

Copyright Undertaking

This work is protected by copyright, with all rights reserved.

By reading and using the work, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the work.
2. The reader will use the work for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this work are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

THE HONG KONG POLYTECHNIC UNIVERSITY

BEng Honours Degree in Mechanical Engineering

Guidance and Navigation of Autonomous Robotics

by

Cui Binghui 1411

KONG Qingyi 1410

LEE,CHIA-HANG 1410

Report submitted in partial fulfilment of the requirements for the Honours Degree of Bachelor of Engineering in Mechanical Engineering

Date of submission: 2018.04.20

Acknowledgement

First of all, we would like to express our sincere heartfelt thanks to our professor, Doctor LT HSU, for his invaluable advice, constant encouragement and precise modification, and we admire his knowledge and his personality.

Then we would also like to express our gratitude to our supervisor, Phd. Wen, who constantly offered tremendous help during the whole project.

Furthermore, we would like to thank those professors and teaching assistants who have been pouring out their care, support and encouragement to us.

Abstract	5
Introduction.....	6
1. Motivation of this project	6
2. Commercial application.....	9
3. Specific project introduction	10
4. Overall objectives	11
5. Specific objectives	12
5.1 Localization.....	12
5.2 Clustering and navigation.....	12
5.3 Path tracking and autonomous control.....	12
5.4 Integration	12
6. Specific planning.....	13
Literature review	21
1. Localization.....	21
1.1 background.....	21
1.2 SLAM (Simultaneous Localization and Mapping)	22
1.3 Map	23
1.4 Localization algorithm selection.....	23
1.5 Autoware platform (open code).....	27
2. Clustering and navigation.....	27
2.1 Pathfinding	27
2.2 Clustering.....	32
3. PID controller.....	37
3.1 PID controller Introduction.....	38
3.2 Some special situation.....	39
3.3 Tuning objective	41
3.4 Tuning method	41
4. Connection and integration.....	42
Technical content	44
1. Technical background.....	44
1.1 Localization.....	44
1.2 Clustering and navigation.....	44
1.3 Path tracking and autonomous control.....	49

1.4 Integration	51
2. Methodology	58
2.1 Localization	58
2.3 Clustering and navigation.....	75
2.3 Path tracking and autonomous control.....	96
2.4 Integration	120
3 Experiment design & data and error analysis	123
3.1 Section test	123
3.2 Communication and connection test	134
3.3 Field test	144
3.4 Problems occurred	147
Discussion and conclusion.....	156
Future work	157
Ref.....	162
Appendix.....	164

Abstract

This project aims to accomplish indoor unmanned drive with KUKA as moving platform and Lidar as 3D sensor. Autoware is utilized as software platform while Rviz is implemented as simulation tool during the whole project. Autonomous movement requires the robot itself to do path planning, collision avoidance, guidance and navigation, which are the main objective functions to develop in this project. To be specific, the whole project can be divided into four parts which are:

1. Localization

2. Path finding and clustering

3. Path tracking and autonomous control

4. Integration

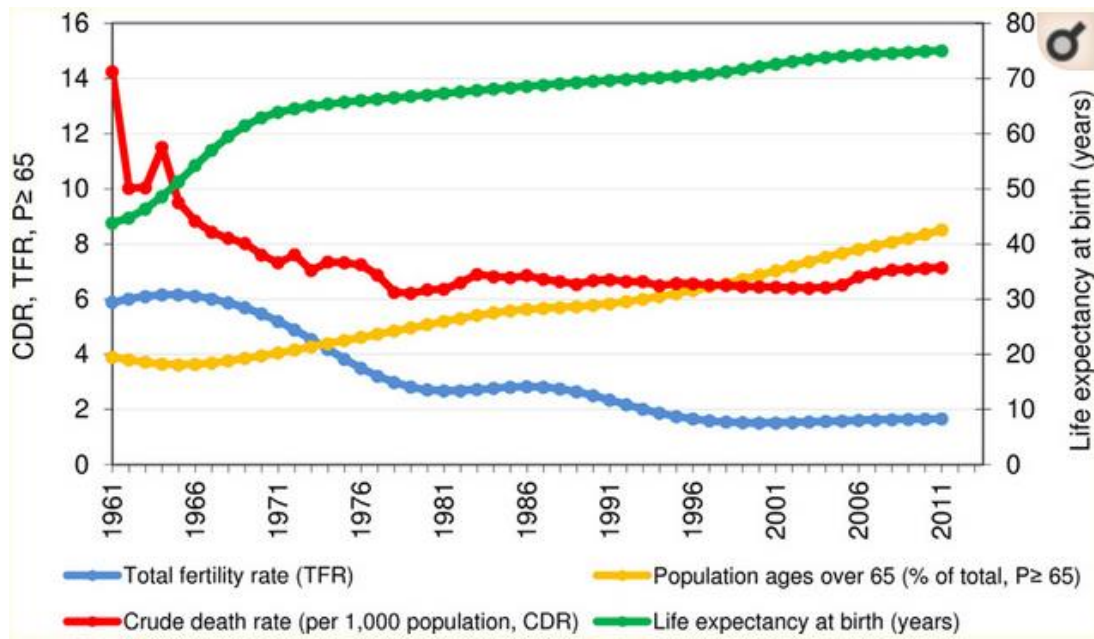
Therefore, first we will introduce the motivation, commercial applications as well as specific objectives and missions. Then literature review and background information of each separate part will be provided, followed by technical content, which contains specific process, details and codes. Some experiments have been designed and conducted to test the operability as well as the accuracy. In the end, conclusion will be given together with discussion and further development.

Introduction

1. Motivation of this project

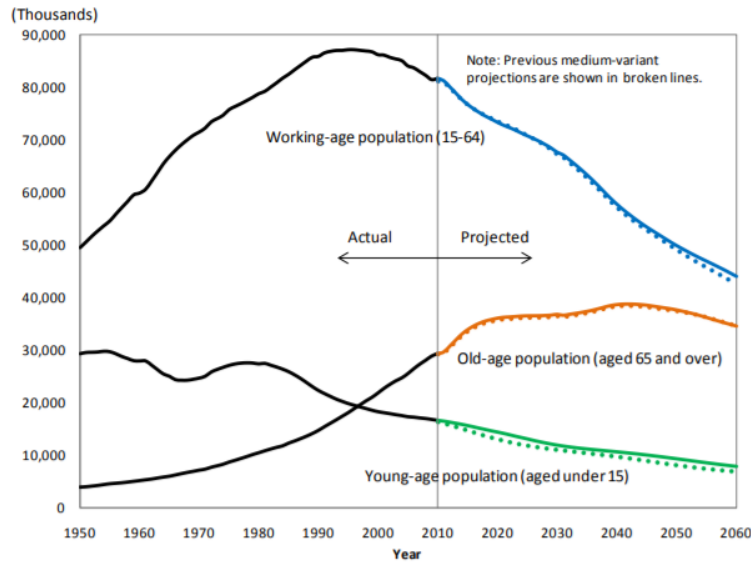
For the past hundreds of years, people have interest in making a robot. Accompany with the development of the computer and automatic technology, robotic technology becomes mature. Meanwhile, 5G and Artificial Intelligence are also developing at a very fast speed. All of these fundamental technologies can help to achieve autonomous technic. So, we try to combine these two concepts and focus on robotic navigation.

Meanwhile, the application of robotic navigation is easy to be commercialized. For example, the express industry in Mainland China thrives, but most of the companies' profit model is still based on demographic dividend (the company can hire courier at a relatively low cost). In the future, aggravating trend of the aging population can change the industry and our indoor robotic navigation system can provide another method- "robotic courier" to greatly decrease their cost. (As time pass by, the fertility rate is continuously decreasing. However, the proportion of population ages over 65 is increasing. So, the aggravating trend of the aging population is more and more intense in China. The Labor cost will also increase and the profit model which is still based on demographic dividend should be changed)



(Fang,2015)

The case of aging is not limited to China. The astonishing trend of aging population exerts great pressure on the labor-intensive mode of manufacturing. Beyond the scope of manufacturing, service industry suffers from the identical shortage of labor force. For the case of Japan, the working-age population shrinks from the peak at 80 million to 70 million in twenty years. The time span is counted from the 90s in last century to present time. Based on the approximation, the working age population will further downgrade to the level less than 50 million. Since the population in Japan is 1.2 billion nowadays, the approximation is terrifying (Takahashi,1999). The severe trend is not limited to Japan. European Union, the coalition led by developed countries, encounters similar threat (See appendix).



(Takahashi,1999).

Automation is one of the solutions to the shrinkage of the labor force. Research and development on the automated substitution of human force have already begun and becomes the souring area of investment. Autonomous driving, one of the automated applications in the service industry, is the main focus of the project.

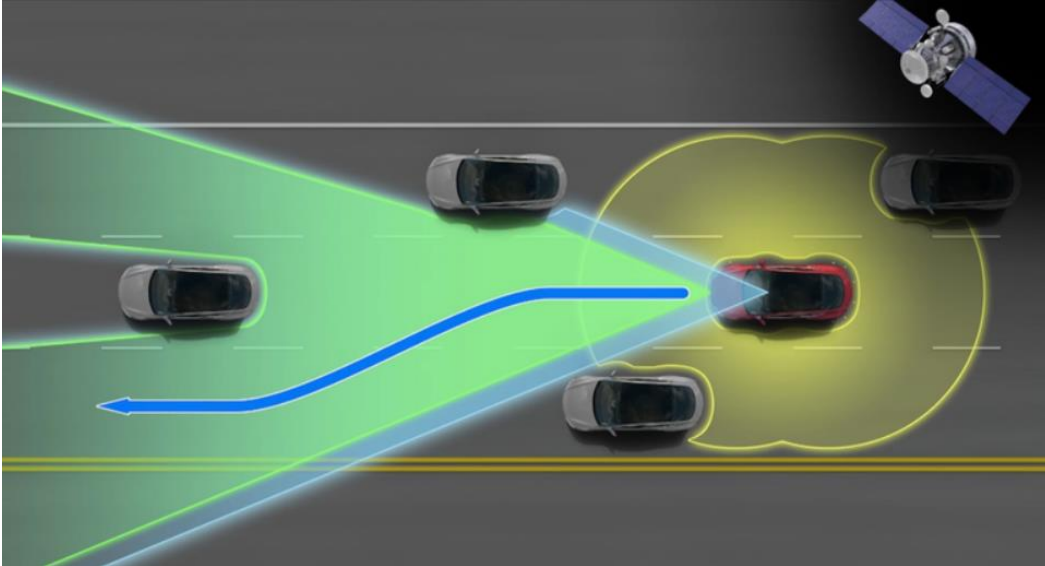
In the project, three main streams of autonomous driving (localization, path-planning and path tracking) will all be mentioned and analyze comprehensively. The interaction among dependent subsystems, error encounter and underlying potentials of the program will be included.

The project aims not only demonstrates the devotion of group members but also attempts to inspire the reader of the report

2. Commercial application

Autonomous driving has the tremendous market in the future. According to Techsci research report published in 2015, the semi-autonomous car will be the fastest growing section in next 10 years. In 2025 and 2035, the full-autonomous car will be the dominant sector (Mathew,2015). In addition, Asia-Pacific region will be the largest market of an autonomous car in the world. Hongkong will be an ideal metropolis to conduct relative research.

The scope of the driverless car should not be limited to private cars. Business cars, included haulage system, forklift, trucks and mining vehicles, are ideal for the application of autonomous driving technology. Tesla 's semi-autonomous electric driverless trucks, accompanied by the auto-pilot system consists of GPS, camera, and radar causes huge impacts in industry. Autopilot system enables the adjustment of speed, detect obstacles and applying brakes. Though autopilot system still demands drivers for accident avoidance, driverless car has the great market potential for fully replacing drivers in future.



3. Specific project introduction

The robotic problem can be roughly divided into three parts: Where am I? &Where will I go? &How to go?

Localization

In this part, the question “where I am?” will be introduced. Localization is an important part of the whole navigation system because its accuracy can greatly influence path planning and then influence the performance of the whole system. By using LiDAR sensor, the robot can get a large quantity of environment information. Then, applying some probabilistic robotics theory and algorithm can help predict the current pose. Combining with the pre-made pcd map, localization output can be more accurate.

Cluster and Path Finding:

The path planning involves the part of object identification by applying mathematical algorithms. Then, based on the object identification and stratification, the edge of the zone for the blockade avoidance is decided. Accompany with the construction of the map, navigation set will integrate the blockade information and derives the path for the robot to follow. The part of path planning aims at resolving the issue of object avoidance and ideal path design for the robot.

Path tracking:

The mission of this part is to do the path tracking and autonomous control for robotics in indoor environment. Furthermore, simple path planning is needed to avoid obstacles in the pathway. Path tracking and autonomous control are fundamental problem that need to tackle in navigation for robotics with lots of movements. With both environment perception and clustering being finished, robotics need to conduct path tracking and control to implement the required task.

4. Overall objectives

The project aims at utilizing the knowledge related to robotic control, guidance and navigation. Some basic requirements are required to be achieved: Path planning, collision avoidance, guidance and navigation.

5. Specific objectives

5.1 Localization

Apply the relative coordinate system to confirm the whole system uses the same coordinate system.

Use LiDAR to scan the indoor experiment region beforehand and generate high precision map

With the help of Autoware platform to achieve SLAM and generate the topic which contains constantly localization coordinate

5.2 Clustering and navigation

Cluster the point into certain group by Euclidean cluster.

Extract the centroid from each clustered group.

Complete the part of conversion from 3D to 2D mapping

Connect the part of clustering to the path planning function.

5.3 Path tracking and autonomous control

Be familiar with KUKA youbot and able to control it.

Design path planning method

Finish and implement packages and topics in KUKA's ROS system

5.4 Integration

Use switch and Ethernet cable to build a LAN and finish the ping test in relevant computers.

Use topic to achieve multi-computer connection in ROS system.

Evaluate the system that can meet the accuracy and stability requirement.

6. Specific planning

Stage	Period	Detail (localization/Detection and path planning/Path tracking and control)
Be familiar with the project background	04/09/2017 to 04/10/2017	Read some literature material. (ndt, Monte Carlo Localization) Be familiar with Ubuntu & ROS. Adjust the software environment. Know about the hardware. (LiDAR, Kuka robot)
		Literature material. (K means, DCDC, SVM) Construct the software system Adjust the software environment.(Ubuntu, ros)

		Literature review on PID controller. Install Ubuntu system and some other software. Practice on Ubuntu system.
Try to complete the whole procedure roughly	04/10/2017 to 31/10/2017	Install the Autoware and be familiar with inner packages
		Autoware installation. Mainly focuses on the processing of euler cluster. Try to install the bounding box in the process.
		Get access to KUKA youbot. Follow the tutorials of the guide book. Learn the codes in the KUKA packages.
Use LiDAR to map the final demonstration field	31/10/2017 to 14/11/2017	Use LiDAR to map the final demonstration field. Improve the open code method to fulfill specific accuracy. Use the mapping data to run the program.
		Mainly try to test the former rough procedure. Analyze the code.

		Balance the workload and unify the work process. Investigate path finding algorithm (A star)
		Build a base for the Lidar. Modify the packages in KUKA youbot to achieve some functions. Build my own executable packages. Conclude the topics and messages we need to use with my teammates.
Make a connection with other group members	14/11/2017 to 22/12/2017	Mainly try to test the former rough procedure. Balance the workload and unify the work process.
		Integrate clustering method with the centroid publication method for path planning.
		Finish the interim report. Continue learning C++ language and ROS system.

Make a connection with other group members	22/12/2017 to 02/01/2018	Mainly try to test the former rough procedure. Balance the work load and unify the work process. Try to install the LiDAR fixer.
		Start to integrate the centroid publication ,localization and path tracking. Try pathfinding (A star) method Finish up the point cloud tutorial
		Finish building my own packages. Learn necessary contents from my teammates to prepare for the integration. DO some simple testing about KUKA.
Test in the real situation(parts)	02/01/2018 to 30/01/2018	Basing on the experiment performance, further improve the accuracy of the system.
		Try to finish up the integration of path finding and clustering.
		Balance the workload and unify the work process.

Test in the real situation(parts)	30/01/2018 to 28/02/2018	Focus on integration. Start to consider alternative methods to integrate the parts
		Testing focuses on the stability. test the path planning algorithm in the project. Integrate the overall programs.
		Start to finalize the parameters and test on the KUKA youbot.
Finish all the new program work and hardware installation	28/02/2018 to 28/03/2018	Design and finish the ping test, topic test and final test. Finish the multi-computer communication.
		Checking for deficiencies and correct whether something wrong.
		Design field tests. Collect necessary data and make videos.
		Cooperate with partners to finish indeed test.

Final preparation and adjustment	28/03/2018 to Deadline	Pay more attention on paper writing and demonstration. Regularly check the system. Organize the useful data for next year student.
		Conduct field tests. Finish final report.

Equipment:

Personal computer:

- CPU: Intel i5-4200U
- Operating system: ubuntu 14.04 indigo
- Mother board: Dell 0T3T8M (Intel Haswell-ULT - Lynx Point-LP)
- DDR SDRAM: 12 GB (Hynix DDR3L 1600MHz)
- Hard Disk: Seagate ST500LT012-1DG142 (500 GB / 5400 rec/min)
- Video card: Nvidia GeForce GT 750M (2 GB)
- Network card: Intel Dual Band Wireless-N 7260

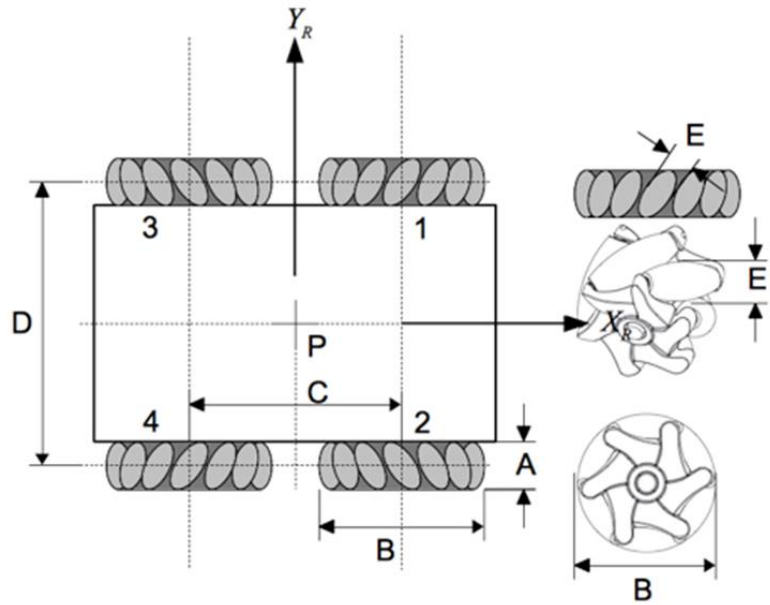
LiDAR:



- ▶ Dual Returns
- ▶ ± 2 cm accuracy
- ▶ 1kg (plus 0.3kg for cabling)
- ▶ 32 Channels
- ▶ 80m-100m Range
- ▶ 700,000 Points per Second
- ▶ 360° Horizontal FOV
- ▶ +10° to -30° Vertical FOV
- ▶ Low Power Consumption
- ▶ Rugged Design

Kuka:

KUKA youbot is widely used in many industrial applications and in this project, we will mainly use its base as a mobile platform. The operating system of KUKA is ROS (Robot Operating System). In the KUKA for this project, the ROS version is Hydro 12.04, which makes it difficult to install packages of other versions. Due to the version problem and the low-performance of the internal computer, we decide to use the external computer and the switch for communication. What needs to be emphasized is that KUKA has four omnidirectional wheels, which makes it easy to control its direction instead of considering the steering angle. The dimension and computer parameters of KUKA are introduced below:



$A = 74.87 \text{ mm}$ $B = 100 \text{ mm}$ $C = 471 \text{ mm}$ $D = 300.46 \text{ mm}$ $E = 28 \text{ mm}$ (max diameter of the roll)

CPU : Intel Atom D510 Dual Core 1.66 GHz

RAM : 2 GB; SO-DDR2-667 200PIN

DC Input : 12 V

Total weight: 20kg

Load: 5kg

Runtime: 90mins

Literature review

1. Localization

1.1 background

Localization is an important part of the whole navigation system because its accuracy can greatly influence path planning and then influence the performance of the whole system.

