onboard computer to provide odometer information and receive movement commands.

Figure 10: Arduino UNO

As is shown in Figure 11, the NVIDIA Jetson Nano 2GB Developer Kit is ideal for teaching, learning, and developing AI and robotics. With an active developer community and ready-to-build open-source projects, all the resources can be found to get start. It delivers incredible AI performance at a low price and makes the world of AI and robotics

accessible to everyone with the exact same software and tools used to create breakthrough AI products across all industries. In this project, Jetson Nano run Hector_SLAM algorithm, AMCL algorithm, path planning algorithm, and ROS [11].

Figure 11: Jetson Nano 2GB

An optic interrupter consists of a source of light, usually an infrared LED, and a phototransistor sensor. The light source is mounted facing the sensor with a gap between them. In operation, the LED is illuminated, and it shines onto the phototransistor, which detects its light and allows current to pass from the collector to the emitter.

Essentially a phototransistor is like a regular bipolar transistor, except instead of reacting to current applied to the base, it reacts to photons of light. If a solid non-transparent object is placed in the slot between the LED and phototransistor, it interrupt the light beam, causing the phototransistor to stop passing the current. In our application, the optic interrupter be positioned with the rotating encoder wheel in the gap between the LED and transistor. As the wheel spins, the slots in the wheel allow pulses of light to reach the phototransistor, causing it to switch on and off in time with the wheel rotation. Each pulse represent a slot in the encoder wheel, so if your encoder wheel has 20 equally spaced slots (a pretty standard value), then each pulse indicates that the wheel has turned 18 degrees

(360 degrees divided by 20) [12]. As is shown in Figure 12, the SongHe IR infrared slotted optical optocoupler module photo interrupter sensor and wheel encoder disk show in Figure 13 provide the odometer information for the mobile robot in this project.

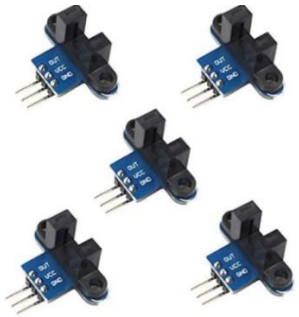

Figure 12: Wheel encoder sensor

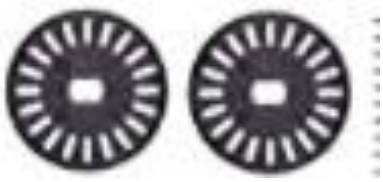

Figure 13: Wheel encoder disk

As is show in Figure 14, the HC-SR04 Ultrasonic sensor is a versatile device that has become a staple in robotics projects. This inexpensive component measures the distance between itself and the nearest solid object using pulses of ultrasonic sound. It has reasonable accuracy “out of the box” and can be made even more accurate with one additional component. In this project, the ultrasonic sensor is used to provide front obstacle information so that the UGV can avoid the front obstacles [13].

Figure 14: HC-SR04 Ultrasonic sensor

An active infrared sensor is used in this project that shown in Figure 15. Active IR sensor uses an emitter and receiver facing the same direction. The two sit very close to each other so the receiver can detect an object's reflection when it enters an area. A fixed reflector bounces the signal back. This method replicates separate emitter and receiver units but without the need to install a remote electrical component. The sensing distance can be modified using the adjustable meter on the sensor . In this project, the IR sensor is used to provide front-side obstacle information [14].

Figure 15: IR sensor

Figure 16 shows 3-6VDC Electric Motor Dual Shaft Geared TT Magnetic Gearbox Engine (1:120 Reduction Ratio). The L298N H-bridge motor drive controller shown in

Figure 17 is the most common and inexpensive motor driver for DC motors. The motor speed and direction are easy to be controlled by PWM signal.

Figure 16: DC motor

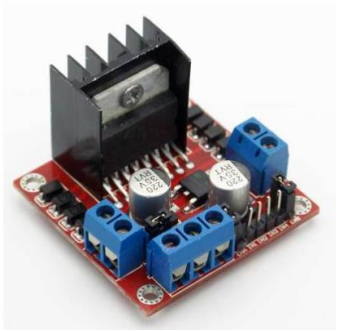

Figure 17: L298N Motor Driver

RPLIDAR that shown in Figure 18 is a low-cost LIDAR sensor suitable for indoor robotic SLAM applications. It provides a 360-degree scan field, 5.5hz/10hz rotating frequency with guaranteed 8-meter ranger distance. RPLIDAR is the ideal sensor in cost-sensitive areas like robots consumer and hardware hobbyist. Besides, ROS (Robot Operating System) is a popular software library for robotics programming. It is free, open-source, and used by robotics researchers and companies around the world [15]. In this project, Lidar scan surroundings and provide information to the SLAM algorithm for map unknown environments.

Figure 18: RPLIDAR

3.3. ROS Software

Mapping, localization, and Path Planning packages are used in this project. Each package consists of several nodes, topics (communication channels), and services. The implementation of each node is described below.

As shown in Figure 19, the mapping package includes four major nodes. Rplidar node for scanning information. This node use RPLIDAR to scan surrounding and publish the data name as /scan, sample output of Lidar scanner shows in Figure 20. Hector_mapping node subscribe to the necessary node for integrating data and publish map and UGV position, Hector_jrajectory_server node for publishing UGV trajectory information. Detail of the launch file and file configuration show in the appendix E.