It is easy to understand because localization part should pass data to path planning part. If the initial data is wrong or inaccurate, the final performance cannot be good.

Nowadays probabilistic robotic is popular and our situation matches with it. So, most of the theories and ideas come from it. First, some basic idea about probabilistic robotic will be introduced. Then, methods will be selected according to our specific situation. For example, relevant algorithms such as Monte Carlo localization, MARKOV Localization and NDT-MC Localization will be introduced and I will explain which algorithm will be chosen and why it is the best choice.

Why localization problem is difficult

First of all, the computer needs time to calculate the data. Even though the modern computer is very high technical, LiDAR map contains a lot of quantities of data compared with the traditional sensor, so the time-consuming cannot be ignored (Thrun, 2005) .

In the other aspect, if the robot only uses the inertia surveying system, drift and skidding cannot be avoided. With time going on, the error will grow and become inaccurate. So, the device needs more detail information to correct the localization.

How to achieve it

The objective is touchable because of these following reasons. First of all, the noise is predictable. After analyzing the predicted data, we can module a conditional probability density. The main idea comes from the Probabilistic Robotics (Thrun, 2005).

This is the basic algorithm of the noise prediction. Noise can be divided to hit, short, max and rand. (Hit: Correct range with local measurement noise; Short: Unexpected objects; Max: Failures; Rand: Random measurements)

```
1:  Algorithm beam_range_finder_model( $z_t, x_t, m$ ):  
2:       $q = 1$   
3:      for  $k = 1$  to  $K$  do  
4:          compute  $z_t^{k*}$  for the measurement  $z_t^k$  using ray casting  
5:           $p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k | x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k | x_t, m)$   
6:               $+ z_{\text{max}} \cdot p_{\text{max}}(z_t^k | x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k | x_t, m)$   
7:           $q = q \cdot p$   
8:      return  $q$ 
```

Figure 6 The basic step of beam range finder model

1.2 SLAM (Simultaneous Localization and Mapping)

SLAM problems are difficult because robot neither has the map nor has position. (Thrun, 2005) Without enough support information, almost all of the decisions are made by the robot itself. Because it is difficult, SLAM is only used in mapping generation part but does not use it in the localization(ndt_matching) in the project. Meanwhile, Autoware platform contains the part and I can directly use it, so I do not focus on SLAM.

1.3 Map

According to the Probabilistic Robotics, map is a list of objectives and it can be divided into many branches depending on its characters such a Feature-based map and Location-based map (Thrun, 2005) . High-precision map is also important in localization part. To achieve getting the relative pose, the system needs a coordinate and reference frame. Meanwhile, LiDAR detection has accuracy advantages, so we use LiDAR making a 3D high-precision map to achieve a relatively high accuracy performance.

The map which is made by LiDAR is point cloud map. Comparing with the traditional map, it is more accurate and reliable. However, it also contains more information and needs more computer power to run it. In the project, the LiDAR will storage it's scanning information and storage it in bag file. Some algorithm is needed to transfer it to pcd file.

1.4 Localization algorithm selection

Localization is a big topic and it also can be divided to many sub-topics. A lot of existing algorithms can solve the localization problem. According to the specific situation environment and accuracy demand, methods can be chosen to achieve localization.

Real situation that our project will face is that

- i. Demo environment is indoor. So, there can be less unpredictable factor. We do need to solve a large quantity of noise because Some noise can be avoided (such as sun, atmosphere pressure difference, temperature difference)

- ii. The environment can be seen as a static environment. The objective require that all of the objects are static.
- iii. Do not need to consider the rotation pose. In localization part, the output should include x,y,z, pitch, yaw, roll and these factors can indicate the pose of the robot. Because Kuka robot can translate left and right, we only need to consider x,y.
- iv. Need relative high accuracy (because of the area and robot size)

Alternative algorithms

MARKOV Localization

MARKOV localization transforms a probabilistic belief at time $t-1$ into a belief at time t and the initial belief, $bel(x_0)$ reflects the initial knowledge of the robot's pose.

Meanwhile, point-mass distributions are discrete. therefore, do not possess a density. The algorithm makes decision depend on the data which is generated by multiplying its belief $bel(x_0)$ by $p(z_t | x_t, m)$. It can be applied on almost all situation.

```

1:  Algorithm Markov Localization( $bel(x_{t-1}), u_t, z_t, m$ ):
2:    for all  $x_t$  do
3:       $\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}, m) bel(x_{t-1}) dx$ 
4:       $bel(x_t) = \eta p(z_t | x_t, m) \overline{bel}(x_t)$ 
5:    endfor
6:    return  $bel(x_t)$ 

```

EKF Localization

EKF Localization is a special case of Markov localization. It represents $bel(x_t)$ by their first and second moment: mean μ_t and the covariance Σ_t .

It also transforms a probabilistic belief at time $t-1$ into a belief at time t same as MARKOV. (input a Gaussian estimate of the robot pose at time $t-1$, with mean μ_{t-1} and covariance Σ_{t-1}) Because it is a second order algorithm, it need more calculation power which can cause delay. So, it cannot consider so many features and need to bound to a limited parametric subset of distributions. In the other word, it cannot be used in the situation that has a lot noisy and error (because Taylor is used to approximate μ_t and Σ_t)

```

1: Algorithm EKF.localization.known_correspondences( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t, m$ ):
2:    $\bar{\mu}_t = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1, \theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1, \theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1, \theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1, \theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$ 
3:    $G_t = \begin{pmatrix} 1 & 0 & \frac{v_t}{\omega_t} \cos \mu_{t-1, \theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1, \theta} + \omega_t \Delta t) \\ 0 & 1 & \frac{v_t}{\omega_t} \sin \mu_{t-1, \theta} - \frac{v_t}{\omega_t} \sin(\mu_{t-1, \theta} + \omega_t \Delta t) \\ 0 & 0 & 1 \end{pmatrix}$ 
4:    $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$ 
5:    $Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_\delta \end{pmatrix}$ 
6:   for all observed features  $z_t^i = (r_t^i \ \phi_t^i \ s_t^i)^T$  do
7:      $j = c_t^i$ 
8:      $\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} m_{j,x} - \bar{\mu}_{t,x} \\ m_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$ 
9:      $q = \delta^T \delta$ 
10:     $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \\ m_{j,s} \end{pmatrix}$ 
11:     $H_t^i = \frac{1}{q} \begin{pmatrix} \sqrt{q} \delta_x & -\sqrt{q} \delta_y & 0 \\ \delta_y & \delta_x & -1 \\ 0 & 0 & 0 \end{pmatrix}$ 
12:     $K_t^i = \bar{\Sigma}_t H_t^{i,T} (H_t^i \bar{\Sigma}_t H_t^{i,T} + Q_t)^{-1}$ 
13:  endfor
14:   $\mu_t = \bar{\mu}_t + \sum_i K_t^i (z_t^i - \hat{z}_t^i)$ 
15:   $\Sigma_t = (I - \sum_i K_t^i H_t^i) \bar{\Sigma}_t$ 
16:  return  $\mu_t, \Sigma_t$ 

```

According to our specific situation, we choice Monte Carlo localization method because of these following reasons: First, Monte Carlo Localization is the most popular localization algorithm in robotics. So, a lot of open code reference is available, and we have many learning materials. Comparing with other algorithms such as MARKOV Localization, it represents the belief $bel(x_t)$ by particles (more particles means high belief). It means that high belief place always correspondence high particle density (Thrun, 2005) .

However, particle density can cause large calculation even though it can increase the accuracy by increasing the total number of the particle (Thrun, 2005) . Meanwhile, it cannot recover from robot kidnapping. In the localization process, it needs LiDAR and Odom work together to fulfill the requirement which is a little bit complex. After consulting with our supervisor, another algorithm- NDT-MCL Localization method (Normal distribution transform) which is a brand of MARKOV Localization was found. NDT-MC Localization method (Normal distribution transform) can generate many 2D maps and integrate them to a 3D map instead of matching each particle. Generate many 2D maps and integrate them to a 3D map (only need to match the surface). So, NDT localization can decrease the computation and increase the reaction speed in some way. Another reason is that NDT-MC Localization does not need Dead Reckoning(Odom). It can decrease my partner's work load. With the help of literature information, the rough direction can be found. Follow the hint, a powerful platform-Autoware was found. The Localization part in it is mainly basing on NDT-MC Localization method.

1.5 Autoware platform (open code)

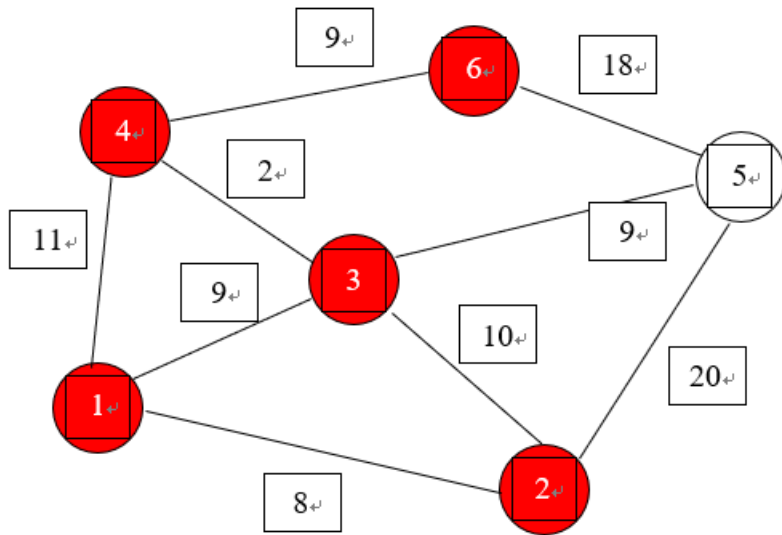
Autoware is an open-source software for urban autonomous driving, maintained by Tier IV. It contains following functions: 3D Localization, 3D Mapping, Path Planning, Path Following, Accel/Brake/Steering Control, Data Logging, Car/Pedestrian/Object Detection, Traffic Signal Detection, Traffic Light Recognition, Lane Detection, Object Tracking, Sensor Calibration, Sensor Fusion, Cloud-oriented Maps, Connected Automation, Smartphone Navigation, Software Simulation, Virtual Reality. 3D localization, 3D Mapping and simulation can be used in our project. In the project, we will use it to achieve our objectives.

2. Clustering and navigation

2.1 Pathfinding

2.1.1 Dijkstra Algorithm

Path planning includes several ways. One way is called Dijkstra algorithm. Dijkstra algorithm calculates the distance between a starting point of the vehicle and the destination of the vehicle. First, we can assume that there are several nodes between the starting point and the destination. There are different weights between nodes. Usually, the weights can be assumed as the distance. If the distance is longer, the weights can be higher, so does vice versa. Dijkstra algorithm will compare the distance between each node and select the one with the shorter distance, or lower weight.



We assume that there is the function called G , which represents the point we have visited. Starting node is called V . U represents the node we have not visited. S is the interim summation of weights.

As shown in the graph, the starting node is 1 and the final node is 6. Initially, G only includes node V , or node 1. Then, selecting from U , nodes of 4 and 5 should be excluded from the consideration, the rest of points should be considered as nearby nodes and comparison process starts. During the comparison, we calculate the interim sum, node 4 is 11, node 3 is 9 and node 2 is 8. We select next node with lower weights, or shorter distance, which is node 2 with the interim sum of 8.. Now, G include point node 1 and node 2. Node 2 is removed from U . Then, from node 2, we repeat the process of comparison and interim summation. Interim summation from node 2 to node 3 is 18. Comparing with the previous interim summation to node 3, which is 9, the interim sum of

the 1 st round is lower than that of the second round, $17 > 9$. The interim sum of node 4 is 22. The summation is Repeating the process of comparison, we can have the interim weight of 20 at point 5, which is the shortest distance.

By using Dijkstra algorithm, we can find the shortest distance between staring node and final node. However, Dijkstra algorithm is time-consuming, we need an alternative solution.

2.1.1 A* Algorithm

A star algorithm aims at finding the least value of f . f consists of two components: path cost function $g(x)$ and $h(x)$ as the heuristic function.

$$f(x) = g(x) + h(x)$$

We can assume cost as the distance. $g(x)$ is the summation of cost from the starting position, or node, to the current node (Wu,Zheng,Lai &Suen,2015). $h(x)$ is the heuristic function, which is the estimated cost from the current node to the end node. In the example, we can assume that the cost of moving parallel to x and y axis is 1 and assume that the cost of diagonal movement as $1.4141\dots$, since root of the summation of the square of x and y unit, which is 1, is 1.4141 .

$$g(x) = \sum \text{distance (cost) have moved}$$

For the part of heuristic function, there are several methods for computation. The most widely used method is Manhattan distance. Manhattan distance calculates the summation of the absolute value of difference between location of x at current node and that of the compared node with the difference between location of y at current node and that of the compared node.

$$h(x) = \sum D * \text{abs}(\text{node}.x - \text{goal}.x) + \text{abs}(\text{node}.y - \text{goal}.y)$$

Manhattan distance has drawbacks. The most serious drawback is that Manhattan distance only accepts the movement parallel to x and y axis. Therefore, we might choose Euclidean distance instead.

Euclidean distance follows some steps of h(x), but still has some differences

function Heuristic =

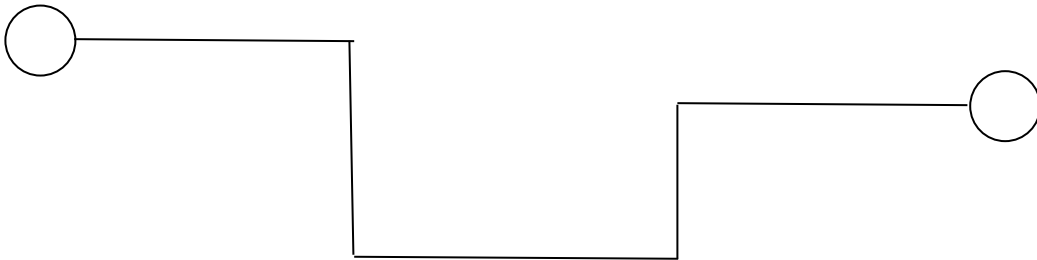
$$dx = \text{abs}(\text{node}.x - \text{goal}.x)$$

$$dy = \text{abs}(\text{node.y} - \text{goal.y})$$

$$\text{distance} = D * \text{sqrt}(\text{dx} * \text{dx} + \text{dy} * \text{dy})$$

Euclidean distance returns the square root of dx square and dy square, which means diagonal movement is possible. Diagonal movement is similar with the real condition.

(a) Manhattan Distance



(b) Euclidean Distance



Then, $h(x)$ will combine with $g(x)$, the summation of the distance that the vehicle has already moved, and we can get $f(x)$. D is the factor that we can adjust for the speed of

finding the path. If D is larger, the heuristic function will be more influential, which means $g(x)$ is less influential. If $h(x)$ is larger, pathfinding function will mainly follow the estimated path and the speed is faster, but we may not find out the shortest path from start node to the end node. If D is larger, opposite outcome will occur. $g(x)$ will dominate and the algorithm will degrade into Dijkstra algorithm.

2.2 Clustering

2.2.1 K means Clustering

K means cluster is one of the major clustering methods that requires no learning, which is different from the classifier. We do not need to classify the member into specific categories. Instead, we need to cluster the unlabeled points into coherent subsets.

There are two steps for K means algorithm. The first step is the cluster assignment step. Next step is the cluster centroid rearrangement. First, we specify the clustering centroid and cluster points that are closer to the preset centroids into groups. In the step of cluster centroid rearrangement, we calculate the mean value of the points in each clustered group and move the centroid to the calculated mean. Then, we repeat the 1st step and 2nd step until points in the clustered group converge.

Mathematical Steps can be written as follow.

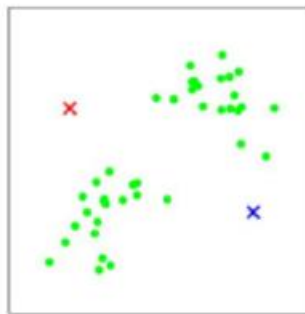
1. Input: point set $\{x_1, x_2, x_3 \dots \dots x_n\}$
2. We randomly select the $\mu_1, \mu_2 \dots \mu_k$ as the clustering mean
3. cluster the points to the nearest μ_k
4. $c(i) = \min \|x(i) - \mu_k\|^2$

Should be aware that μ_k in the 4th step is the mean of the points in the cluster. The resulted $c(i)$ is the index that the cluster k where $x(i)$ is assigned for.

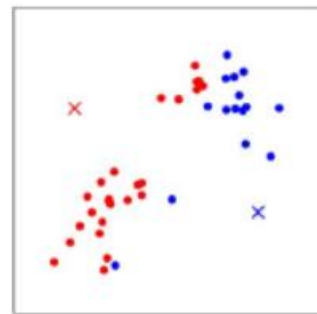
5. Calculate the mean $\mu_{c(i)}$ based on the value of x with the same $c(i)$ in the cluster k ,
where $k=c(i)$
6. Repeat 4th and 5th step.



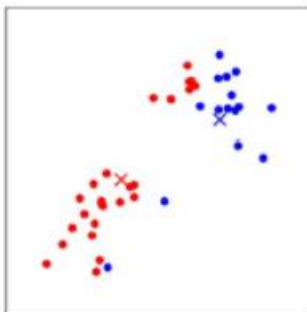
(a)



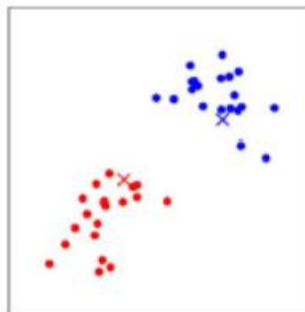
(b)



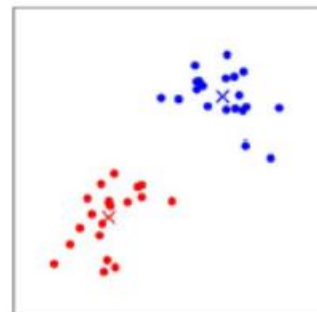
(c)



(d)



(e)



(f)

In the graph (a), green data indicates points scanned. In the graph (b), we randomly assign two centroids μ_1 and μ_2 , labeled as red and blue in the graph. Then, apply the formula stated in the 4th step. $x(i)$ point will be assigned to the cluster k with the minimum with the distance. The targeted cluster k of point $x(i)$ is stored in the index $c(i)$. For instance, if $c(1)=2$, that means point $x(1)$ is assigned to the cluster number 2. The computing result is shown in graph c. Then, we calculate the mean $\mu_{c(i)}$, which is based on the points $x(i)$ clustered in the clusters with the index value $c(i)$. We repeat the process of assigning points to cluster and calculation of new mean points. Finally, the cluster will reach the equilibrium, which indicates both cluster and cluster centroid will be fixed. The state of the clustering is calculated based on the distortion cost function.

$$J(c^{(i)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c(i)}\|^2$$

The J indicated the average square distance between point $x(i)$ and cluster $\mu_{c(i)}$. In general, the J will decrease continuously until it reaches the constant value.