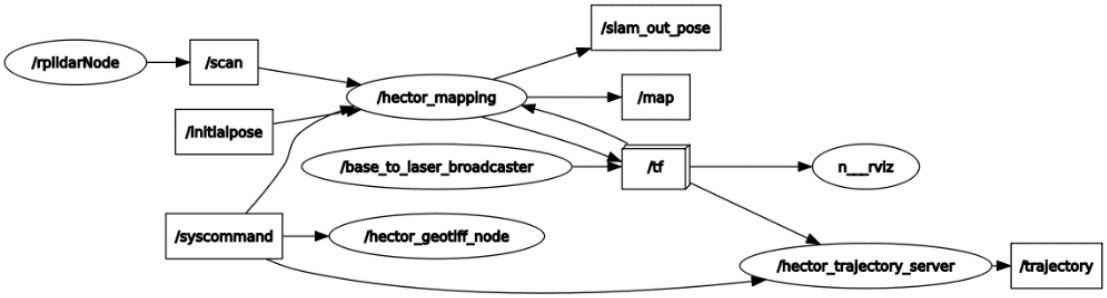

Figure 19: Hector SLAM Package Node Instruction

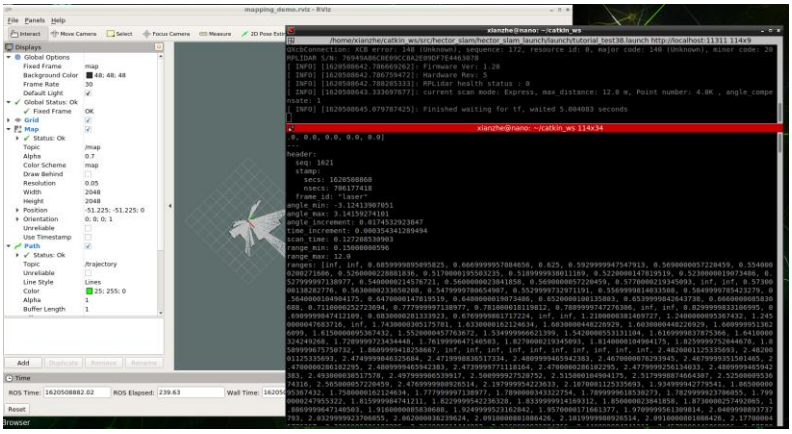

Figure 20: Sample Output of Lidar Scanner

As shown in Figure 21, the localization package includes four major nodes. Rplidar node for scanning information, map_server node provides the created map, and the serial node lets the micro-controller communicate with ROS. AMCL node subscribe to other nodes and publish particle cloud information, send transform information node for integrating data from other node and publish map and UGV position. The detail of the launch file and configuration shown in the appendix F.

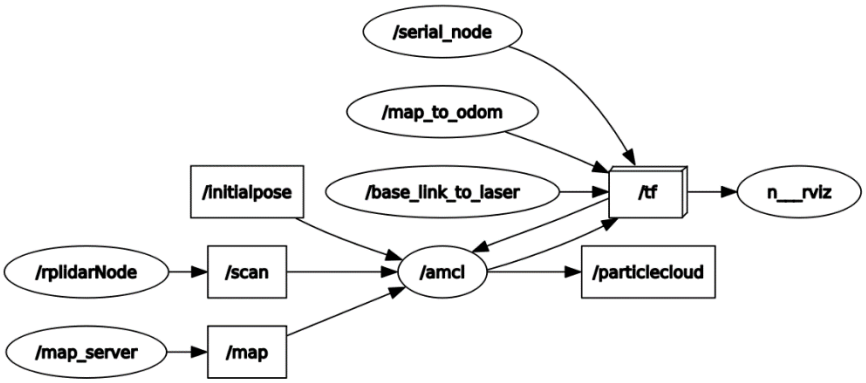

Figure 21: AMCL Package Node Instruction

After starting the Rviz visualization tool, the trajectory and the 2D map are updated and shown on the screen simultaneously (see Figure 22).

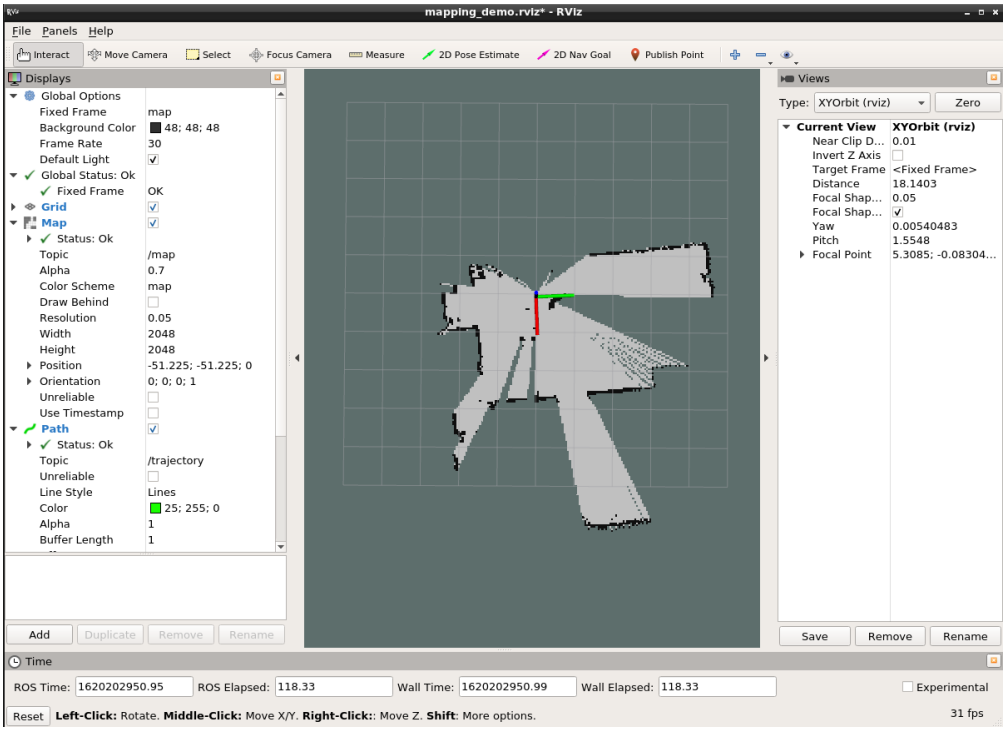

Figure 22: Rviz Interface

Virtual Network Computing, or VNC, is a sharing system that allows user to remotely control one computer from another. A VNC server is installed on the system under test

(SUT) and works with Eggplant Functional to allow control of this second machine. In this project. Desktop is used to access onboard computer and control onboard computer remotely.

3.4. Obstacle Detection and Avoidance Algorithm

The Ultrasonic and IR sensors are implemented to achieve autonomous movement in an unknown environment and obstacle detection and avoidance. Both Certain and random movement commands apply to the UGV when considering numerous conditions for the obstacles. In this project, an ultrasonic sensor is used to detect front obstacles and determine how far the obstacles from the UGV. IR sensors put on the front side to detect side obstacles which in close range. The scenario is explained below.

1. If the front ultrasonic sensor does not detect obstacles within 30 cm and both front left and right IR sensor shows no obstacles, UGV execute a forward command.
2. Suppose the front ultrasonic sensor does not detect obstacles within 30 cm and front left, but the front right IR sensor detects the obstacles. In that case, UGV stop, then execute a certain distance of backward command, then turning a certain angle to the left.
3. Suppose the front ultrasonic sensor does not detect obstacles within 30 cm and front right, but the front left IR sensor detects the obstacles. In that case, UGV stop, then execute a certain distance of backward command, then turning a certain angle to the right.
4. Suppose the front ultrasonic sensor detects obstacles within 30 cm and both front right and left detect the obstacles. In that case, UGV stop, then execute a certain

distance of backward command, then turning a certain angle to the right or left by random.