K means is useful but still has some drawbacks. One of the major drawback is that the initial clustering centroid need to be preset, but most of the cases that require clustering involves points with preset value.

2.2.2 Euclidean Cluster

Euclidean cluster is the special type of Euclidean cluster, which contains the kernel. The kernel of Euclidean cluster involves the conjugate and defines in the complex space, which is different from the traditional kernel. Euler kernel is a matrix. Each element of the matrix in (j,q) form is calculated based on Kernel matrix $G_{jq} = \Phi^T \Phi$. $\Phi = \{ \phi_1, \phi_2, \dots, \phi_n \}$

$$G_{jq} = \frac{1}{2} \sum_{c=1}^d \cos(\alpha\pi(x_j(C) - x_q(C))) - i \frac{1}{2} \sum_{c=1}^d \sin(\alpha\pi(x_j(C) - x_q(C)))$$

Big letter C represents the number of clusters. Small letter c represents the cluster that the point set x(i) locates. d is the number of dimensions (Wu,Zheng, Lai &Suen,2015) .

The mapping function

ϕ transforms the d dimensional real space into d dimensional complex space.

ϕ is the mapping function. The mapping function is the key component in the kernel.

$$\phi(x_j) = \frac{1}{\sqrt{2}} e^{i\alpha\pi} = \frac{1}{\sqrt{2}} (\cos(\alpha\pi x_j) + i\sin(\alpha\pi x_j))$$

Since Euclidean cluster is the special type of K means, determination of distance between points and centroid of cluster is necessary.

$$d(\phi(x_j), \phi(x_q)) = \|\phi(x_j) - \phi(x_q)\|^2 = \sum_{c=1}^d (1 - \cos(\theta_j(c) - \theta_q(c)))$$

The mean of the cluster m_c can be determined from

$$m_c = \frac{1}{\sqrt{2}}(a + bi)$$

$$a = \frac{1}{nc} \sum_{x_j \in g_c} \cos(\alpha \pi x_j)$$

$$b = \frac{1}{nc} \sum_{x_j \in g_c} \sin(\alpha \pi x_j)$$

g_c cluster includes the x_j points. Since we have distance and mean functions, we have to converge the data set, or letting the points to fix in the cluster by minimizing the error.

The distortion error f is

$$f = \sum_{c=1}^C \sum_{j=1}^n U_{jc} d(\phi(x_j), m_c)$$

$U_{jc} = 1$ if the point x_j belongs to g_c . $U_{jc}=0$ is the point x_j does not set in g_c .

Algorithm can be written as follow.

Algorithm 1 Euler Clustering

- 1: **Input:** Dataset $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, the number of clusters C , the value α , and the maximum number of iterations t_{max} .
 - 2: **Output:** The cluster sets $\mathcal{G}_c, c = 1, \dots, C$.
 - 3: **Initialization:**
Initialize the cluster labels of data points.
Transform the data points from data space onto feature space using Eq. (6).
 - 4: **repeat**
 - 5: $t = t + 1$;
 - 6: Fix the cluster labels of the data points, and update the cluster representative prototypes $\mathbf{m}_c$ using Eq. (10).
 - 7: Fix the cluster representative prototypes $\mathbf{m}_c$, and update the cluster labels of the data points using Eq. (12).
 - 8: **until** convergence || $t > t_{max}$.
-

(Wu,Zheng,Lai &Suen,2015)

3. PID controller

PID controller: In this project, knowledge about control system is required and PID controller, which stands for the proportional-integral-derivative controller, will be briefly introduced in this section. PID controller is widely accepted in industrial applications because it is able to compensate a variety of practical industrial processes. Although with the development of the automatic control, there are more options for control schemes, yet more than 90% of industrial controllers are still implemented based around PID algorithms, particularly at lowest levels (W. S. Levine, Ed. Piscataway, NJ: IEEE Press,

1996). In this section, I will introduce the basic knowledge, tuning methods as well as tuning objectives of PID controller.

3.1 PID controller Introduction

PID controller consists of three terms: proportional term, the integral term, and derivative term. The function of proportional term is to provide an overall control action proportional to the error signal. The integral term is the sum of instantaneous error and aims to reduce the steady-state error. The derivative term is related to the slope of the error over time and aims to predict the system behaviors. The S domain function of PID controller is given below:

$$G(s) = K_p + K_i \cdot \frac{1}{s} + K_d \cdot s$$

The following picture shows the time-domain function, which can be transferred to S-domain function by Laplace Transfer.

As these three terms change, many parameters in the control system will change correspondingly. The following table shows the influence of these three terms to the closed-loop response.

Closed-loop response	Rise time	Overshoot	Settling time	Steady-state response	Stability
Increase K_p	Decrease	Increase	Small increase	Decrease	Degrade
Increase K_i	Small decrease	Increase	Increase	Largely decrease	Degrade
Increase K_d	Small decrease	Decrease	Decrease	Minor change	Improve

3.2 Some special situation

There are some special cases for the control system if some terms are not chosen properly. One case is called integral windup. Integral windup happens when the process variable is larger than the maximum range limit or lower than the minimum range limit. This may cause by the wrong selection of proportional or integral term. When this case happens, the future correction in the control system will not be processed until the saturation is over. Currently, nearly all software packages and hardware modules have implemented some form of integrator anti-windup protection. (Kiam Heong Ang, Chong

and Yun Li, 2005) One general method is to add an anti-windup term by modifying the integral term to:

$$\frac{1}{T_I s} \left[K_P E(s) - \frac{U(s) - U'(s)}{\gamma} \right]$$

where $U'(s)$ represents the saturated control action and γ is a correcting factor. Usually, when γ is in the range of [0.1, 1.0], the whole system may perform well.

Generally, the derivative term will improve the stability because it can provide a phase lead to offset phase lag which is caused by integration. However, the derivative term may destroy the stability because of the sudden and large change of the output. Noise from the environment can also result in the instability. Thus, remedies are needed to improve the performance of PID controller. One method is to average the process variable through a low-pass filter. The following equation is a common filter where β is usually between 8 to 16:

$$G'_D(S) = \frac{T_D S}{1 + \frac{T_D}{\beta} S}$$

Modifying the whole structure can also eliminate the sudden and large change in the output. Two samples are given below:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau - K_D \frac{d}{dt} y(t)$$

$$u(t) = -K_P y(t) + K_I \int_0^t e(\tau) d\tau - K_D \frac{d}{dt} y(t)$$

Furthermore, adding a non-singular median filter can also prevent the sudden change. The filter will compare the neighboring data points and select the median as process variable.

3.3 Tuning objective

The control system should meet several requirements so that we can regard it as good performance. Tuning objectives are given below:

1. stability robustness;
2. set-point following and tracking performance at transient, including rise-time, overshoot, and settling time;
3. regulation performance at steady-state, including load disturbance rejection;
4. robustness against plant modeling uncertainty;
5. noise attenuation and robustness against environmental uncertainty. (Kiam Heong Ang, Chong and Yun Li, 2005)

3.4 Tuning method

Since one term of the system may destroy the whole system, we need tuning methods to appropriately determine all the three terms. The most common manual tuning method is

Z-N method. It increases proportional term to make the system oscillate. At this time we use the proportional term as K_p and the period time as T_u . Each term can be determined by the following table according to different control type:

Control type	K_p	K_i	K_d
P	$0.5K_u$		
PI	$0.45K_u$	$0.54K_u/T_u$	
PID	$0.6K_u$	$1.2K_u/T_u$	$3K_u \cdot T_u/40$

Many tuning software have also been developed to gather the data, develop process models, and suggest optimal tuning. One of the best tuning software is called PIDeasy, which can support C++ and JAVA language and give a better result after tuning.

4. Connection and integration

In the part, most of work is practical and integration is a task orient task. So, it is not that necessary to do a lot of literature review in this part. In order to help you understand the methodology part, some definition about integration part will be provided.

The network switch and LAN will be used to set up the multi-computer communication system.

Network switch is a computer networking device that connects devices together on a computer network by using packet switching to receive, process, and forward data to the destination device. It will be used to build the multi-computer communication in hardware aspect. (Backplanes, O., & Cables, C,2014)

Local area network (LAN) is a computer network that interconnects computers within a limited area. It will be used to build the multi-computer communication in software aspect. (Backplanes, O., & Cables, C,2014)

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. The whole project is based on the system. (osrfoundation,2014)

Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic (osrfoundation,2014)

Technical content

1. Technical background

1.1 Localization

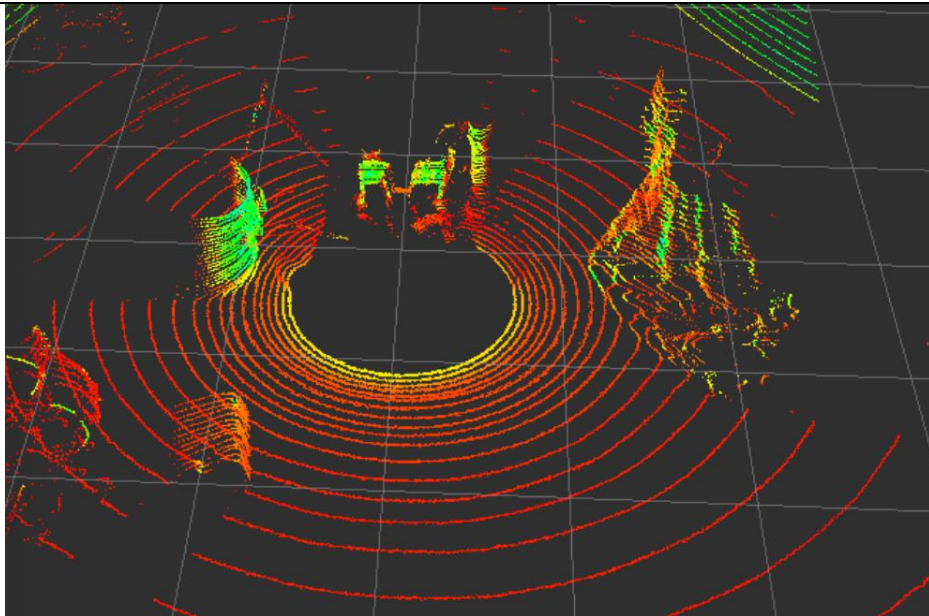
Autoware system is a powerful and updated tool comparing with other software which based on other algorithms, so we choose to achieve localization function based on the software.

First of all, some installation and building software development environment procedures will be provided. The user only needs to follow the methodology step by step. After installing relevant dependence, platform and driver. How it works and how to operate them will be introduced. Then the output data will be introduced. Finally, simulation and test procedure will be explained.

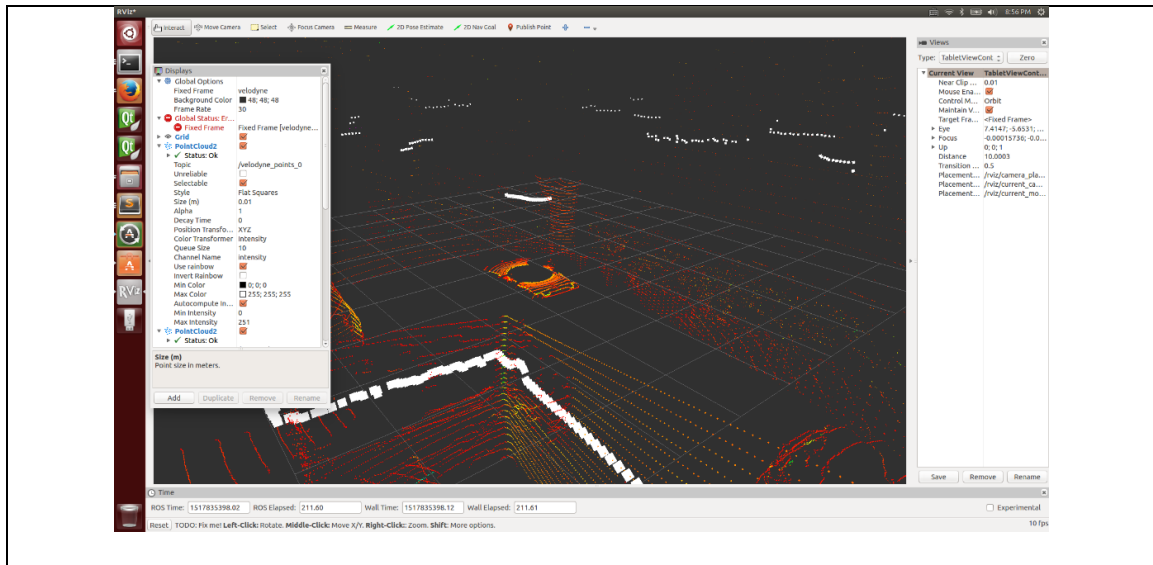
1.2 Clustering and navigation

Navigation process of the robot can be divided into 3D or 2D. 3D navigation is more complex than 2D case because 3D case needs to consider centroids at a different height, it will add the complexity to the calculation and lower the speed of data processing. As the result, we select 2D instead of 3D to be the target. In the final year project, there are two methods applied.

3D point cloud mode (sensor_msgs::pointcloud)



2D point cloud model (sensor_msgs::Laserscan)



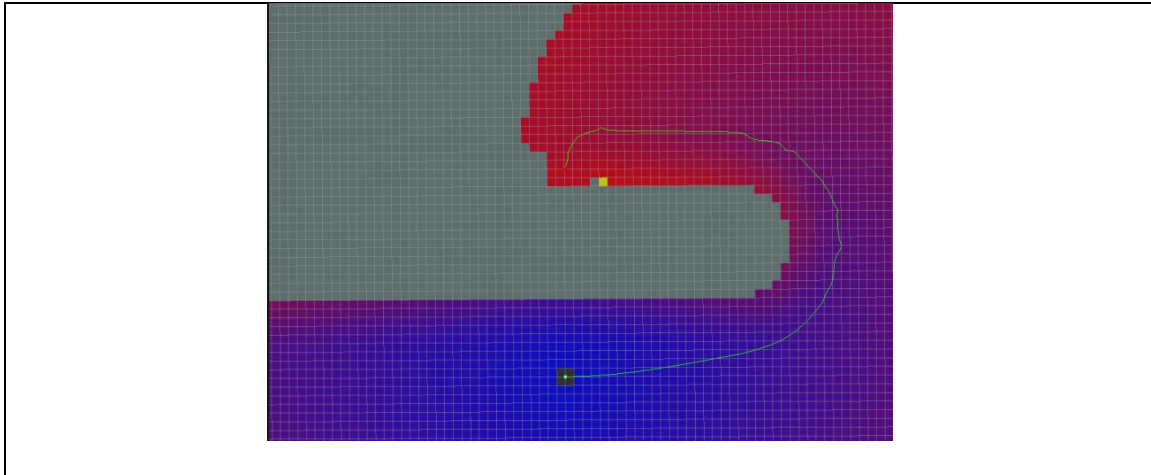
Mapping

(navs_msgs::occupancygrid)

Costmap (costmap2D)



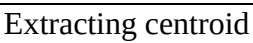
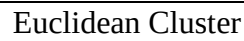
Astar (odometry)

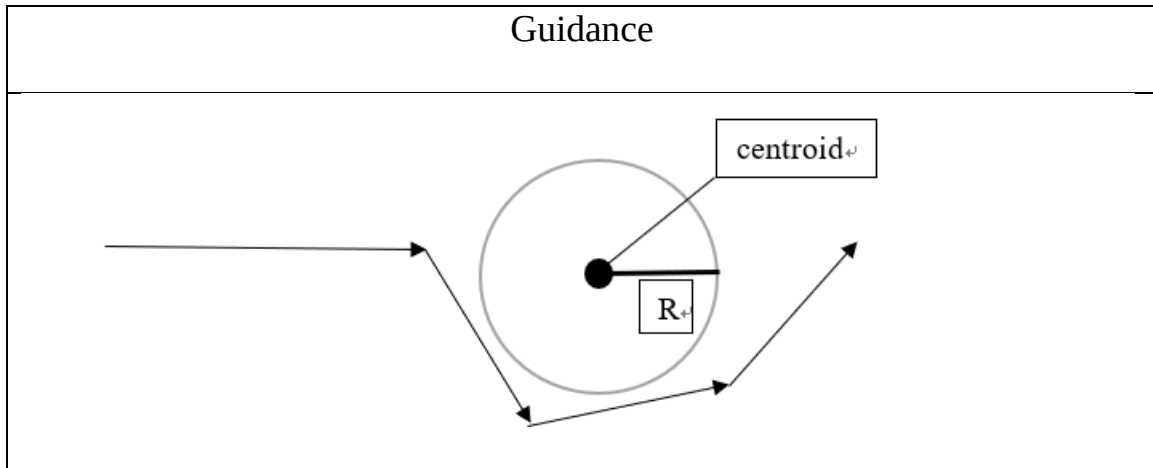


Each parenthesis represents the output message of the block. In the beginning, we transform the point cloud, the basic 3D format of the data detected from the lidar to laser scan, the 2D format of blockade detected, which is presented in white dots. Then, we transform the laser scan message to the map. Before applying A-star algorithm, the cost of the step must be one of the input messages. As the result, through the processing of star, the combination of grid map and cost map will produce the path of the shortest route that the robot follows.

Another method is simpler than the method mentioned above. The detailed process is shown.





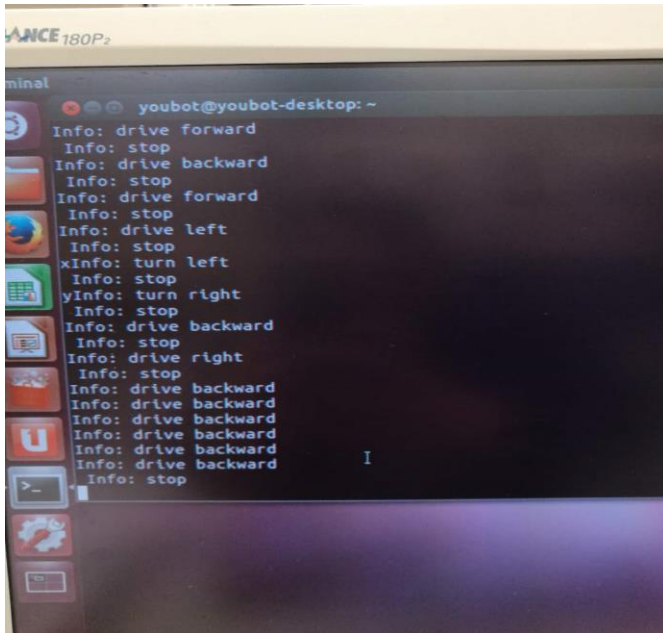


The second method focuses on the extraction of the centroid from the Euclidean cluster. Based on the principle of Euclidean cluster, each cluster is based on the distance from the centroid identified. Therefore, each cluster will contain one centroid, though the cluster will shift with the number of iteration. After the extraction of the centroid, we assume that there is a cost area with radius R . During the application, if the robot enters the cost area, the robot will shift the direction and move out from the cost region.