5. Suppose the front ultrasonic sensor detects obstacles within 30 cm, and both the front right and left do not detect the obstacles. In that case, UGV stop, then execute a certain distance of backward command, then turning a certain angle to the right or left by random.

The turning angle and backward distance can be modified to reduce the explore time. The code please refer to the appendix C.

CHAPTER 4:

EVALUATION

In this chapter, the test scenarios that are set up for evaluating the functionalities of the prototype, including obstacle detection and avoidance, localization, and 2D mapping, as well as the test results, are described. The limitations of the prototype are discussed as well.

4.1. Requirements of UGV

In this project, The UGV can explore the unknown environment and avoid obstacles autonomously. At the same time, UGV can localize and build a 2D map using data from the lidar sensor. After the unknown environment's map was created, the UGV can localize itself in the known map and path planning and autonomously drive to the target.

Requirement	**Test Result**
The UGV can autonomously explore unknown environment and obstacle avoid.	PASSED
ROS can output a usable 2D map of the test site with the lidar data.	PASSED
The UGV can localize itself in created map	PASSED
The UGV can path planning and follow the planning path in real time.	In Progress

Table 1: Requirements for the UGV

4.2. Explore Unknown Environment, 2D Mapping, and Trajectory

First, the UGV was tested in a simple environment. The UGV was able to detect walls then avoid a collision. Also created a useable 2D map and the trajectory of the UGV in real-time. Figure 23 shows the Test 1 setting. And Figure 24 shows the created map on the

Rviz. On the Rviz, the green trajectory line, the UGV's position, and orientation are shown in the Figure 24 below.

Figure 23: Test 1

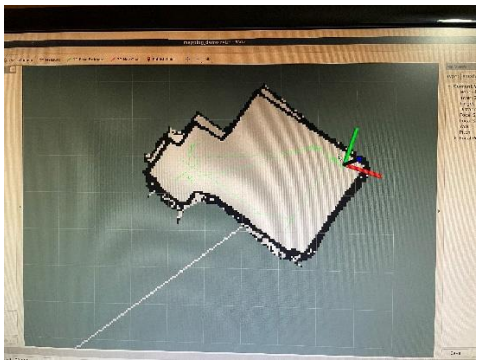

Figure 24: Test 1 Map

Second, the UGV was tested in a little complicate environment. The UGV was able to detect walls then avoid a collision. Also created a useable 2D map and the trajectory of the UGV in real-time. Figure 25 shows the Test 2 setting that the room was added an obstacle in the middle. And Figure 26 shows the created map on the Rviz. On the Rviz, the green trajectory line, the UGV's position, and orientation are shown in the Figure 26 below.

Figure 25: Test 2

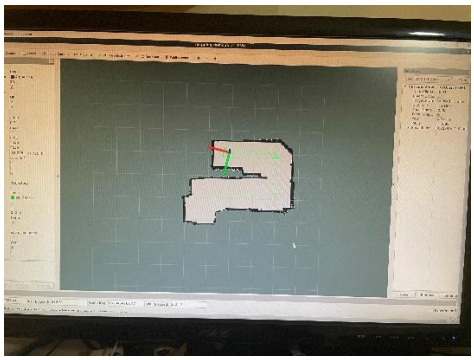

Figure 26: Test 2 Map

Third, the UGV was tested in a more complicate environment. The UGV was able to detect walls then avoid a collision. Also created a useable 2D map and the trajectory of the UGV in real-time. Figure 27 shows the Test 3 setting that the room was added more convex obstacles in order to make the environment complicate. And Figure 28 shows the created map on the Rviz. On the Rviz, the green trajectory line, the UGV's position, and orientation are shown in the Figure 28 below.

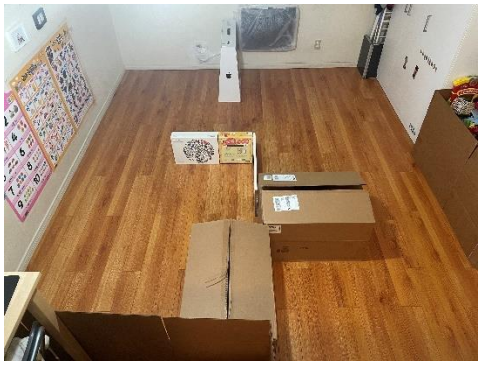

Figure 27: Test 3

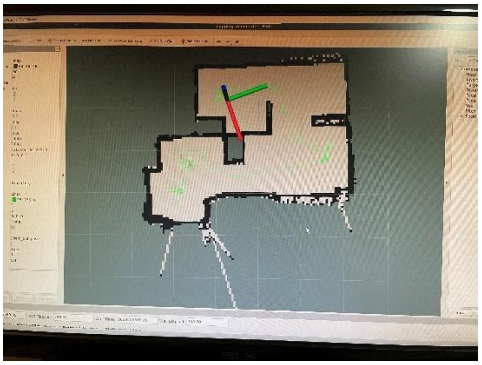

Figure 28: Test 3 Map

4.3. Localization in a Known Map

In this evaluation, the AMCL algorithm package was tested on the second map that created before. As Figure 29 shows, before the UGV moving, estimating the UGV position is spreading on the map. After moving the UGV around on the map, the estimated position of the UGV converse to a particular area, as Figure 30 shows below. So we can confirm that the AMCL algorithm is working correctly and send us a usable estimation position of the UGV on the map.

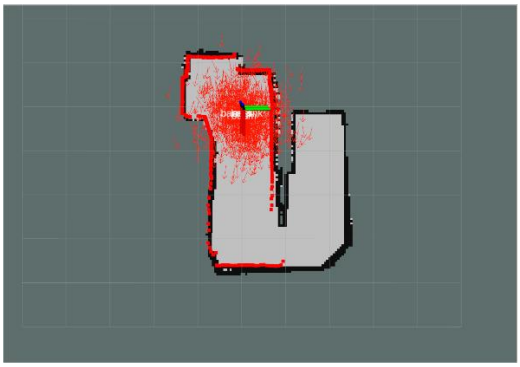

Figure 29: Estimation Position at begin

Figure 30: Estimation Position after Moving

4.4. Limitations

Some limitations of the system are founded during the evaluation:

1. The explore time for the unknown environment. When the UGV met some narrow space or corner, the UGV get a hard time going through it or unable to get out from the corner. Because the ultrasonic and IR sensors were fixed on the UGV at certain angles, that cause the UGV cannot check the obstacles for all directions, then make a decision.
2. The IR sensor be affected by sunlight or other light sources, including infrared light.