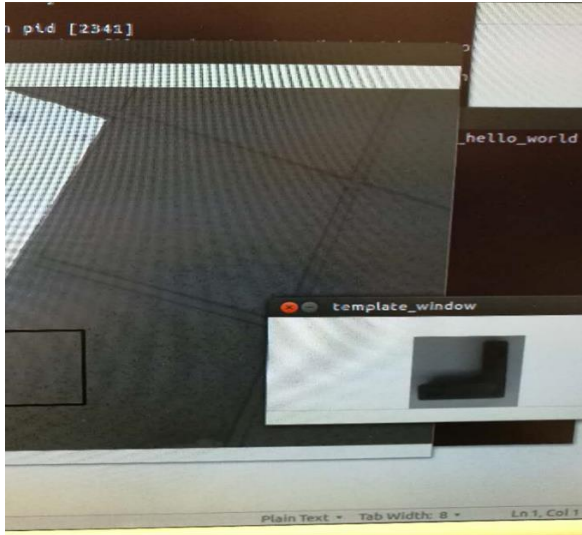
In each iteration, we have to ensure that the centroid is in the range of the map.

1.3 Path tracking and autonomous control

KUKA is a really powerful robot because in its source code, it has already programmed the controller for the base and some basic functions. We can easily use a joypad or keyboard to control its base to move vertically or horizontally. Its arm and gripper can also be controlled by a joypad or a keyboard, but we only focus on the base in this project. An example displaying command from a keyboard is shown below:



There is a camera near the gripper, which can find certain shape of materials. Since Lidar is used as the sensor, we will not further explore it. Pictures showing its function are given below:



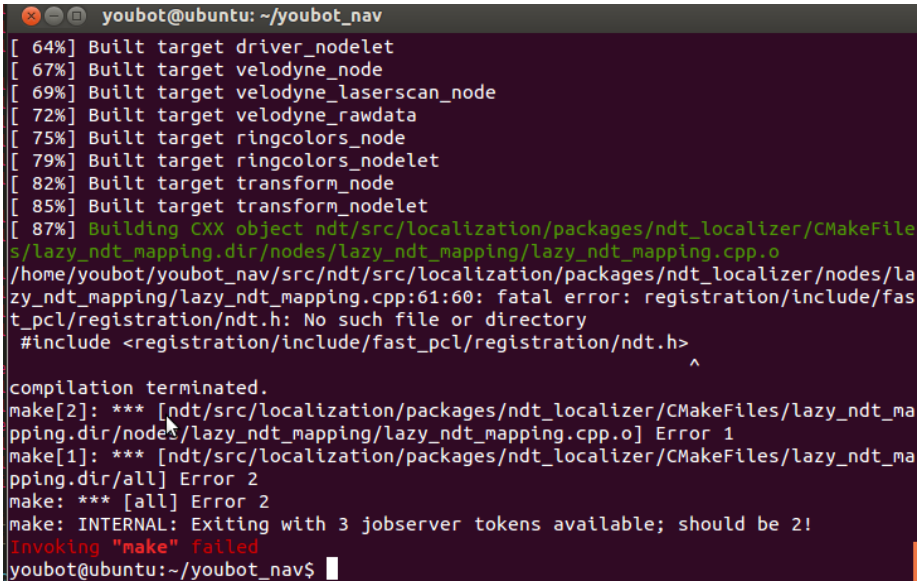
To give Lidar the biggest degree for detection and avoid blocking too much sight of Lidar, we decide to settle Lidar at the back of KUKA and drive KUKA backward, which means in our case, left is right and forward is backward. For example, `twist.linear.x = -0.05` is moving forward and `twist.linear.x = -0.05` is moving left for this project.

1.4 Integration

In initial plan, all three parts will be integrated into one computer which is installed in Kuka. So, the useful part should be picked from Autoware and be transplanted to kuka.

Then the problem occurs when the software is catkin_maked in kuka computer. Because the ubuntu's version in kuka is hydro 13.04 which is not same as my PC's version (Indigo 14.04), a lot of PCL version is not new enough to fulfill the localization requirement.

First, I try to equip and update some missing PCL files by finding them in indigo ubuntu and pasting them to kuka ubuntu. But when the situation comes to the file “ndt.h”, the file is not readable. The problem is a very deep layer question which is also related to kuka develop problems.

A terminal window titled 'youbot@ubuntu: ~/youbot_nav' showing the output of a 'make' command. The output lists several built targets: driver_nodelet, velodyne_node, velodyne_laserscan_node, velodyne_rawdata, ringcolors_node, ringcolors_nodelet, transform_node, and transform_nodelet. It then shows the building of a CXX object for lazy_ndt_mapping. The compilation fails at line 61:60 of lazy_ndt_mapping.cpp with a fatal error: 'registration/include/fast_pcl/registration/ndt.h: No such file or directory'. The error message includes a snippet of code: '#include <registration/include/fast_pcl/registration/ndt.h>'. The compilation is terminated, and the terminal shows 'make[2]: *** [ndt/src/localization/packages/ndt_localizer/CMakeFiles/lazy_ndt_mapping.dir/nodes/lazy_ndt_mapping/lazy_ndt_mapping.cpp.o] Error 1' and 'make[1]: *** [ndt/src/localization/packages/ndt_localizer/CMakeFiles/lazy_ndt_mapping.dir/all] Error 2'. Finally, it says 'make: *** [all] Error 2' and 'make: INTERNAL: Exiting with 3 jobserver tokens available; should be 2!'. The prompt 'youbot@ubuntu:~/youbot_nav\$' is visible at the bottom.

```
youbot@ubuntu: ~/youbot_nav
[ 64%] Built target driver_nodelet
[ 67%] Built target velodyne_node
[ 69%] Built target velodyne_laserscan_node
[ 72%] Built target velodyne_rawdata
[ 75%] Built target ringcolors_node
[ 79%] Built target ringcolors_nodelet
[ 82%] Built target transform_node
[ 85%] Built target transform_nodelet
[ 87%] Building CXX object ndt/src/localization/packages/ndt_localizer/CMakeFiles/lazy_ndt_mapping.dir/nodes/lazy_ndt_mapping/lazy_ndt_mapping.cpp.o
/home/youbot/youbot_nav/src/ndt/src/localization/packages/ndt_localizer/nodes/lazy_ndt_mapping/lazy_ndt_mapping.cpp:61:60: fatal error: registration/include/fast_pcl/registration/ndt.h: No such file or directory
#include <registration/include/fast_pcl/registration/ndt.h>
                                                                    ^
compilation terminated.
make[2]: *** [ndt/src/localization/packages/ndt_localizer/CMakeFiles/lazy_ndt_mapping.dir/nodes/lazy_ndt_mapping/lazy_ndt_mapping.cpp.o] Error 1
make[1]: *** [ndt/src/localization/packages/ndt_localizer/CMakeFiles/lazy_ndt_mapping.dir/all] Error 2
make: *** [all] Error 2
make: INTERNAL: Exiting with 3 jobserver tokens available; should be 2!
Invoking "make" failed
youbot@ubuntu:~/youbot_nav$
```

Meanwhile, the Autoware platform also require that the version of ubuntu should at least 14.04.

Requirements

- ROS indigo (Ubuntu 14.04) or ROS jade (Ubuntu 15.04) or ROS kinetic (Ubuntu 16.04)
- OpenCV 2.4.10 or higher
- Qt 5.2.1 or higher
- CUDA(Optional)
- FlyCapture2 (Optional)
- Armadillo (Optional)

(Autoware platform software and hardware requirement)

After sending email to kuka company and confirming that the problem is difficult to solve, an alternative plan has to be chosen to solve the communication problem between Localization, Path tracking & control, and Detection & path planning.

The rough idea is that different parts run on different computers. Topic in ROS and LAN can help achieve multi-computer connection. Comparing with integrating all parts into one kuka computer, multi-computer communication method owns these advantages:

- a. The computer in Kuka is an obsolete model and it cannot fulfill algorithm requirement, but the external computer can provide enough computing power.
- b. Indirectly solve the PCL version problem and provide the possibility to use Autoware platform which can greatly decrease the difficulty of the project.

Meanwhile, the short coming is that net wire is required to deliver information. So, kuka has to drag the net wire when someone wants to use the system.

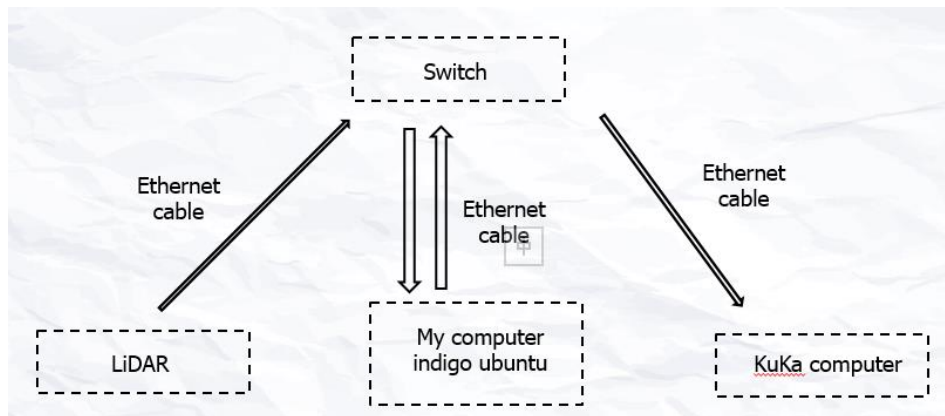
Use router and WIFI

Before considering use switch to achieve multi-computer connection, I used to try to use router and WIFI to achieve computer communication. The LAN structure needs at least two WAN. One is reserved for LiDAR and the other WAN responses to connect with Kuka. Because of the limited number of WAN and network card of my computer, I abandon the idea. As I have mentioned, the whole system needs at least two network cards and two WAN. So, I come up with the idea that I can use the switch to achieve

multi-computer connection. By using the system, I can achieve the communication requirement by taking a detour.

Use switch

Before introducing the whole system, defining the switch in advance is necessary. A network switch is a computer networking device that connects devices together on a computer network by using packet switching to receive, process, and forward data to the destination device. Meanwhile, local area network (LAN) is a computer network that interconnects computers within a limited area.



(rough structure about the multicomputer connection)

After introducing the hardware connection theory, some software theories will be introduced. According to ROS Wiki, roscore is a collection of nodes and programs that are pre-requisites of a ROS-based system. Having a roscore running is necessary to achieve ROS nodes' communication. If we see all of the parts as nodes and roscore are

responsible to the communication between these nodes, solving roscore problem cannot be avoided when the situation comes to multicomputer connection.

In order to uniform standard, the master of roscore should be selected to command all



parts in different computers. The subordinate roscore can use “export ROS_MASTER_URI=http:// ahren-Inspiron-7537:11311” code in terminal to communicate with master roscore and more information will be provided in methodology part latter.

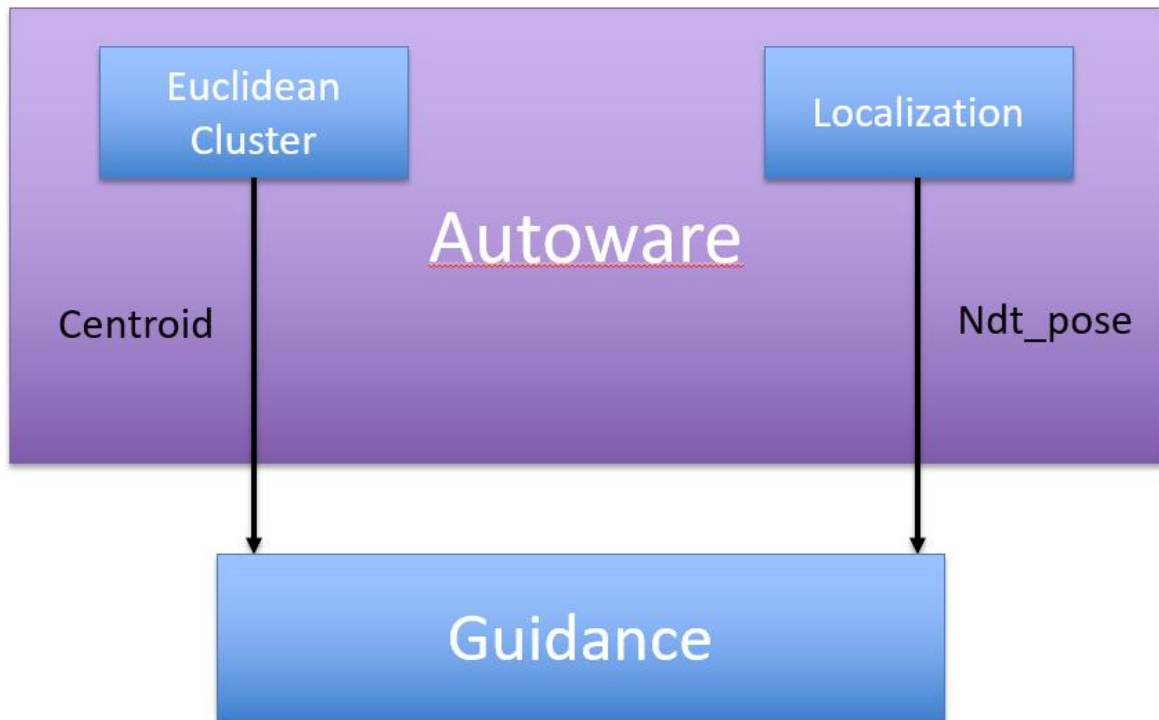
After trying many time, the sequence rules about roscore can be concluded as follow:

- a. first come first served
- b. One computer only can own one roscore in localhost.

```
NODES
roscore cannot run as another roscore/master is already running.
Please kill other roscore/master processes before relaunching.
The ROS_MASTER_URI is http://ahren-Inspiron-7537:11311/
The traceback for the exception was written to the log file
ahren@ahren-Inspiron-7537:~$
```

(it you repeat opening roscore, the error may occur)

After completion of part of ndt mapping, which is used for localization, and Euler clustering, which is used for object classification, the combination of both function is achieved by placing those functions in Autoware, the platform used for autonomous driving. The basic flow chart is shown as follows:



In the part of path tracking and control, we obtain current location from localization and centroid from Euclidean cluster. Then, we build up the cost area with radius R that encircles the centroid obtained from the cluster. If the robot moves into the area of inclusion, the robot will shift from the preset route and move out from the cost area.

In the combination, the path tracking file receives location information and centroid of blockade simultaneously. The function will publish the current location of robot in parameter x and y , the velocity of the robot in parameter v_x and v_y and the block centroid

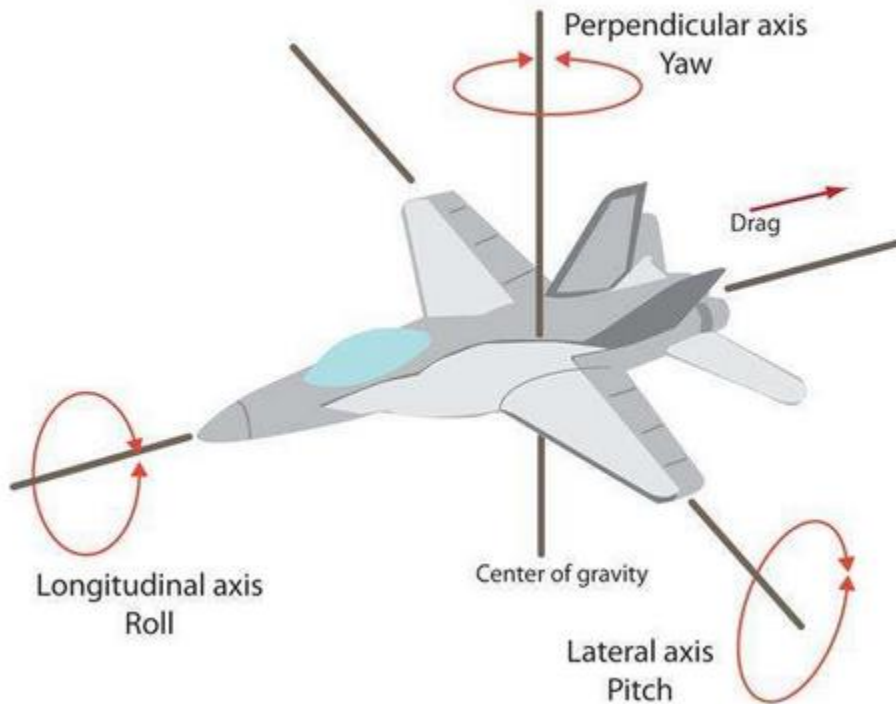
presented in parameter `block_x` and `block_y`. In the function, only coordinate `x` and `y` are required.

2. Methodology

2.1 Localization

2.1.1 Relevant code explanation

Because most of the code about localization was found in Autoware platform and it is really a complex system, I only pick some typical function and explain it to give you a rough idea about localization.



Before explaining the code, some theories about coordinate should be introduced: x , y , z , roll, yaw, and pitch. Because x , y , z is easy to understand, they will be skipped. Roll, yaw, and pitch are three angles to describe the orientation of a rigid body with respect to a fixed coordinate system. (Imperatorskai, 2012) They can also represent the orientation

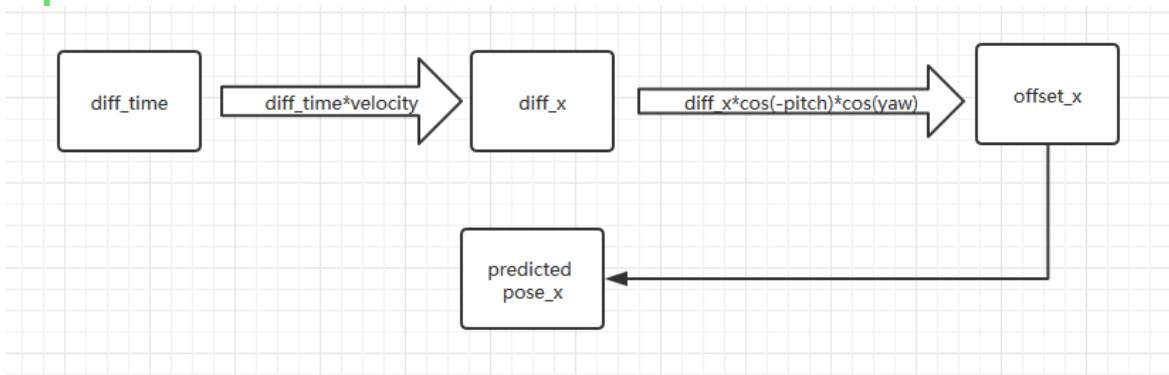
of a mobile frame of reference in physics or the orientation of a general basis in 3-dimensional linear algebra

```
1. static void imu_odom_calc (ros::Time current_time)
2. {
3.     static ros::Time previous_time = current_time;
4.     double diff_time = (current_time - previous_time).toSec();
5.
6.     double diff_imu_roll = imu.angular_velocity.x * diff_time;
7.     double diff_imu_pitch = imu.angular_velocity.y * diff_time;
8.     double diff_imu_yaw = imu.angular_velocity.z * diff_time;
9.
10.    current_pose_imu_odom.roll += diff_imu_roll;
11.    current_pose_imu_odom.pitch += diff_imu_pitch;
12.    current_pose_imu_odom.yaw += diff_imu_yaw;
13.
14.    double diff_distance = odom.twist.twist.linear.x * diff_time;
15.    offset_imu_odom_x += diff_distance*cos(-
        current_pose_imu_odom.pitch)*cos(current_pose_imu_odom.yaw);
16.    offset_imu_odom_y += diff_distance*cos(-
        current_pose_imu_odom.pitch)*sin(current_pose_imu_odom.yaw);
17.    offset_imu_odom_z += diff_distance*sin(-current_pose_imu_odom.pitch);
18.
19.    offset_imu_odom_roll += diff_imu_roll;
```

```

20. offset_imu_odom_pitch += diff_imu_pitch;
21. offset_imu_odom_yaw  += diff_imu_yaw;
22.
23. predict_pose_imu_odom.x  = previous_pose.x  + offset_imu_odom_x;
24. predict_pose_imu_odom.y  = previous_pose.y  + offset_imu_odom_y;
25. predict_pose_imu_odom.z  = previous_pose.z  + offset_imu_odom_z;
26. predict_pose_imu_odom.roll = previous_pose.roll + offset_imu_odom_roll;
27. predict_pose_imu_odom.pitch = previous_pose.pitch + offset_imu_odom_pitch;
28. predict_pose_imu_odom.yaw  = previous_pose.yaw  + offset_imu_odom_yaw;
29.
30. previous_time = current_time;
31. }
32.