3. The ultrasonic sensor has limitations in some environments. For instance, when the front obstacle surface does not have 90 degrees with an ultrasonic search direction not able to detect obstacle at some time.
4. The error for localization is a little high due to the different performance of each motor. For instance, the UGV drift to another side if it executes a forward command.

CHAPTER 5:

CONCLUSIONS

In this project, a low-cost prototype of a UGV capable of building a 2D map of its surroundings with a lidar sensor is presented. The prototype consists of a mobile platform, RPLidar A1, an ultrasonic sensor, two IR sensors, an Arduino UNO board, a Jetson Nano onboard computer, Hector SLAM, and an AMCL package in ROS. The UGV to carry the lidar scanner. Lidar scan points are sent to a Jetson Nano running ROS, and the Hector mapping node in ROS converts the lidar data to a 2D map and realizes the localization functionality. Hector trajectory server node saves the trajectory of the UGV. The UGV use the autonomous navigation provided by Arduino autonomous navigation algorithm using ultrasonic and IR sensor. After the unknown environment's map is created, it is saved using the map_server node. It is then using the AMCL package in ROS, localization the UGV on the map when it moves around. This step is to prepare work for the UGV autonomous drive itself from one point to another point using path planning algorithm.

In conclusion, obstacle detection and avoidance are realized by using a local Arduino board. Localization and 2D mapping functionality are verified in the tests conducted. Usable 2D maps of the unknown environment can be built; thus, the requirement of building a prototype capable of localization and 2D mapping simultaneously and avoiding collision is achieved.

CHAPTER 6:

FUTURE WORK

In the future, Some more work to optimize this project need to be finished. First, for a more accurate odometer date, the algorithm for the control motor needs to be modified. One option is to change the motor control algorithm from PWM control mode to a speed encoder sensor to control the motor. Another option upgrades motor to high torque with speed encoder to move UGV more smoothly and accurately. Also, an IMU sensor can be tested to increase the accuracy of the odometer information. Second, the RRT*FND path planning algorithm and create a UGV move node in ROS need to be accomplished for point-to-point movement with a dynamic environment with a known map.

REFERENCES

[1] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on, pages 155–160. IEEE, 2011.

[2] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In Robotics and Automation (ICRA), 2016 IEEE International Conference on, pages 1271–1278. IEEE, 2016

[3] AMCL-ROS Wiki. amcl - ROS Wiki. http://wiki.ros.org/

[4] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” The International Journal of Robotics Research, vol. 5, no. 1, pp. 90–98, 1986.

[5] T. Lozano-Perez and M. A. Wesley, “An algorithm for planning collision-´ free paths among polyhedral obstacles,” Communications of the ACM, vol. 22, no. 10, pp. 560–570, 1979.

[6] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566–580, 1996.

[7] S. M. LaValle and J. J. Kuffner, “Rapidly-exploring random trees: Progress and prospects,” 2000.

[8] O. Adiyatov and H. A. Varol, “Rapidly-exploring random tree-based memory efficient motion planning,” in Proc. of IEEE International Conference on Mechatronics and Automation (ICMA), 2013, pp. 354– 359.

[9] Olzhas Adiyatov and Huseyin Atakan Varol “A Novel RRT*-Based Algorithm for Motion Planning in Dynamic Environments” Proceedings of 2017 IEEE international Conference on Mechatronics and Automation August 6-9, Takamatsu, Japan.

[10] Documentation – ROS Wiki http://wiki.ros.org/

[11] Jetson Nano Developer Kits. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/education-projects/

[12] DroneBot Workshop. Build a Robot Car with Speed Sensors using Arduino Interrupts. https://dronebotworkshop.com/robot-car-with-speed-sensors/

[13] Using the HC-SR04 Ultrasonic Distance Sensor with Arduino https://forum.dronebotworkshop.com/2017/using-the-hc-sr04-ultrasonic-distance-sensor-with-arduino-everything-you-need-to-know/

[14] Understanding Active & Passive Infrared Sensors (PIR) and .... https://www.arrow.com/en/research-and-events/articles/understanding-active-and-passive-infrared-sensors

[15] RPLIDAR Tutorial Review - DFRobot. https://www.dfrobot.com/blog-1463.html

[16] Robotic Operating System Docs. Retrieved from http://docs.ros.org/

APPENDIX A

RRT* FND SOURCE CODE (SHARED BY AUTHOR OF ALGORITHM)

```
import random
import math
import sys
import pygame
import timeit, time
import numpy as np
show_animation = True
XDIM = 800
YDIM = 600
windowSize = [XDIM, YDIM]
pygame.init()
fpsClock = pygame.time.Clock()
screen = pygame.display.set_mode(windowSize)
pygame.display.set_caption('Performing RRT')
class RRT():
    """
    Class for RRT Planning
    """
    def __init__(self, start, goal, obstacleList,
                 randArea, expandDis=15.0, goalSampleRate=10, maxIter=1500):
        self.start = Node(start[0], start[1])
```

```
        self.end = Node(goal[0], goal[1])
        self.Xrand = randArea[0]
        self.Yrand = randArea[1]
        self.expandDis = expandDis
        self.goalSampleRate = goalSampleRate
        self.maxIter = maxIter
        self.obstacleList = obstacleList
    def Planning(self, animation=True):
        """
        Pathplanning
        animation: flag for animation on or off
        """
        self.nodeList = {0:self.start}
        i = 0
        while True:
            print(i)
            rnd = self.get_random_point()
            nind = self.GetNearestListIndex(rnd) # get nearest node index to random point
            newNode = self.steer(rnd, nind) # generate new node from that nearest node in direction of random point
            if self.__CollisionCheck(newNode, self.obstacleList): # if it does not collide
                nearinds = self.find_near_nodes(newNode, 5) # find nearest nodes to newNode
```

```
            newNode = self.choose_parent(newNode, nearinds) # from that nearest nodes find the best parent to newNode
            self.nodeList[i+100] = newNode # add newNode to nodeList
            self.rewire(i+100, newNode, nearinds) # make newNode a parent of another node if necessary
            self.nodeList[newNode.parent].children.add(i+100)
            if len(self.nodeList) > self.maxIter:
                leaves = [ key for key, node in self.nodeList.items() if len(node.children) == 0 and len(self.nodeList[node.parent].children) > 1 ]
                if len(leaves) > 1:
                    ind = leaves[random.randint(0, len(leaves)-1)]
                    self.nodeList[self.nodeList[ind].parent].children.discard(ind)
                    self.nodeList.pop(ind)
                else:
                    leaves = [ key for key, node in self.nodeList.items() if len(node.children) == 0 ]
                    ind = leaves[random.randint(0, len(leaves)-1)]
                    self.nodeList[self.nodeList[ind].parent].children.discard(ind)
                    self.nodeList.pop(ind)
        i+=1
        if animation and i%25 == 0:
            self.DrawGraph(rnd)
        for e in pygame.event.get():
```

```
            if e.type == pygame.MOUSEBUTTONDOWN:
                if e.button == 1:
                    self.obstacleList.append((e.pos[0],e.pos[1],30,30))
                    self.path_validation()
                elif e.button == 3:
                    self.end.x = e.pos[0]
                    self.end.y = e.pos[1]
                    self.path_validation()
    def path_validation(self):
        lastIndex = self.get_best_last_index()
        if lastIndex is not None:
            while self.nodeList[lastIndex].parent is not None:
                nodeInd = lastIndex
                lastIndex = self.nodeList[lastIndex].parent
                dx = self.nodeList[nodeInd].x - self.nodeList[lastIndex].x
                dy = self.nodeList[nodeInd].y - self.nodeList[lastIndex].y
                d = math.sqrt(dx ** 2 + dy ** 2)
                theta = math.atan2(dy, dx)
                if not self.check_collision_extend(self.nodeList[lastIndex].x,
self.nodeList[lastIndex].y, theta, d):
                    self.nodeList[lastIndex].children.discard(nodeInd)
                    self.remove_branch(nodeInd)
    def remove_branch(self, nodeInd):
```

```
    for ix in self.nodeList[nodeInd].children:
        self.remove_branch(ix)
    self.nodeList.pop(nodeInd)
def choose_parent(self, newNode, nearinds):
    if len(nearinds) == 0:
        return newNode
    dlist = []
    for i in nearinds:
        dx = newNode.x - self.nodeList[i].x
        dy = newNode.y - self.nodeList[i].y
        d = math.sqrt(dx ** 2 + dy ** 2)
        theta = math.atan2(dy, dx)
        if self.check_collision_extend(self.nodeList[i].x, self.nodeList[i].y, theta, d):
            dlist.append(self.nodeList[i].cost + d)
        else:
            dlist.append(float("inf"))
    mincost = min(dlist)
    minind = nearinds[dlist.index(mincost)]
    if mincost == float("inf"):
        print("mincost is inf")
        return newNode
    newNode.cost = mincost
    newNode.parent = minind
```

```
        return newNode
    def steer(self, rnd, nind):
        # expand tree
        nearestNode = self.nodeList[nind]
        theta = math.atan2(rnd[1] - nearestNode.y, rnd[0] - nearestNode.x)
        newNode = Node(nearestNode.x, nearestNode.y)
        newNode.x += self.expandDis * math.cos(theta)
        newNode.y += self.expandDis * math.sin(theta)
        newNode.cost = nearestNode.cost + self.expandDis
        newNode.parent = nind
        return newNode
    def get_random_point(self):
        if random.randint(0, 100) > self.goalSampleRate:
            rnd = [random.uniform(0, self.Xrand), random.uniform(0, self.Yrand)]
        else:  # goal point sampling
            rnd = [self.end.x, self.end.y]
        return rnd
    def get_best_last_index(self):
        disglist = [(key, self.calc_dist_to_goal(node.x, node.y)) for key, node in
self.nodeList.items()]
        goalinds = [key for key, distance in disglist if distance <= self.expandDis]
        if len(goalinds) == 0:
            return None
```

```
        mincost = min([self.nodeList[key].cost for key in goalinds])
        for i in goalinds:
            if self.nodeList[i].cost == mincost:
                return i
        return None
    def gen_final_course(self, goalind):
        path = [[self.end.x, self.end.y]]
        while self.nodeList[goalind].parent is not None:
            node = self.nodeList[goalind]
            path.append([node.x, node.y])
            goalind = node.parent
        path.append([self.start.x, self.start.y])
        return path
    def calc_dist_to_goal(self, x, y):
        return np.linalg.norm([x - self.end.x, y - self.end.y])
    def find_near_nodes(self, newNode, value):
        r = self.expandDis * value
        dlist = np.subtract( np.array([ (node.x, node.y) for node in self.nodeList.values() ]),
(newNode.x,newNode.y))**2
        dlist = np.sum(dlist, axis=1)
        nearinds = np.where(dlist <= r ** 2)
        nearinds = np.array(list(self.nodeList.keys()))[nearinds]
```

```
        return nearinds

    def rewire(self, newNodeInd, newNode, nearinds):
        nnode = len(self.nodeList)
        for i in nearinds:
            nearNode = self.nodeList[i]
            dx = newNode.x - nearNode.x
            dy = newNode.y - nearNode.y
            d = math.sqrt(dx ** 2 + dy ** 2)
            scost = newNode.cost + d
            if nearNode.cost > scost:
                theta = math.atan2(dy, dx)
                if self.check_collision_extend(nearNode.x, nearNode.y, theta, d):
                    self.nodeList[nearNode.parent].children.discard(i)
                    nearNode.parent = newNodeInd
                    nearNode.cost = scost
                    newNode.children.add(i)
    def check_collision_extend(self, nix, niy, theta, d):
        tmpNode = Node(nix,niy)
        for i in range(int(d/5)):
            tmpNode.x += 5 * math.cos(theta)
            tmpNode.y += 5 * math.sin(theta)
            if not self.__CollisionCheck(tmpNode, self.obstacleList):
                return False
```

```
        return True
    def DrawGraph(self, rnd=None):
        u"""
        Draw Graph
        """
        screen.fill((255, 255, 255))
        for node in self.nodeList.values():
            if node.parent is not None:
pygame.draw.line(screen,(0,255,0),[self.nodeList[node.parent].x,self.nodeList[node.pare
nt].y],[node.x,node.y])
        for node in self.nodeList.values():
            if len(node.children) == 0:
                pygame.draw.circle(screen, (255,0,255), [int(node.x),int(node.y)], 2)
        for(sx,sy,ex,ey) in self.obstacleList:
            pygame.draw.rect(screen,(0,0,0), [(sx,sy),(ex,ey)])
        pygame.draw.circle(screen, (255,0,0), [self.start.x, self.start.y], 10)
        pygame.draw.circle(screen, (0,0,255), [self.end.x, self.end.y], 10)
        lastIndex = self.get_best_last_index()
        if lastIndex is not None:
            path = self.gen_final_course(lastIndex)
            ind = len(path)
            while ind > 1:
                pygame.draw.line(screen,(255,0,0),path[ind-2],path[ind-1])
```

```
                ind-=1
        pygame.display.update()
    def GetNearestListIndex(self, rnd):
        dlist = np.subtract( np.array([ (node.x, node.y) for node in self.nodeList.values() ]),
(rnd[0],rnd[1]))**2
        dlist = np.sum(dlist, axis=1)
        minind = list(self.nodeList.keys())[np.argmin(dlist)]
        return minind
    def __CollisionCheck(self, node, obstacleList):
        for(sx,sy,ex,ey) in obstacleList:
            sx,sy,ex,ey = sx+2,sy+2,ex+2,ey+2
            if node.x > sx and node.x < sx+ex:
                if node.y > sy and node.y < sy+ey:
                    return False
        return True  # safe
class Node():
    """
    RRT Node
    """
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.cost = 0.0
```

```
        self.parent = None
        self.children = set()
def main():
    print("start RRT path planning")
    # ====Search Path with RRT====
    obstacleList = [
        (400, 380, 400, 20),
        (400, 220, 20, 180),
        (500, 280, 150, 20),
        (0, 500, 100, 20),
        (500, 450, 20, 150),
        (400, 100, 20, 80),
        (100, 100, 100, 20)
    ]  # [x,y,size]
    # Set Initial parameters
    rrt = RRT(start=[20, 580], goal=[540, 150],
              randArea=[XDIM, YDIM], obstacleList=obstacleList)
    path = rrt.Planning(animation=show_animation)
if __name__ == '__main__':
    main()
```