```



Because of the information deliver delay and computer calculation time, the measured pose is not current pose. Because of the limitation of computer power and hardware, the measurement is difficult to fulfill the requirement. According to probability robotics, predicting its pose might be a reasonable method to solve the inaccurate problem.

(Thrun, 2005)

The sample can be used if it both has imu and Odom input. Depending on the scanning frequency, the `diff_time` can be determined. By multiplying with velocity, we can get the `diff_distance` which is a scalar. In order to transform the scalar to vector, the trigonometric function will be used to calculate the distance in x, y, z-direction. Because our experiment surface can seem like a plane, we do not need to consider pitch and roll. Then the value of distance x can be assigned to `offset_x`.

Because the velocity cannot be suddenly changed, if the frequency is very high, the prediction of the pose can be achieved by considering velocity as an offset.

```
1. predict_pose_imu_odom.x = previous_pose.x + offset_imu_odom_x;
```

2.1.2 Equipment driver installation and operation

a. Install the LiDAR driver

```
/home/ahren/15_velodyne/src/velodyne/velodyne_pointcloud/launch/32e_points.launch h
ahren@ahren-Inspiron-7537:~$ cd 15_velodyne/
ahren@ahren-Inspiron-7537:~/15_velodyne$ ./runVelodyne.sh
... logging to /home/ahren/.ros/log/ac3b305c-3710-11e8-b856-ef87451b8a82/roslaun
ch-ahren-Inspiron-7537-2109.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ahren-Inspiron-7537:34252/

SUMMARY
=====

PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.21
* /velodyne_nodelet_manager_cloud/calibration: /home/ahren/15_ve...
* /velodyne_nodelet_manager_cloud/max_range: 130.0
* /velodyne_nodelet_manager_cloud/min_range: 0.4
* /velodyne_nodelet_manager_driver/device_ip:
* /velodyne_nodelet_manager_driver/frame_id: velodyne
* /velodyne_nodelet_manager_driver/model: 32E
* /velodyne_nodelet_manager_driver/pcap:
* /velodyne_nodelet_manager_driver/port: 2368
* /velodyne_nodelet_manager_driver/read_fast: False
* /velodyne_nodelet_manager_driver/read_once: False
* /velodyne_nodelet_manager_driver/repeat_delay: 0.0
* /velodyne_nodelet_manager_driver/rpm: 600.0

NODES
/
  rviz (rviz/rviz)
  velodyne_nodelet_manager (nodelet/nodelet)
  velodyne_nodelet_manager_cloud (nodelet/nodelet)
  velodyne_nodelet_manager_driver (nodelet/nodelet)

auto-starting new master
process[master]: started with pid [2164]
ROS_MASTER_URI=http://localhost:11311
```

Because LiDAR is very complex equipment, it need a driver to deliver data to PC

```
$ cd 15_velodyne/
```

```
$ ./runVelodyne.sh
```

b. Autoware platform installation

In the terminal, you can follow the following code to install the Autoware in Github and run it.

```
1. $ cd $HOME
```

```
2. $ git clone https://github.com/CPFL/Autoware.git
3. $ cd ~/Autoware/ros/src
4. $ catkin_init_workspace
5. $ cd ../
6. $ ./catkin_make_release
7. $ cd $HOME/Autoware/ros
8. $ ./run
```

c. Procedure to generate map and SLAM

It is necessary to use LiDAR to scan the environment and generate bag file.

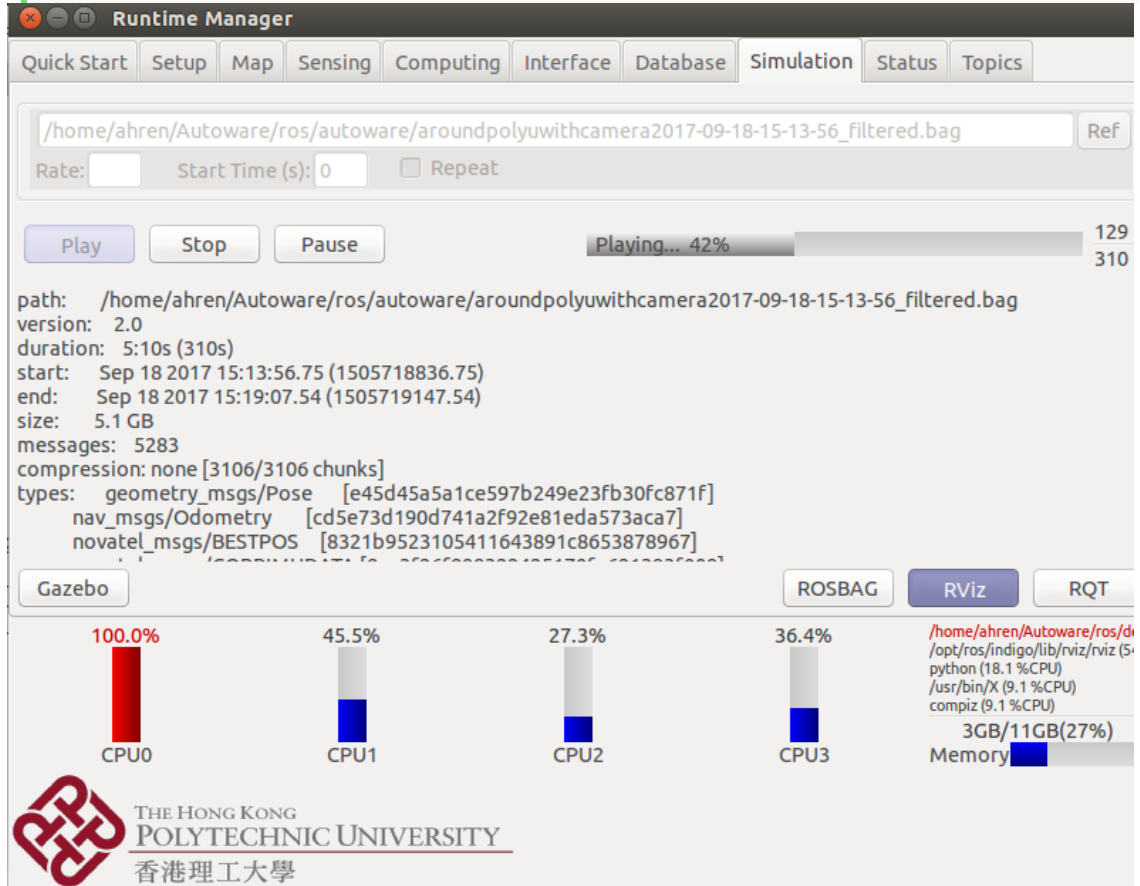
- i. Carry the kuka robot to barrow and install the LiDAR on kuka.
- ii. Follow LiDAR driver procedure to let all equipment standby.
- iii. After opening the LiDAR and test the connection between PC and LiDAR.
- iv. Move the kuka scan the environment of FJ005 for about 3 min. (Because scanning time influences the size of. bag file)

After getting the bag file, the pcd mapping can be formed by using ndt_mapping.

First, the Autoware should be opened.

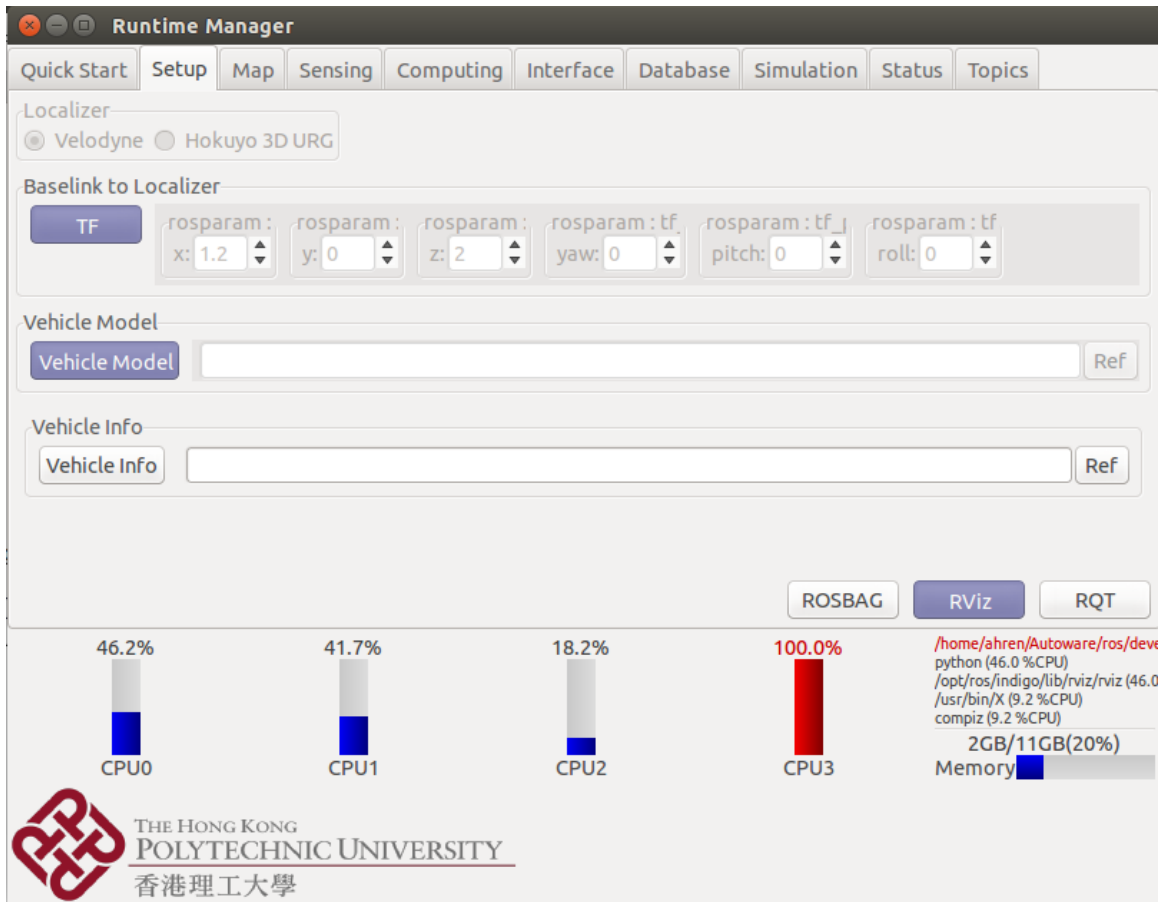
```
1. $ cd $HOME/Autoware/ros
```

2. \$./run



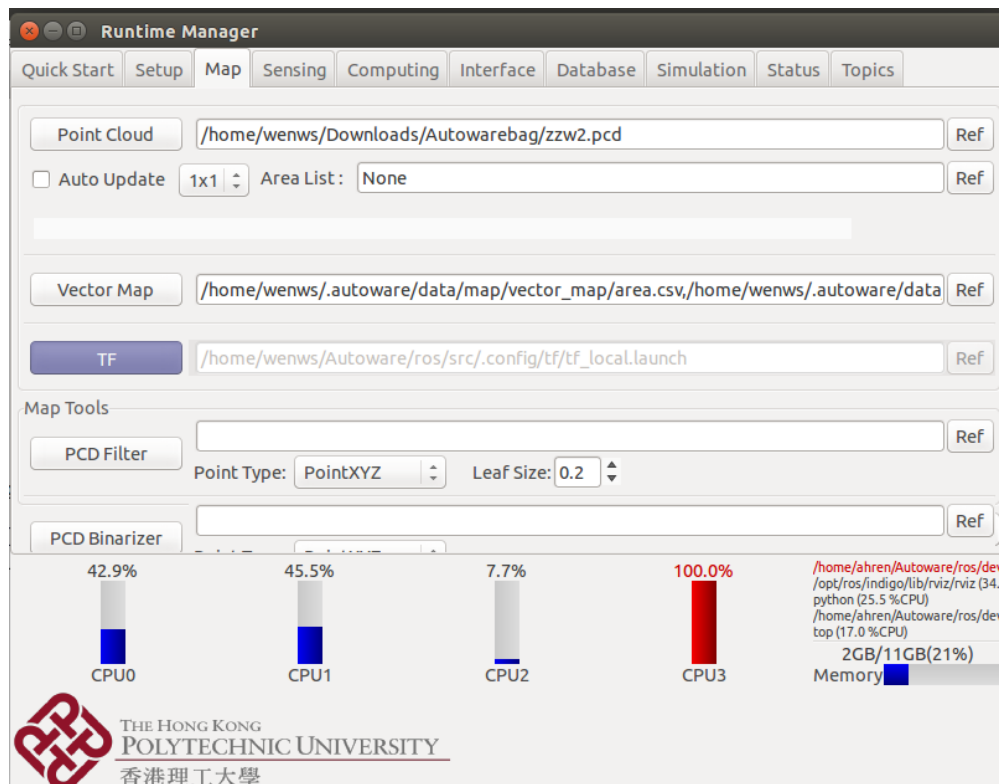
Then, click the “ref” button and find “. Bag” file. Before starting choosing function, the bag file should standby by clicking “play” and “pause”.

If the computer is directly connected with LiDAR, the data will be sent to Autoware automatically.

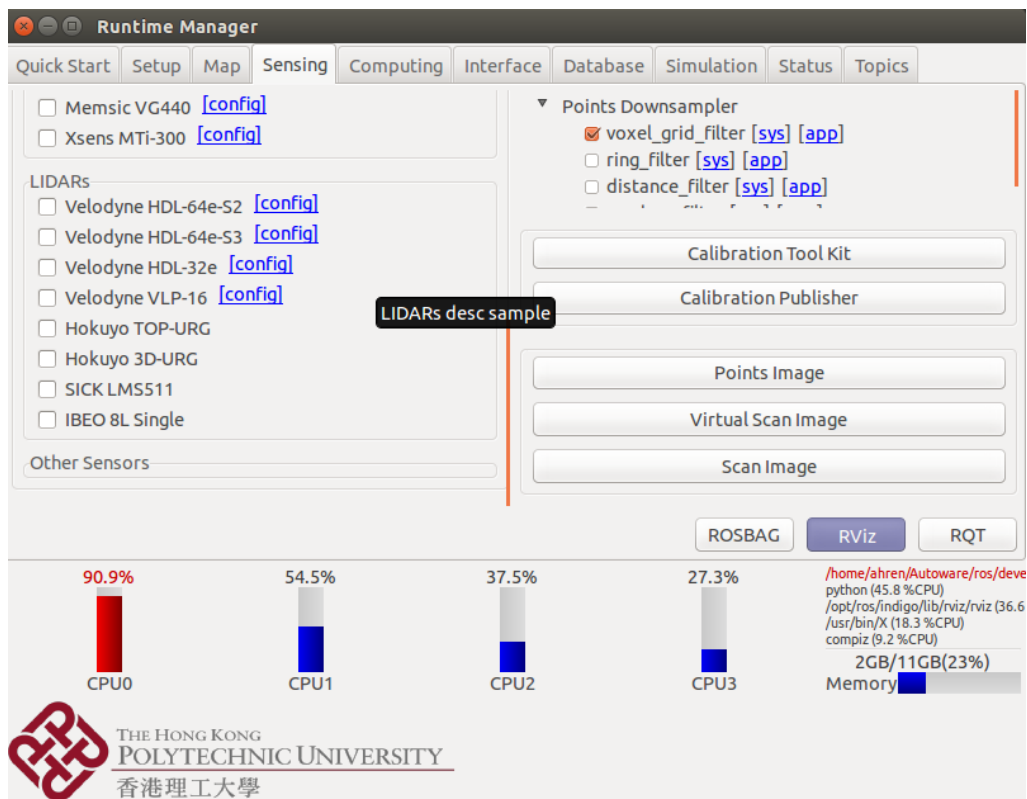


Setup part: select the Baselink TF and Vehicle Model.

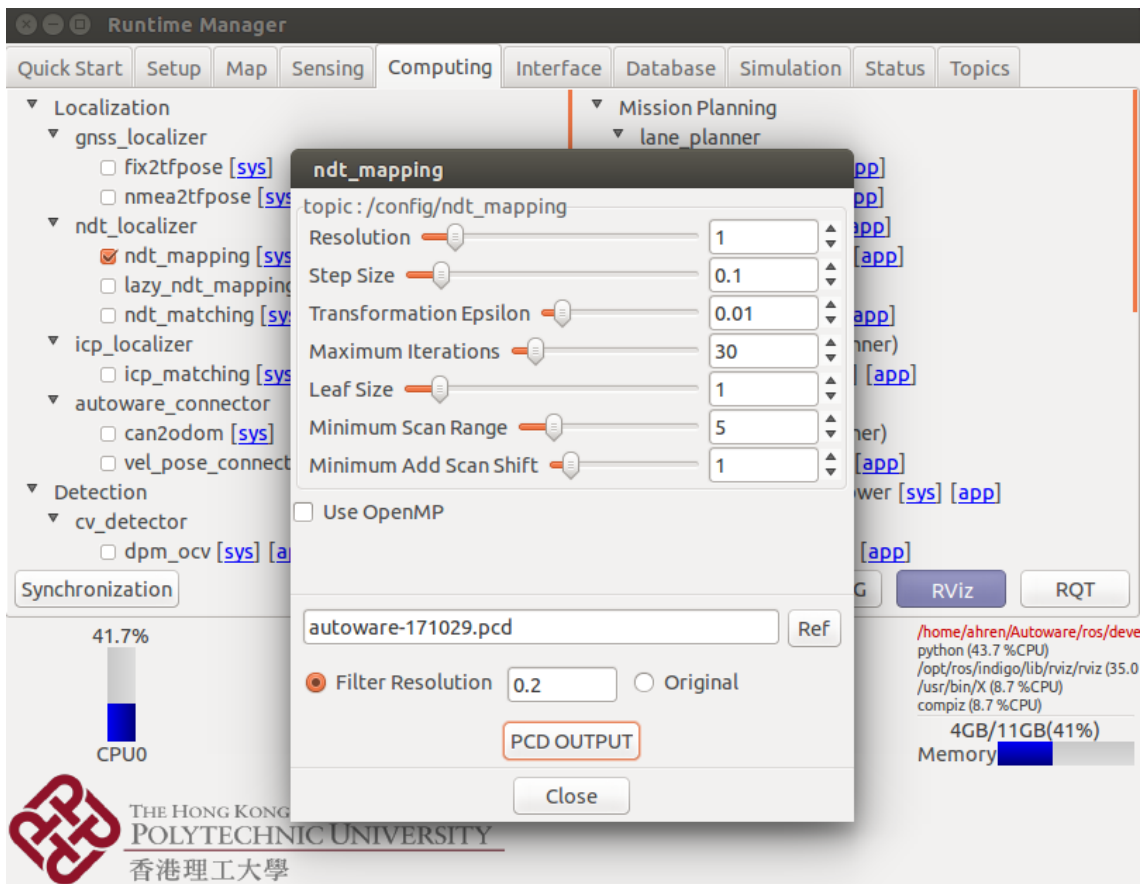
The baselink is used to correct the localization because the localization data which is sent by LiDAR is LiDAR's coordinate point instead of the point of robot.



TF: Local launch is used in mapping & Localization launch is used in matching because we need different transformations to fit different topics.

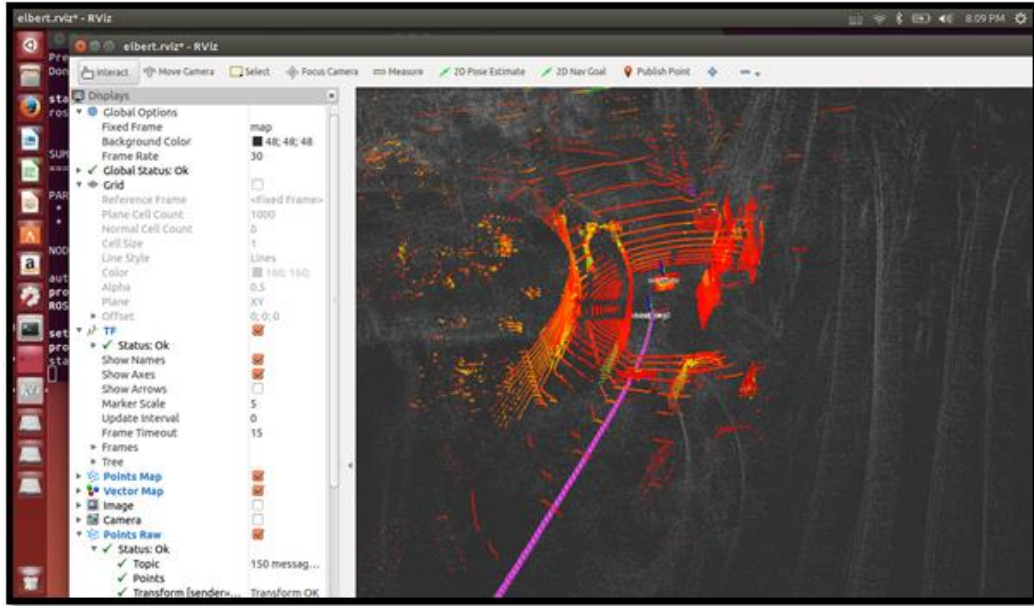


Sensing part: Select Voxel_grid_filter. Voxel_grid_filter is mainly responsible to select the useful data and get rid of the calculation load.

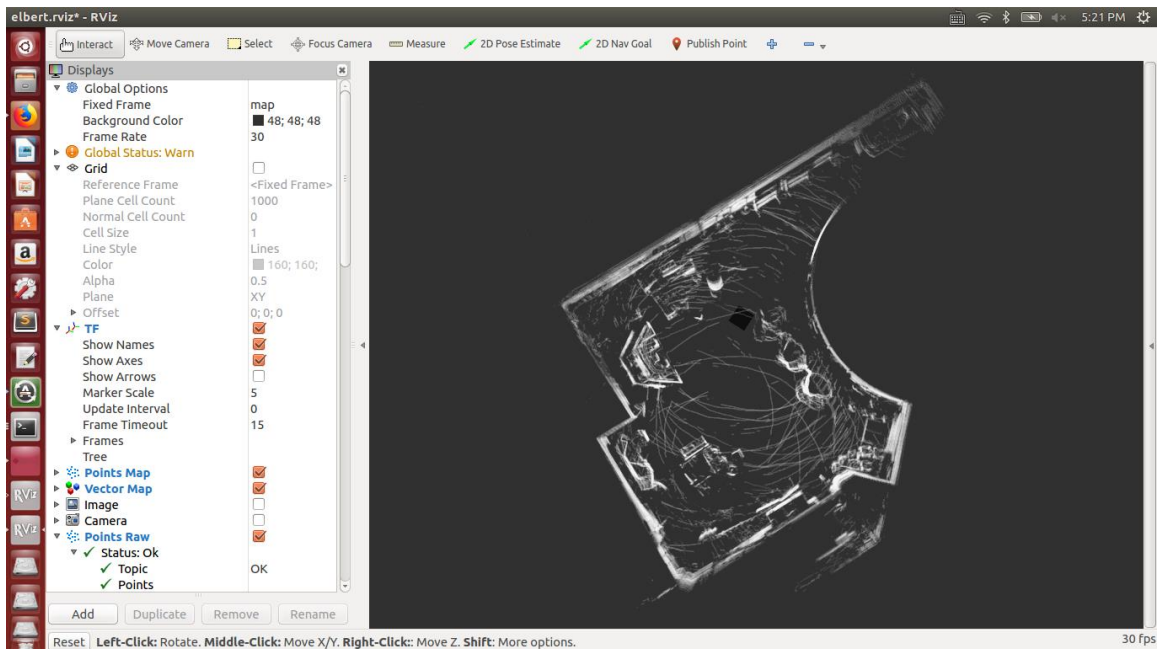


Computing part: NDT_mapping is used to generate a point cloud map (pcd file).

Depending on the map accuracy requirement, following factors can be edited to control the quantity of information content.



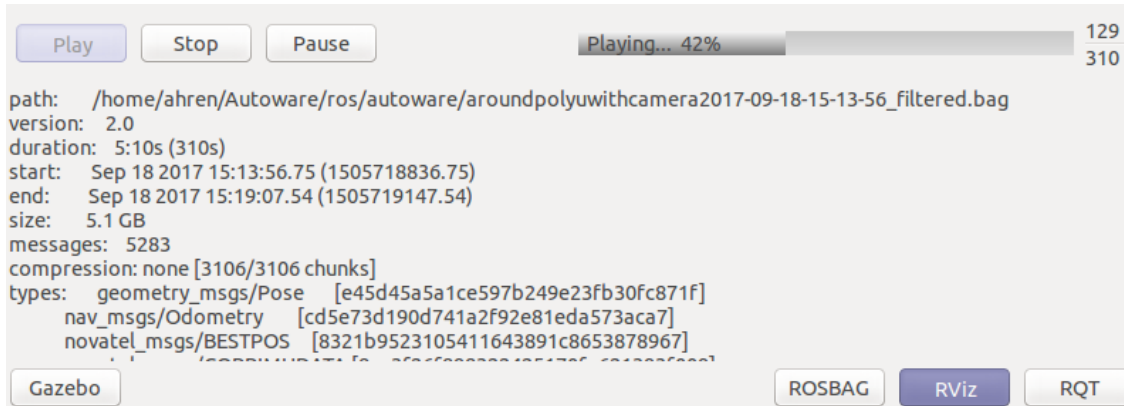
Here is the screenshot of Rviz. The red concentric circle means that the LiDAR is scanning the environment data and the grey structure means the pcd (3D map) that is generated before. The pink line is the path line of the Vehicle Model.



(figure a the pcd mapping of FJ005 generated by ourselves.)

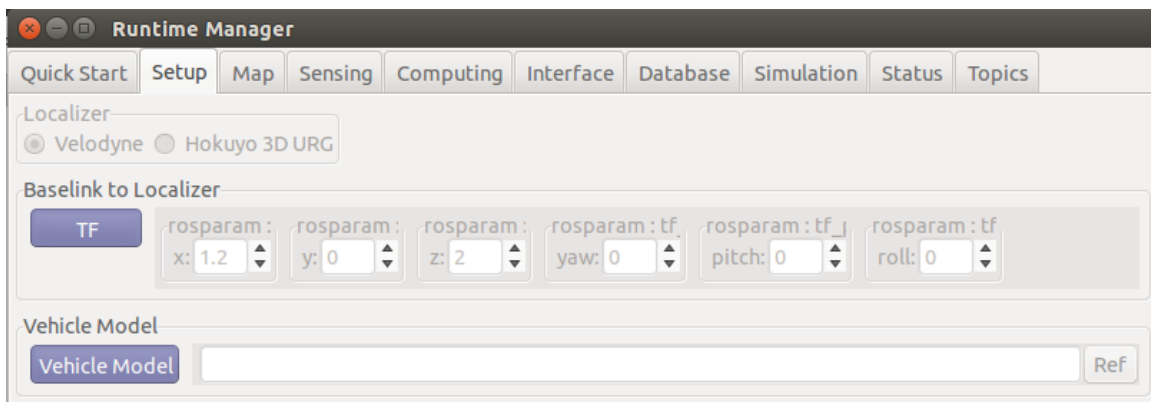
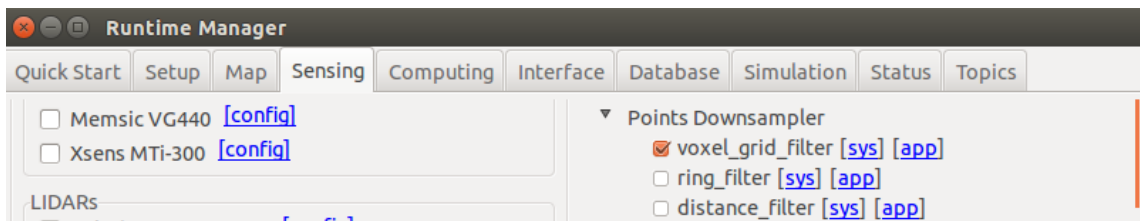
d. Localization

localization procedure is roughly similar to mapping generation.

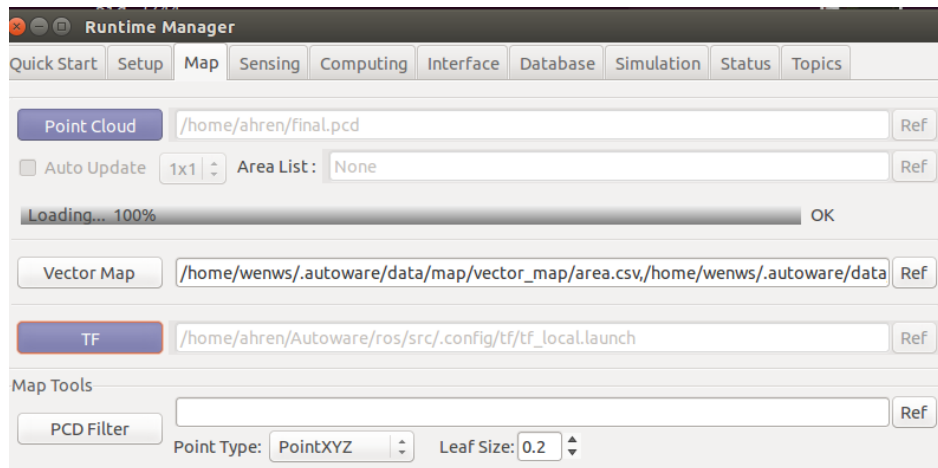


Simulation part: Ref to the bag file that generated by the LiDAR. If the computer is directly connected with LiDAR, the data will be sent to Autoware automatically.

Setup part: select the Baselink TF and Vehicle Model.



Sensing part: Select Voxel_grid_filter.

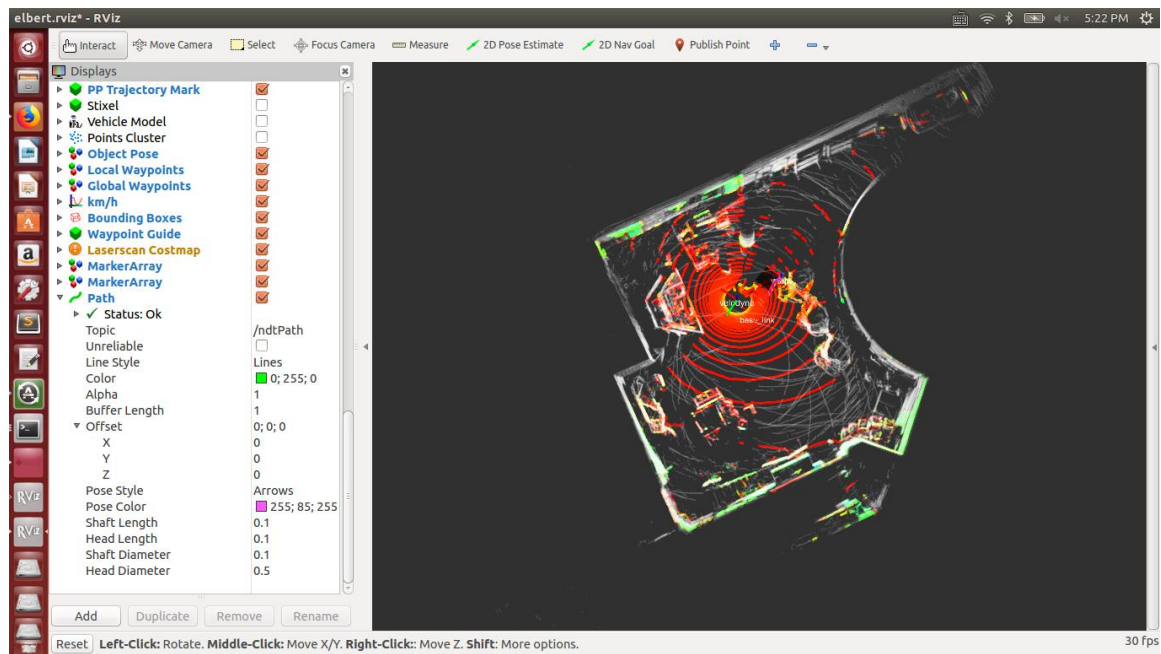


Map part: Point Cloud refers to the pcd file (the grey structure) and it can be generated by using NDT_mapping

TF: Local launch is used in mapping .Localization launch is used in matching because we need different transforms to fit different topics.

Computing part:

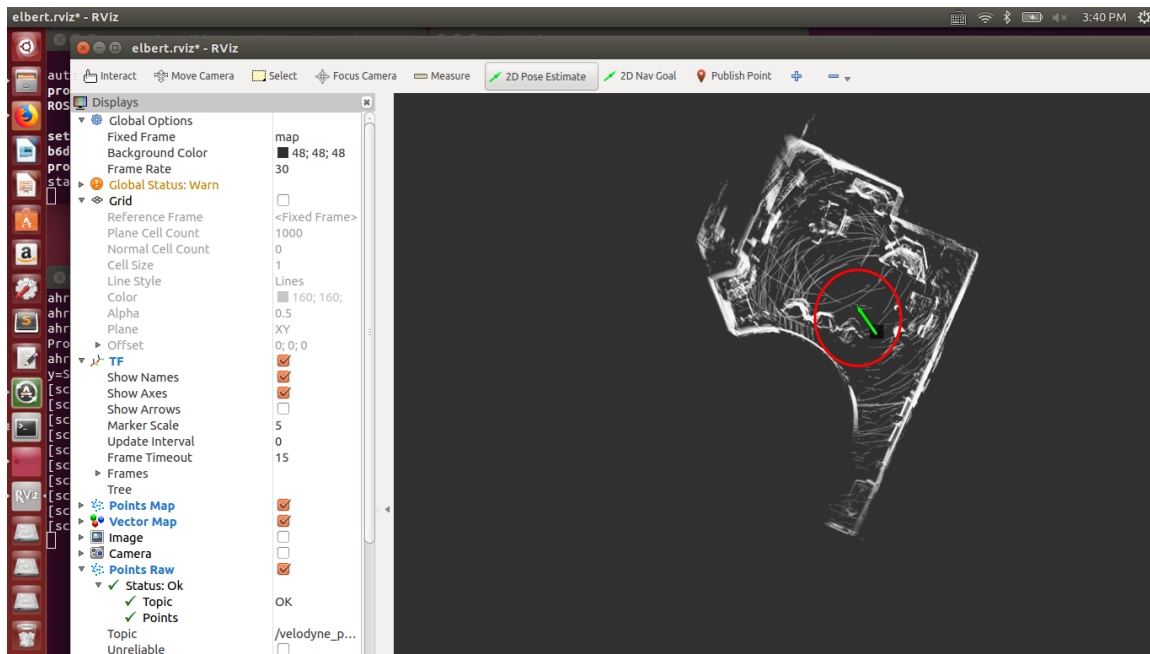
NDT_matching can achieve the localization



e. coordinate selection

Because all of the systems share one coordinate, select the coordinate is important. First pressure the 2D pose green arrow. Then ,drop the mouse to the positive direction.

It can be observed in sample pcd map. (the black rectangular area without any particle density.)

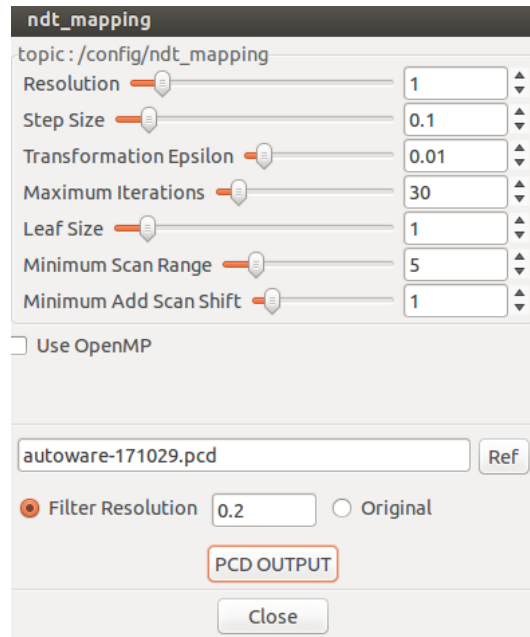


f. Further improvement

Because Autoware platform is a complex system and it is difficult to change the inside code. Luckily, the system provides some changeable factor for us and make the system more flexible. According to our design requirement, these factors will be changed to fulfill the specific environment requirement.

- i. Ndt_mapping: user can change resolution, step size. transformation epsilon, maximum iteration, leaf size. minimum scan range, minimum add scan shift and filter resolution. By changing these factor, some characters of pcd map can be changed such as changing the pcd file's size by decrease the filter resolution. Because

the project is in an in-door environment and our computer is not that advance, the accuracy requirement is not that high. So, the filter resolution is reduced to 0.2.



ii. Ndt_matching

In the part, the initial pose can be choosed. Error threshold, resolution, step size and transformation epsilon and maximum interation. According to PC calculation power, these factors can be changed to change the accuracy of SLAM. In this way, the Autoware system can be changed to match with the computer power.

ndt

topic: /config/ndt

☒ Initial Pos x: 0 y: 0 z: 0 roll: 0 pitch: 0 yaw: 0

☐ GNSS

Predict Pose

☐ OFF ☒ ON

Error Threshold 1

Resolution 1

Step Size 0.01

Transformation Epsilon 0.01

Maximum Iterations 30

☐ Use OpenMP ☐ Get Height

OK Cancel

2.3 Clustering and navigation

2.2.1 Cost mapping method

2.2.1.1 Conversion from 3D to 2D point cloud

A.PassThrough Filter

As mentioned above, before the application of the algorithm, the preprocessing of the cloud is required. In general, the map is stored in the pcd file. Each point in the pcd files is stored in the struct `pcl::PointXYZ`.