APPENDIX B

TRANSFORMATIONS IN ROS

The following shows all the frames that can be considered for a 2D motion of a robot. The robot moves around in space and the main idea of these transformations is to be able to quickly answer the question- where is a certain part of the robot located in relation to another part of the robot. It also gives the ability to correctly transform measurements taken from one coordinate frame to another.

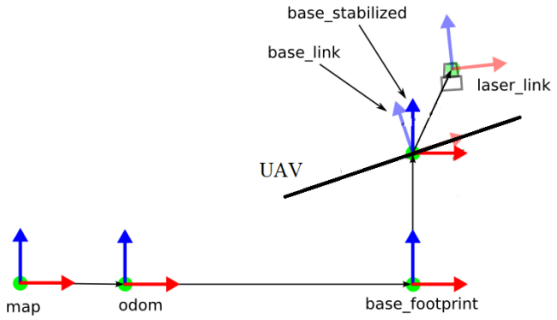

Figure 31: Coordinate Frames for mobile platforms

The base_stabilized frame adds information about height. The base_link frame is related to roll and pitch and gives angles to the base_stabilized frame. If roll and pitch are considered to be nonexistent, the base_link and the base_stabilized frames are the same. The laser could also be located on any part of the robot. So, it is vital to know the transformation from the laser_link to the base_link. All these are available through the transform library.

APPENDIX C

Arduino code for autonomous drive

```
// define for Ultrasonic sensor

#define trigPin 13

#define echoPin_C 11

// Motor A

int enA = 10;

int IN1 = 9;

int IN2 = 8;

// Motor B

int enB = 5;

int IN3 = 7;

int IN4 = 6;

float duration_C,distance_C;

int number;

// Function for check obstacles

float check_obstacle_C(){

  digitalWrite(trigPin,HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin,LOW);

// Measure the response from the echo pin

duration_C = pulseIn(echoPin_C,HIGH);

// determin distance from duraion
```

```
distance_C = (duration_C/2)*0.0343;
if (distance_C > 100 || distance_C == 0){
  distance_C = 100;
}else{
  distance_C = distance_C;
}
return distance_C;
}
void MoveForward(){
  analogWrite(enA,90);
  analogWrite(enB,90+25);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,HIGH);
  digitalWrite(IN4,LOW);
}
void MoveBackward(){
  analogWrite(enA,90);
  analogWrite(enB,90+25);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,HIGH);
```

```
}
void Left_turn(){
  analogWrite(enA,80);
  analogWrite(enB,80+25);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,HIGH);
}
void Right_turn(){
  analogWrite(enA,80);
  analogWrite(enB,80+25);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
  digitalWrite(IN3,HIGH);
  digitalWrite(IN4,LOW);
}
void Stop(){
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,LOW);
}
```

```
void setup() {
  // put your setup code here, to run once:
    // Serial.begin(9600);
    pinMode(trigPin,OUTPUT);
    pinMode(echoPin_C,INPUT);
  // IR sensor input
    pinMode(4,INPUT);
    pinMode(12,INPUT);
}
void loop() {
  // put your main code here, to run repeatedly:
  // Serial.println(distance_C);
check_obstacle_C();
if(distance_C > 30 && digitalRead(4)==HIGH && digitalRead(12)==HIGH){
  MoveForward();
}
else if(distance_C > 30 && digitalRead(4)==LOW && digitalRead(12)==HIGH){
  Stop();
  delay(2000);
  MoveBackward();
  delay(1000);
  Right_turn();
  delay(800);
```

```
}
else if(distance_C > 30 && digitalRead(4)==HIGH && digitalRead(12)==LOW){
  Stop();
  delay(2000);
  MoveBackward();
  delay(1000);
  Left_turn();
  delay(800);
}
else if(distance_C <=30 && digitalRead(4)==LOW && digitalRead(12)==LOW){
  Stop();
  delay(1000);
  MoveBackward();
  delay(2000);
  number = random(1,3);
  if (number==1){
  Left_turn();
  delay(2000);
  }
  else if(number==2){
  Right_turn();
  delay(2000);
  }
```

```
}
else if(distance_C <=30 && digitalRead(4)==HIGH && digitalRead(12)==HIGH){
  Stop();
  delay(1000);
  MoveBackward();
  delay(2000);
  number = random(1,3);
  if (number==1){
  Left_turn();
  delay(2000);
  }
  else if(number==2){
  Right_turn();
  delay(2000);
  }
}
else {
  Stop();
}
}
```

APPENDIX D

Arduino code for localization (rosserial node)

```
/*
 * rosserial Planar Odometry Example
 */
#include <ros.h>
#include <ros/time.h>
#include <tf/tf.h>
#include <tf/transform_broadcaster.h>
// define for Ultrasonic sensor
#define trigPin 13
#define echoPin_C 11
//constans for Interrupt Pins
const byte MOTOR_A = 2;
const byte MOTOR_B = 3;
// Float for number of slots in encoder disk
const float stepcount = 20.00;
const float wheeldiameter = 0.072;
// Integers for pulse conters
volatile int counter_A = 0;
volatile int counter_B = 0;
// Motor A
```

```
int enA = 10;
int IN1 = 9;
int IN2 = 8;
// Motor B
int enB = 5;
int IN3 = 7;
int IN4 = 6;
void ISR_countA(){
  counter_A++;
}
void ISR_countB(){
  counter_B++;
}
float duration_C,distance_C;
int number;
// Function for check obstacles
float check_obstacle_C(){
  digitalWrite(trigPin,HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin,LOW);
// Measure the response from the echo pin
duration_C = pulseIn(echoPin_C,HIGH);
```

```
// determin distance from duraion
distance_C = (duration_C/2)*0.0343;
if (distance_C > 100 || distance_C == 0){
  distance_C = 100;
}else{
  distance_C = distance_C;
}
return distance_C;
}
void MoveForward(){
  analogWrite(enA,90);
  analogWrite(enB,150);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,HIGH);
  digitalWrite(IN4,LOW);
}
void MoveBackward(){
  analogWrite(enA,90);
  analogWrite(enB,90+25);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
  digitalWrite(IN3,LOW);
```

```
  digitalWrite(IN4,HIGH);
}
void Left_turn(){
  analogWrite(enA,90);
  analogWrite(enB,150);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,HIGH);
}
void Right_turn(){
  analogWrite(enA,80);
  analogWrite(enB,80+25);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
  digitalWrite(IN3,HIGH);
  digitalWrite(IN4,LOW);
}
void Stop(){
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,LOW);
```

```
}
ros::NodeHandle  nh;
geometry_msgs::TransformStamped t;
tf::TransformBroadcaster broadcaster;
double x = 0.0;
double y = 0.0;
double A = 0.0;
double theta;
int i=1;
char base_link[] = "/base_link";
char odom[] = "/odom";
void setup()
{
  nh.initNode();
  broadcaster.init(nh);
    // put your setup code here, to run once:
    // Serial.begin(9600);
    pinMode(trigPin,OUTPUT);
    pinMode(echoPin_C,INPUT);
  // IR sensor input
    pinMode(4,INPUT);
    pinMode(12,INPUT);
```

```
  //Serial.begin(9600);
  pinMode(enA,OUTPUT);
pinMode(IN1,OUTPUT);
pinMode(IN2,OUTPUT);
pinMode(enB,OUTPUT);
pinMode(IN3,OUTPUT);
pinMode(IN4,OUTPUT);
attachInterrupt(digitalPinToInterrupt (MOTOR_A), ISR_countA, RISING);// Increase
counter 1 when speed sensor pin goes High
attachInterrupt(digitalPinToInterrupt (MOTOR_B), ISR_countB, RISING);// Increase
counter 2 when speed sensor pin goes high
}
void loop()
{
  // drive in a circle
  //double dx = 0.01;
  //double dtheta = 0.18;
  //x += cos(theta)*dx*0.1;
  //y += sin(theta)*dx*0.1;
  //theta += dtheta*0.1;
  //if(theta > 3.14)
    //theta=-3.14;
 //x += dx*0.1;
```

```
 check_obstacle_C();
if(distance_C > 30 && theta == 0){
  MoveForward();
 A = A + counter_A;
 counter_A = 0;
 x = cos(theta)*(A/20)*(wheeldiameter*3.14);
 y = sin(theta)*(A/20)*(wheeldiameter*3.14);
}
if(distance_C > 30 && theta == 1.57){
 MoveForward();
 A = A + counter_A;
 counter_A = 0;
 x = x;
 y = sin(theta)*(A/20)*(wheeldiameter*3.14);
}
else if(distance_C <= 65 && theta == 0){
  Stop();
  delay(2000);
  Left_turn();
  delay(1450);
 theta = theta + 1.57;
 x = x;
 y = y;
```

```
 A = 0;
}
else if(distance_C <= 40 && theta == 1.57){
  Stop();
  delay(20000);
}
  // tf odom->base_link
  t.header.frame_id = odom;
  t.child_frame_id = base_link;
  t.transform.translation.x = x;
  t.transform.translation.y = y;
  t.transform.rotation = tf::createQuaternionFromYaw(theta);
  t.header.stamp = nh.now();
  broadcaster.sendTransform(t);
  nh.spinOnce();
  delay(10);
}
```

APPENDIX E

ROS HECTOR SLAM LAUNCH FILE AND CONFIGURATION

Mapping_default.launch file

```
<?xml version="1.0"?>
<launch>
  <arg name="tf_map_scanmatch_transform_frame_name"
default="scanmatcher_frame"/>
  <!--<arg name="base_frame" default="base_footprint"/> modified by xianzhe -->
  <arg name="base_frame" default="base_link"/>
  <!--<arg name="odom_frame" default="nav"/> modified by xianzhe -->
  <arg name="odom_frame" default="base_link"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>
  <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping"
output="screen">
    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="$(arg base_frame)" />
    <param name="odom_frame" value="$(arg odom_frame)" />
    <!-- Tf use -->
    <param name="use_tf_scan_transformation" value="true"/>
```

```
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg
pub_map_odom_transform)"/>
    <!-- Map size / start point -->
    <param name="map_resolution" value="0.050"/>
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />
    <!-- Map update parameters -->
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="map_update_distance_thresh" value="0.4"/>
    <param name="map_update_angle_thresh" value="0.06" />
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />
    <!-- Advertising config -->
    <param name="advertise_map_service" value="true"/>
    <param name="scan_subscriber_queue_size" value="$(arg
scan_subscriber_queue_size)"/>
    <param name="scan_topic" value="$(arg scan_topic)"/>
    <param name="tf_map_scanmatch_transform_frame_name" value="$(arg
tf_map_scanmatch_transform_frame_name)" />
```

```
  </node>
  <!--<node pkg="tf" type="static_transform_publisher" name="map_nav_broadcaster"
args="0 0 0 0 0 0 map nav 100"/> modified by xianzhe-->
<node pkg="tf" type="static_transform_publisher" name="base_to_laser_broadcaster"
args="0 0 0 0 0 0 base_link laser 100" />
</launch>
<?xml version="1.0"?>
```