In the format struct `pcl::PointXYZ`, the 3D information is stored.

```
Pcl::PointXYZ::PointXYZ (  
    float _x,  
    float _y,  
    float _z,  
) inline
```

We load the PCD file created in the localization process, named `fj005`. The input cloud is read. The value is given into the pointer cloud and the prefiltered points are shown.

```

01. #include <iostream>
02. #include <pcl/point_types.h>
03. #include <pcl/io/pcd_io.h>
04. #include <pcl/filters/passthrough.h>
05.
06. int
07. main (int argc, char** argv)
08. {
09.     pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);
10.     pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_filtered (new pcl::PointCloud<pcl::PointXYZ>);
11.
12.     pcl::io::loadPCDFile<pcl::PointXYZ> ("parklot.pcd", *cloud);
13.
14.     std::cerr << "Cloud before filtering: " << std::endl;
15.     for (size_t i = 0; i < cloud->points.size (); ++i)
16.         std::cerr << "      " << cloud->points[i].x << " "
17.                     << cloud->points[i].y << " "
18.                     << cloud->points[i].z << std::endl;

```

Then, we apply the filter. We target the axis for the filtering and the range of the axis that is acceptable.

Last ,we store the filtered data into the new pcd in the format pcl::pointxyz.

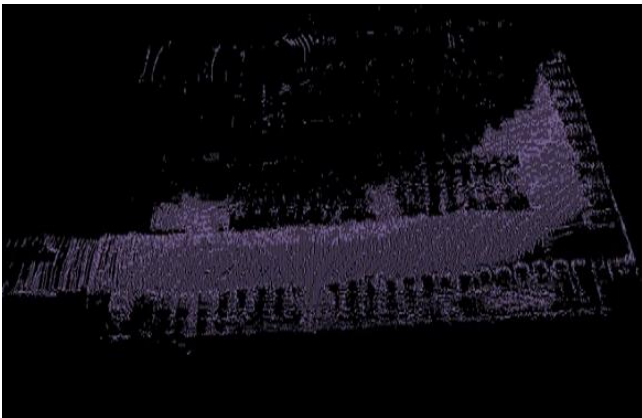
```

01. // Create the filtering object
02. pcl::PassThrough<pcl::PointXYZ> pass;
03. pass.setInputCloud (cloud);
04. pass.setFilterFieldName ("z");
05. pass.setFilterLimits(0.0,0.3);
06. //pass.setFilterLimitsNegative (true);
07. pass.filter (*cloud_filtered);
08.
09. std::cerr << "Cloud after filtering: " << std::endl;
10. for (size_t i = 0; i < cloud_filtered->points.size (); ++i)
11.     std::cerr << "      " << cloud_filtered->points[i].x << " "
12.                 << cloud_filtered->points[i].y << " "
13.                 << cloud_filtered->points[i].z << std::endl;
14. pcl::io::savePCDFileASCII ("parklot_f.pcd", *cloud_filtered);
15. return (0);
16. }

```

In the function, setinputcloud receive the data from the pointer cloud, which points to the

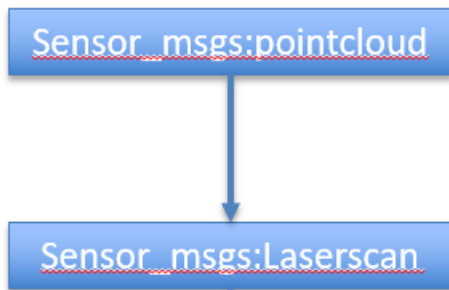
data in pcd file. Setfilterfieldname and setfilterlimits set the targeted axis of filtration and the range for the filtration. In this case, the range is from 0 to 0.3. The outcome is shown as follows:



The outcome of the new pcd file is approximate to the flat plane. However, it is still not the actual 2D file we need in the construction of the map. The data is stored in the `pcl::PointXYZ`. Therefore, we need another way to complete the target.

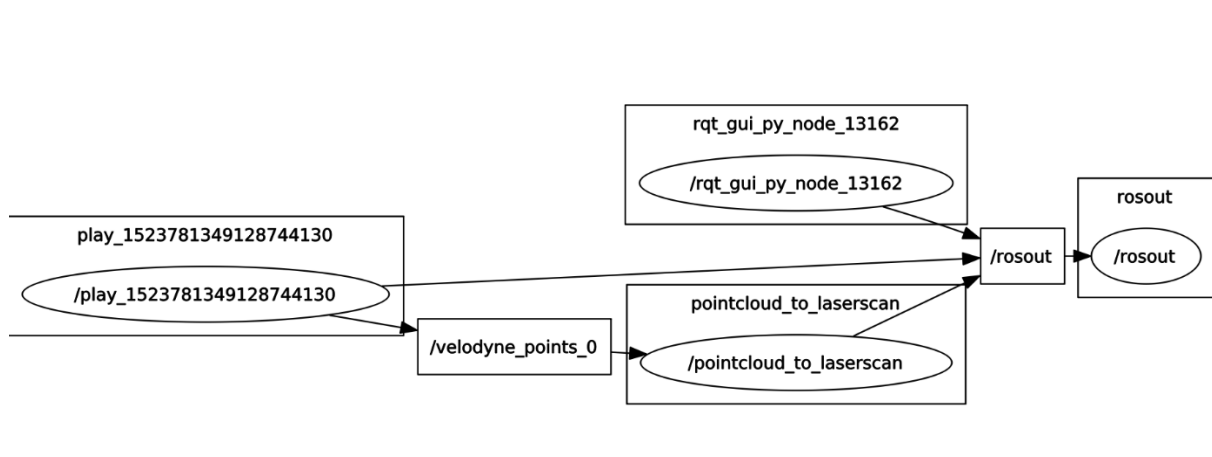
B. Laserscan

Another way to complete the job is to transform the bag of pointcloud to laserscan. The format transformation is shown as follows:

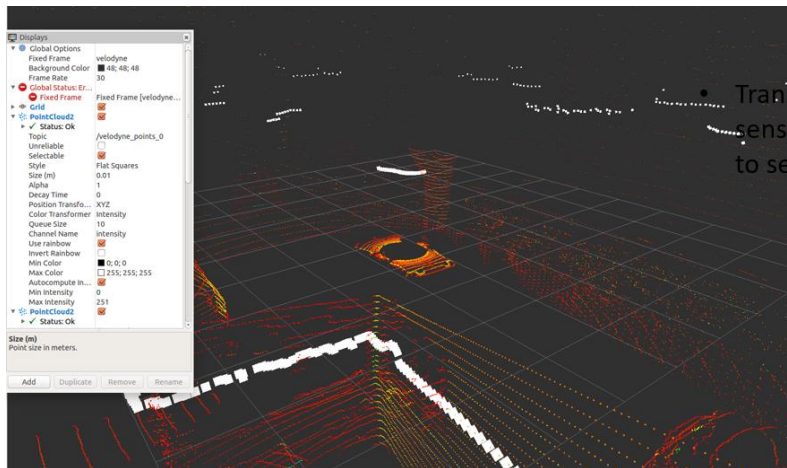


Sensor_msgs::Laserscan is the format that includes the information of the range of scanning between preset minimum angle and maximum angle, scanning time between measurement and angle increment. These parameters can be retrieved from the server.

The overall flow chart of the scan is shown as follows:



When the bag of pointcloud is played, the points will be published to the function pointcloud_to_laserscan and the final message is the laserscan in 2D mode. The final work is shown as follows:



The function of transformation from 3D to 2D is posted on appendix.

2.2.2 Clustering Method

2.2.2.1 Euclidean Cluster

A. Basic code explanation:

The coding of the Euclidean cluster is complex to understand. Therefore, I prefer to explain part of the code that is crucial to the analysis.

In the clustering process, performer sets the minimum clustering size and maximum clustering size of each cluster. In addition, the maximum range of clustering is specified. Each cluster occupies specific range. The input cloud is set in the 2D form.

```
1. std::vector<pcl::PointIndices> cluster_indices;
2.
3.     //perform clustering on 2d cloud
4.     pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;
5.     ec.setClusterTolerance (in_max_cluster_distance); //parameters of clustering
        are set.
6.     ec.setMinClusterSize (_cluster_size_min);
7.     ec.setMaxClusterSize (_cluster_size_max);
8.     ec.setSearchMethod(tree);
9.     ec.setInputCloud (cloud_2d);
10.    ec.extract (cluster_indices);
```

The merging process of euclidean cluster is strongly recommended. If there are numerous clusters, the speed of calculation will be reduced. Therefore, the assigned value of threshold will be set. If the distance between centroids is less than threshold value, clusters will merge. Threshold is the mean distance of the points in the cluster.

```
1. void mergeClusters(const std::vector<ClusterPtr>& in_clusters, std::vector<ClusterPtr>& out_clusters, std::vector<size_t> in_merge_indices, const size_t& current_index, std::vector<bool>& in_out_merged_clusters)
2. {
```

```

3.      //std::cout << "mergeClusters:" << in_merge_indices.size() << std::endl;
4.      pcl::PointCloud<pcl::PointXYZRGB> sum_cloud;
5.      pcl::PointCloud<pcl::PointXYZ> mono_cloud;
6.      ClusterPtr merged_cluster(new Cluster());
7.      for (size_t i=0; i<in_merge_indices.size(); i++)
8.      {
9.          sum_cloud += *(in_clusters[in_merge_indices[i]]->GetCloud());
10.         in_out_merged_clusters[in_merge_indices[i]] = true;
11.     }
12.     std::vector<int> indices(sum_cloud.points.size(), 0);
13.     for (size_t i=0; i<sum_cloud.points.size(); i++)
14.     {
15.         indices[i]=i;
16.     }
17.
18.     if (sum_cloud.points.size() > 0)
19.     {
20.         pcl::copyPointCloud(sum_cloud, mono_cloud);
21.         //std::cout << "mergedClusters " << sum_cloud.points.size() << " mono:"
22.         << mono_cloud.points.size() << std::endl;
23.         //cluster->SetCloud(in_cloud_ptr, it->indices, _velodyne_header, k, (int)
24.         _colors[k].val[0], (int)_colors[k].val[1], (int)_colors[k].val[2], "", _pose_es
25.         timation);
26.         merged_cluster->SetCloud(mono_cloud.makeShared(), indices, _velodyne_header,
27.         current_index, (int)_colors[current_index].val[0], (int)_colors[current_index].val[1],
28.         (int)_colors[current_index].val[2], "", _pose_estimation);
29.         out_clusters.push_back(merged_cluster);
30.     }
31. }

```

In this case, the researcher sets different thresholds for different sizes of clusters. For the clustering process, points that are distant from the clusters will have larger threshold value.

Cluster_array (n)	Distance from Robot (m)	Threshold (m)
0	0-15	0.5
1	15-30	1
2	30-45	1.6
3	45-60	2.1
4	>60	2.6

```
1. for(unsigned int i=0; i<cloud_segments_array.size(); i++)
2.     {
3.         pcl::PointCloud<pcl::PointXYZ>::Ptr tmp_cloud(new pcl::PointCloud<pcl::PointXYZ>);
4.         cloud_segments_array[i] = tmp_cloud;
5.     }
6.
7.     for (unsigned int i=0; i<in_cloud_ptr->points.size(); i++)
8.     {
9.         pcl::PointXYZ current_point;
10.        current_point.x = in_cloud_ptr->points[i].x;
11.        current_point.y = in_cloud_ptr->points[i].y;
12.        current_point.z = in_cloud_ptr->points[i].z;
13.
```

```

14.         float origin_distance = sqrt( pow(current_point.x,2) + pow(current_point
        .y,2) );
15.
16.         if      (origin_distance < _clustering_distances[0] )      {cloud_segments
        _array[0]->points.push_back (current_point);}
17.         else if(origin_distance < _clustering_distances[1])      {cloud_segments
        _array[1]->points.push_back (current_point);}
18.         else if(origin_distance < _clustering_distances[2])      {cloud_segments
        _array[2]->points.push_back (current_point);}
19.         else if(origin_distance < _clustering_distances[3])      {cloud_segments
        _array[3]->points.push_back (current_point);}
20.         else                                                    {cloud_segments_
        array[4]->points.push_back (current_point);}
21.     }

```

The clustered points will be classified into four main categories based on the distance specified.

B.Euler Clustering

Before applying the euler cluster, some settings need to be set up.



Pcd is the map we need. We also are aware that the transfer function is necessary. We have to transform the data collected from the world to map. The world is the surrounding environment. The map is the map we want to build. The transfer function can transform the coordinate of point cloud from the world to the coordinate of point cloud from the map. Transfer functions are shown below.

tf_lidar_localization

```
<launch>
  <node pkg="tf" type="static_transform_publisher" name="world_to_map" args="0 0 0 0 0 /world /map 100" />
  <node pkg="tf" type="static_transform_publisher" name="odom_to_map" args="0 0 0 0 0 /map /odom 100" />
  <node pkg="tf" type="static_transform_publisher" name="velodyne_to_base" args="0 0 0 0 0 /base_link /velodyne_0 100" />
</launch>
```

tf_local.launch

```
<launch>
```

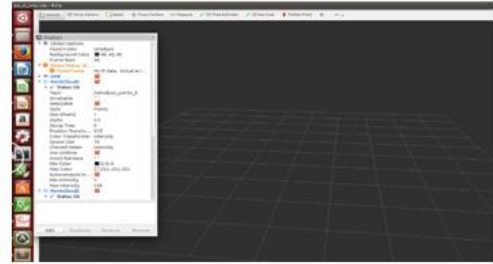
```
  <node pkg="tf" type="static_transform_publisher" name="world_to_map" args="0 0 0 0 0 /world /map 100" />
```

tf_lidar_localization will be used for creating the map. Mapping requires the additional transformation from velodyne, or vehicle, to the base.

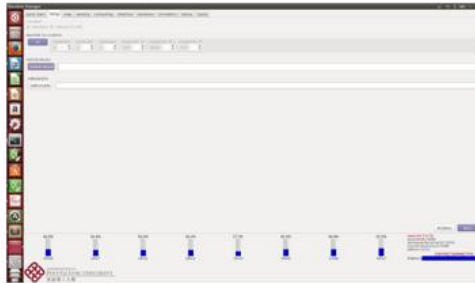
The next part is operation. During the operation, we follow the steps below.



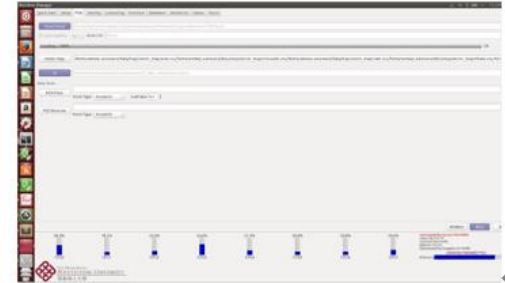
1. Run the point cloud bag



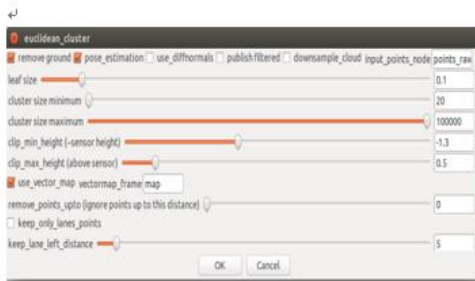
2. Open rviz



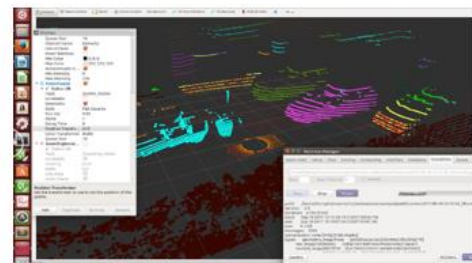
3. Load the transfer function



4. Load the PCD (Point Cloud Display)

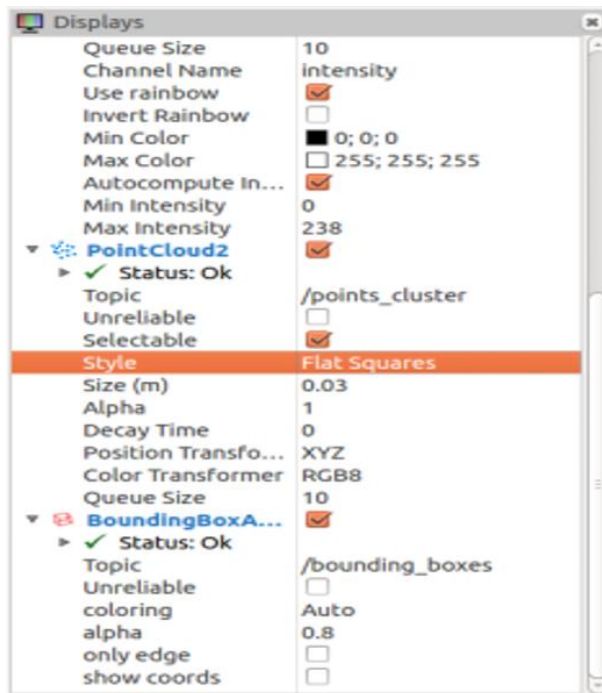


5, modify the setting of euler cluster



6. Euler cluster successfully operate

Euler cluster successfully clusters the points into the cluster. We can clearly remember the shape of the car, the points at the right and side shows the shape of the wall. Euler cluster demonstrates the good performance.



The topic of Euclidean cluster is presented in points_cluster. In the following step, if the operator requires the information from the cluster, the subscriber will subscribe the topic points_cluster. The bounding box is the form that presents the cluster in the rectangular box. The detail information will be explained in the following section.

2.2.2.2 Centroid identification

A. Bounding box method

After the completion of clustering, we can form the bounding box, the result of euler clustering. Bounding box is the application that is used to detect the corner of the cluster and represent in the shape of rectangular boxes (Wu, Zheng, Lai, & Suen ,2015). In the bounding box format, we have pose and orientation of the bounding box, which is in the format of `geometry_msgs::Pose`.