Geotiff_mapper.launch file

```
<launch>
  <arg name="trajectory_source_frame_name" default="/base_link"/>
  <arg name="trajectory_update_rate" default="4"/>
  <arg name="trajectory_publish_rate" default="0.25"/>
  <arg name="map_file_path" default="$(find hector_geotiff)/maps"/>
  <arg name="map_file_base_name" default="hector_slam_map"/>
  <node pkg="hector_trajectory_server" type="hector_trajectory_server"
name="hector_trajectory_server" output="screen">
    <param name="target_frame_name" type="string" value="/map" />
    <param name="source_frame_name" type="string" value="$(arg
trajectory_source_frame_name)" />
    <param name="trajectory_update_rate" type="double" value="$(arg
trajectory_update_rate)" />
    <param name="trajectory_publish_rate" type="double" value="$(arg
trajectory_publish_rate)" />
```

```
  </node>
  <node pkg="hector_geotiff" type="geotiff_node" name="hector_geotiff_node"
output="screen" launch-prefix="nice -n 15">
    <remap from="map" to="/dynamic_map" />
    <param name="map_file_path" type="string" value="$(arg map_file_path)" />
    <param name="map_file_base_name" type="string" value="$(arg
map_file_base_name)" />
    <param name="geotiff_save_period" type="double" value="0" />
    <param name="draw_background_checkerboard" type="bool" value="true" />
    <param name="draw_free_space_grid" type="bool" value="true" />
    <param name="plugins" type="string"
value="hector_geotiff_plugins/TrajectoryMapWriter" />
  </node>
</launch>
```

APPENDIX F

AMCL launch file and configuration

```
<?xml version="1.0"?>
<!--Software License Agreement (BSD)\file      amcl_demo.launch
\authors   Paul Bovbel <pbovbel@clearpathrobotics.com>, Prasenjit Mukherjee
<pmukherj@clearpathrobotics.com>
\copyright Copyright (c) 2015, Clearpath Robotics, Inc., All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are
permitted provided that
the following conditions are met:
 * Redistributions of source code must retain the above copyright notice, this list of
conditions and the
   following disclaimer.
 * Redistributions in binary form must reproduce the above copyright notice, this list of
conditions and the
   following disclaimer in the documentation and/or other materials provided with the
distribution.
 * Neither the name of Clearpath Robotics nor the names of its contributors may be used
to endorse or promote
   products derived from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WAR-
```

RANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, IN-
DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
-->

```
<launch>
  <!-- Run the map server -->
  <arg name="map_file" default="$(find map_server)/test1.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg
map_file)" />
  <arg name="scan_topic" default="$(eval optenv('rplidarNode', 'scan'))" />
 <arg name="tf_map_scanmatch_transform_frame_name"
default="scanmatcher_frame"/>
<arg name="base_frame" default="base_link"/>
 <arg name="odom_frame" default="map"/>
```

```
  <node name="rplidarNode"          pkg="rplidar_ros"  type="rplidarNode"
output="screen">
  <param name="serial_port"         type="string" value="/dev/ttyUSB0"/>
  <param name="serial_baudrate"     type="int"    value="115200"/><!--A1/A2 -->
  <!--param name="serial_baudrate"     type="int"    value="256000"--><!--A3 -->
  <param name="frame_id"            type="string" value="laser"/>
  <param name="inverted"            type="bool"   value="false"/>
  <param name="angle_compensate"    type="bool"   value="true"/>
  </node>
  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find amcl_rviz)/rviz/amcl.rviz"/>
  <!--- Run AMCL -->
  <include file="$(find husky_navigation)/launch/amcl.launch">
  <!--include file="$(find amcl)/examples/amcl_diff.launch"xianzhe-->
    <arg name="scan_topic" value="$(arg scan_topic)" />
  </include>
<node pkg="tf" type="static_transform_publisher" name="base_link_to_laser" args="0 0
0 0 0 0 /base_link /laser 100" />
<node pkg="tf" type="static_transform_publisher" name="map_to_odom" args="0 0 0 0
0 0 /map /odom 100" />
<!--node pkg="tf" type="static_transform_publisher" name="map_to_base_link" args="0
0 0 0 0 0 /map /base_link 100" /-->
</launch>
```

Amcl node configuration

<?xml version="1.0"?>

<!--Software License Agreement (BSD)

\file amcl.launch

\authors Paul Bovbel <pbovbel@clearpathrobotics.com>, Prasenjit Mukherjee <pmukherj@clearpathrobotics.com>

\copyright Copyright (c) 2015, Clearpath Robotics, Inc., All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that

the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the

following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the

following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Clearpath Robotics nor the names of its contributors may be used to endorse or promote

products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WAR-

RANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, IN-
DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
-->

```
<launch>
  <!--arg name="use_map_topic" default="true"/xianzhe-->
  <arg name="use_map_topic" default="true"/>
  <!--arg name="scan_topic" default="$(eval optenv('HUSKY_LASER_TOPIC', 'scan'))"
/xianzhe-->
  <arg name="scan_topic" default="/scan" />
  <node pkg="amcl" type="amcl" name="amcl">
    <param name="use_map_topic" value="$(arg use_map_topic)"/>
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <param name="odom_model_type" value="diff"/>
```

<param name="odom_alpha5" value="0.1"/>

<param name="gui_publish_rate" value="10.0"/>

<param name="laser_max_beams" value="60"/>

<param name="laser_max_range" value="12.0"/>

<param name="min_particles" value="500"/>

<param name="max_particles" value="2000"/>

<param name="kld_err" value="0.05"/>

<param name="kld_z" value="0.99"/>

<param name="odom_alpha1" value="0.2"/>

<param name="odom_alpha2" value="0.2"/>

<!-- translation std dev, m -->

<param name="odom_alpha3" value="0.2"/>

<param name="odom_alpha4" value="0.2"/>

<param name="laser_z_hit" value="0.5"/>

<param name="laser_z_short" value="0.05"/>

<param name="laser_z_max" value="0.05"/>

<param name="laser_z_rand" value="0.5"/>

<param name="laser_sigma_hit" value="0.2"/>

<param name="laser_lambda_short" value="0.1"/>

<param name="laser_model_type" value="likelihood_field"/>

<!-- <param name="laser_model_type" value="beam"/> -->

<param name="laser_likelihood_max_dist" value="2.0"/>

<param name="update_min_d" value="0.25"/>

```
    <param name="update_min_a" value="0.2"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="resample_interval" value="1"/>
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance" value="1.0"/>
    <param name="recovery_alpha_slow" value="0.0"/>
    <param name="recovery_alpha_fast" value="0.0"/>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>
</launch>
```