Compact Message Definition

```
std_msgs/Header header  
geometry_msgs/Pose pose  
geometry_msgs/Vector3 dimensions  
float32 value  
uint32 label
```

Compact Message Definition

```
geometry_msgs/Point position  
geometry_msgs/Quaternion orientation
```

Compact Message Definition

```
float64 x
float64 y
float64 z
```

Jsk_recognition_msgs::BoundingBox



Geometry_msgs/Pose



Geometry_msgs/Point



float x, float y, float z

The x ,y, and z in the bounding box indicate the pose of the bounding box in the free space, which represents the centroid of the bounding box. We apply the following code to extract the bounding box centroid.

```
1. for(unsigned int i=0; i<final_clusters.size(); i++)
2.
3.     {
4.         *out_cloud_ptr = *out_cloud_ptr + *(final_clusters[i]->GetCloud());
5.
6.         jsk_recognition_msgs::BoundingBox bounding_box = final_clusters[i]->GetBoundingBox();
7.
8.         geometry_msgs::PolygonStamped polygon = final_clusters[i]->GetPolygon();
9.
10.        jsk_rviz_plugins::Pictogram pictogram_cluster;
11.
12.        pictogram_cluster.header = _velodyne_header;
13.
14.        //PICTO
15.        pictogram_cluster.mode = pictogram_cluster.STRING_MODE;
16.
17.        pictogram_cluster.pose.position.x = final_clusters[i]->GetMaxPoint().x;
18.
19.        pictogram_cluster.pose.position.y = final_clusters[i]->GetMaxPoint().y;
20.
21.        pictogram_cluster.pose.position.z = final_clusters[i]->GetMaxPoint().z;
22.
23.        tf::Quaternion quat(0.0, -0.7, 0.0, 0.7);
24.
25.        tf::quaternionTFToMsg(quat, pictogram_cluster.pose.orientation);
```

```

19.         pictogram_cluster.size = 4;
20.         std_msgs::ColorRGBA color;
21.         color.a = 1; color.r = 1; color.g = 1; color.b = 1;
22.         pictogram_cluster.color = color;
23.         pictogram_cluster.character = std::to_string( i );
24.         //PICTO
25.
26.         //pcl::PointXYZ min_point = final_clusters[i]->GetMinPoint();
27.         //pcl::PointXYZ max_point = final_clusters[i]->GetMaxPoint();
28.         pcl::PointXYZ center_point = final_clusters[i]->GetCentroid();
29.         geometry_msgs::Point centroid;
30.         centroid.x = center_point.x; centroid.y = center_point.y; centroid.z = c
        enter_point.z;
31.         bounding_box.header = _velodyne_header;
32.         polygon.header = _velodyne_header;
33.         geometry_msgs::Point centroid2;
34.         centroid2.x=bounding_box.pose.position.x;
35.         centroid2.y=bounding_box.pose.position.y;
36.         centroid2.z=bounding_box.pose.position.z;
37.         chat.publish(centroid);
38.         chat1.publish(centroid2)

```

Code for bounding box to get centroid:

```

bounding_box.header = _velodyne_header;
jsk_recognition_msgs::BoundingBox bounding_box = final_clusters[i]->GetBoundingBox();
geometry_msgs::Point centroid2;
        centroid2.x=bounding_box.pose.position.x;

```

Be aware that the bounding box header should be received from the header of Velodyne, or lidar. The header file includes the information of the frame and the time the

After we receive the bounding box of each cluster, we add the desired value to the class centroid2 and publish to the topic chatter1. The subscriber will read the center of the bounding box.

```
1. void chatterCallback(const geometry_msgs::Point::ConstPtr&roid2)
2. {
3.     ROS_INFO("I heard: [%d],[%d],[%d]", centroid2->x,centroid2->y,centroid2->z);
4. }
5.
6. int main(int argc, char**argv)
7. {
8.
9.     ros::init(argc, argv, "boxing");
10.    ros::NodeHandle n;
11.    ros::Subscriber sub = n.subscribe("chatter1", 10, chatterCallback);
12.    ros::spin();
13.    return 0;
14. }
```

The output format is shown as follows:

```
[ INFO] [1523524542.771348230]: I heard: [4271034],[1610612736],[1207402720]
[ INFO] [1523524542.771499053]: I heard: [4271034],[-536870912],[1207402720]
[ INFO] [1523524542.771678960]: I heard: [4271034],[-1073741824],[1207402720]
[ INFO] [1523524542.771866795]: I heard: [4271034],[-1073741824],[1207402720]
[ INFO] [1523524542.772060831]: I heard: [4271034],[0],[1207402720]
[ INFO] [1523524542.772286333]: I heard: [4271034],[-536870912],[1207402720]
[ INFO] [1523524542.772547569]: I heard: [4271034],[-2147483648],[1207402720]
[ INFO] [1523524542.772792233]: I heard: [4271034],[1073741824],[1207402720]
[ INFO] [1523524542.773174274]: I heard: [4271034],[0],[1207402720]
[ INFO] [1523524544.649970511]: I heard: [4271034],[0],[1207402720]
[ INFO] [1523524544.650713997]: I heard: [4271034],[-536870912],[1207402720]
[ INFO] [1523524544.657387195]: I heard: [4271034],[1073741824],[1207402720]
[ INFO] [1523524544.657808771]: I heard: [4271034],[-1610612736],[1207402720]
[ INFO] [1523524544.657939148]: I heard: [4271034],[-1073741824],[1207402720]
[ INFO] [1523524544.658038716]: I heard: [4271034],[-536870912],[1207402720]
[ INFO] [1523524544.658107993]: I heard: [4271034],[-1073741824],[1207402720]
[ INFO] [1523524544.658166289]: I heard: [4271034],[1073741824],[1207402720]
[ INFO] [1523524544.658212942]: I heard: [4271034],[1073741824],[1207402720]
[ INFO] [1523524544.658256327]: I heard: [4271034],[0],[1207402720]
```

B.Cluster method

Another way to extract the centroid from the cluster is to directly receive the centroid information from each cluster since each Euclidean cluster contains the information of finalized centroid. From the code mentioned above, we get the centroid of each cluster.

The code for extracting centroid from the cluster:

```
pcl::PointXYZ center_point = final_clusters[i]->GetCentroid();

    geometry_msgs::Point centroid;

    centroid.x = center_point.x; centroid.y = center_point.y; centroid.z =
```

Then, we construct the subscriber to receive the data from the topic called chatter.

```
1. void chatterCallback(const geometry_msgs::Point::ConstPtr&roid)
2. {
3.     ROS_INFO("I heard: [%d],[%d],[%d]", centroid->x,centroid->y,centroid->z);
4. }
5.
6. int main(int argc, char**argv)
7. {
8.
9.     ros::init(argc, argv, "listener");
10.
11.     ros::NodeHandle n;
12.     ros::Subscriber sub = n.subscribe("chatter", 10, chatterCallback);
13.     ros::spin();
14.     return 0;
15. }
```

The result is shown in the following diagram

```
[ INFO] [1523175833.313431284, 1510654735.174504003]: I heard: [4271034],[1610612736],[-44054368]
[ INFO] [1523175835.709216990, 1510654735.472423322]: I heard: [4271034],[-536870912],[-44054368]
[ INFO] [1523175835.711518617, 1510654735.472423322]: I heard: [4271034],[-2147483648],[-44054368]
```

The centroids are published in 3d format, but we have already decided to use the info in 2D. Therefore, in the youbot. We only assign x and y coordinates to the centroid of the block

```
block_x-0.726277
block_y0.0562454
[ INFO] [1523342954.071195054, 1510654950.566981032]: I heard: Henry
block_x-2.65855
block_y0.118644
[ INFO] [1523342954.071544890, 1510654950.566981032]: I heard: Henry
block_x-5.43189
block_y0.0418869
[ INFO] [1523342954.071943895, 1510654950.566981032]: I heard: Henry
block_x-7.23344
block_y0.0897913
[ INFO] [1523342954.072310563, 1510654950.566981032]: I heard: Henry
block_x-5.51811
block_y0.0601897
[ INFO] [1523342954.072669827, 1510654950.566981032]: I heard: Henry
block_x-3.6921
block_y0.2167
[ INFO] [1523342954.073030907, 1510654950.566981032]: I heard: Henry
block_x-7.73088
block_y-0.376084
[ INFO] [1523342954.073384165, 1510654950.566981032]: I heard: Henry
block_x6.5703
block_y0
```

2.3 Path tracking and autonomous control

In this part, the mission is to directly control the KUKA youbot based on the signals from the cluster part and localization part. Furthermore, simple path planning is required to avoid obstacles. The communication between KUKA and external computer is also necessary to consider and study. To install Lidar, a structure made by acrylic boards is manufactured and assembled on the KUKA youbot. Therefore, the methodology can be divided into several parts as followed:

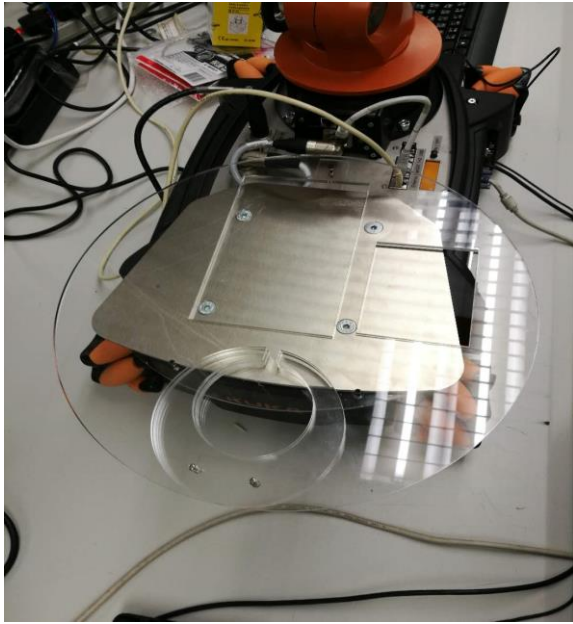
1. Manufacturing acrylic structure
2. How to control KUKA to move by coding
3. Communication between KUKA and external computer
4. Real-time speed calculation and path planning

2.3.1 Manufacturing acrylic board

To fasten the Lidar on the KUKA, a structure which can hold Lidar, its switch and its battery are manufactured. CAD is used for the drawings and acrylic boards are chosen as the raw materials. We went to IC center to use laser machine for cutting. To stabilize the upper acrylic board, a T-shape board is installed below and screws are used to fasten them. Two square holes are for the switch and battery of Lidar. A 'cup' for holding Lidar,

which height is approximately 5cm, is consisted of three c-shape acrylic board and long 4mm screws. This height can make sure to hold Lidar tightly without blocking its sight.

A hole is left for the wire of Lidar. The structure is shown below:



(upper view)



(install Lidar)



(T-shape acrylic board)

2.3.2 How to control KUKA to move by coding

To control KUKA to move a certain pathway, it is necessary to follow the ROS rules to create a package and add dependencies. The system will automatically generate Source file and CMakeLists in the package. In the CMakeLists, dependencies and executable cpp for this package should be pointed out. In this case, the code should be as followed:

1. find_package

2. (catkin REQUIRED COMPONENTS
3. geometry_msgs
4. roscpp
5. std_msgs
6. rospy
7. message_generation)
8. add_executable(sam1 src/sam1.cpp)
9. target_link_libraries(sam1 \${catkin_LIBRARIES})

The executable file name is sam1.cpp, which is inside the Source file. In the executable cpp, C++ language can be used to write codes and since the source code is quite comprehensive in KUKA, we can directly give the total speed instead of the speeds for each wheel. An example is given below:

1. geometry_msgs::Twist twist;
2. twist.linear.x = -0.05;
3. twist.linear.y = 0.1;
4. platformPublisher.publish(twist);
5. ros::Duration(30).sleep();

In this case, the speed in the x-direction is -0.05 meter per second and 0.1 meters per second in the y-direction. This speed will last for 30 seconds before changing to the following speeds.

Furthermore, in the main function, node name should be included and the command should be published to the topic that directly controls KUKA. The code should be like this:

```
1. ros::init(argc, argv, "sam1");  
2. platformPublisher = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);
```

After completing the code in cpp, it is required to build the package so that it can be executed. We need to type the following command in a terminal:

```
1. cd youbot_nav  
2. catkin_make
```

If the package can be built successfully (see the picture below), we can run the package like this:

The screenshot shows a Linux desktop with a terminal window and a code editor. The terminal window displays the output of a ROS build process for the 'sam1' package. The build process includes scanning dependencies, building targets like 'sam1_generate_messages_eus', 'std_msgs_generate_messages_py', 'geometry_msgs_generate_messages_py', 'std_msgs_generate_messages_lisp', 'geometry_msgs_generate_messages_lisp', 'std_msgs_generate_messages_cpp', 'geometry_msgs_generate_messages_cpp', 'sam1_generate_messages_py', 'sam1_generate_messages_lisp', 'sam1_generate_messages_cpp', and 'sam1_generate_messages'. The final step is linking the CXX executable to '/home/youbot/youbot_nav/devel/lib/sam1/sam1'. The code editor shows a C++ file 'sam1.cpp' with lines 66 to 75, which include ROS_INFO and geometry_msgs::PoseStamped declarations.

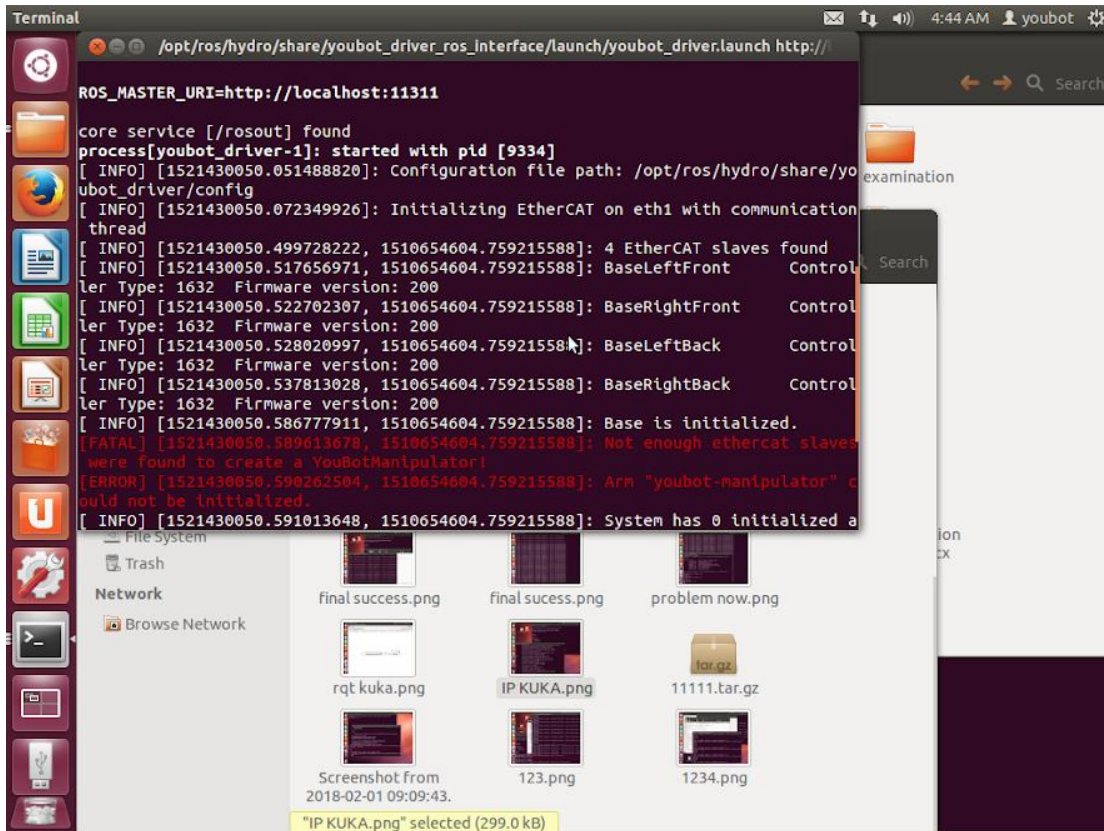
```
Terminal
youbot@ubuntu: ~/youbot_nav
Install space: /home/youbot/youbot_nav/install
####
#### Running command: "make cmake_check_build_system" in "/home/youbot/youbot_nav/build"
####
#### Running command: "make -j2 -l2" in "/home/youbot/youbot_nav/build"
####
Scanning dependencies of target sam1
[ 50%] Built target sam1_generate_messages_eus
[100%] [100%] Built target std_msgs_generate_messages_py
Building CXX object sam1/CMakeFiles/sam1.dir/src/sam1.cpp.o
[100%] Built target geometry_msgs_generate_messages_py
[100%] Built target std_msgs_generate_messages_lisp
[100%] Built target geometry_msgs_generate_messages_lisp
[100%] Built target std_msgs_generate_messages_cpp
[100%] Built target geometry_msgs_generate_messages_cpp
[100%] Built target sam1_generate_messages_py
[100%] Built target sam1_generate_messages_lisp
[100%] Built target sam1_generate_messages_cpp
[100%] Built target sam1_generate_messages
Linking CXX executable /home/youbot/youbot_nav/devel/lib/sam1/sam1
[100%] Built target sam1
youbot@ubuntu:~/youbot_nav$
```

```
66 //std::cout<<"vx"<<twist.linear.x<<std::endl;
67 //std::cout<<"vy"<<twist.linear.y<<std::endl;
68 }
69
70 void ndt_poseCallback(const
  geometry_msgs::PoseStamped::ConstPtr& msg)
71 {
72   ROS_INFO("I heard: ndt_matching pose");
73   //geometry_msgs::PoseStamped msg_1;
74   //msg_1 = msg;
75   current ndt pose x = (*msg).pose.position.x;
```

In one terminal, to start the driver, type:

1. `roslaunch youbot_driver_ros_interface youbot_driver.launch`

If the driver is started successfully, the terminal should be like:



(base driver starts)

In another terminal, type:

1. `roslaunch sam1 sam1`

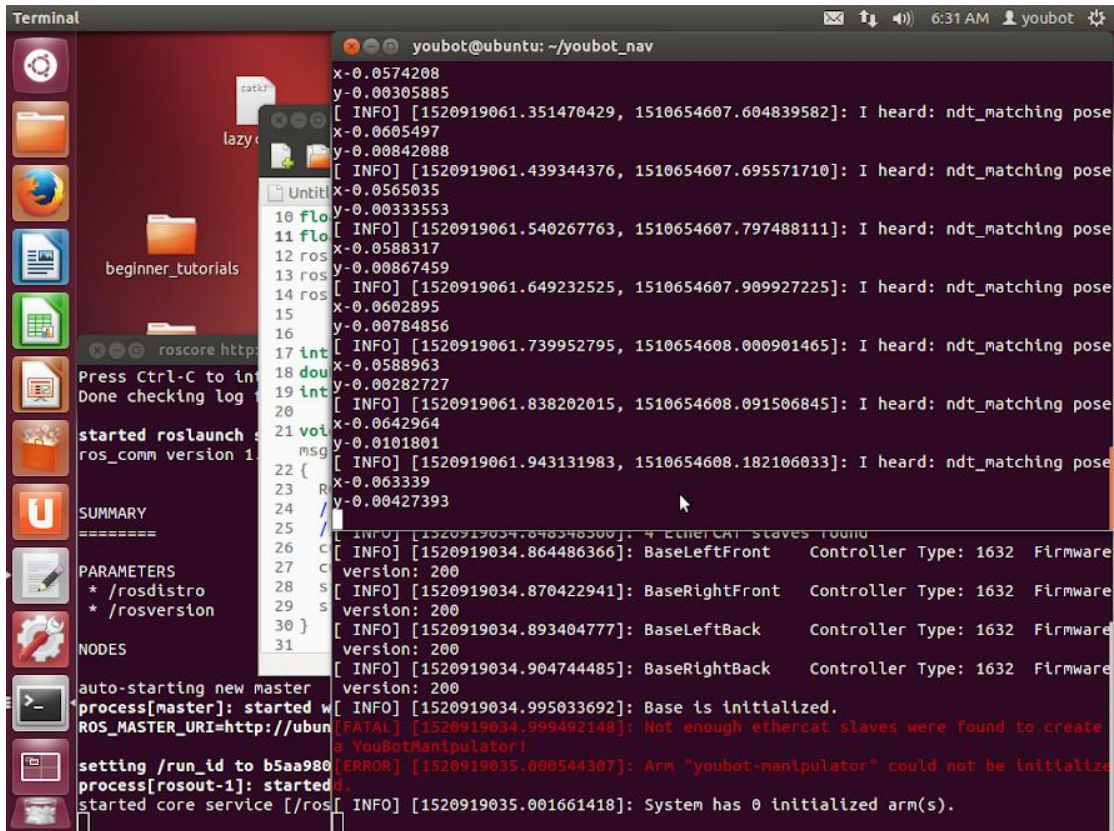
In this way, KUKA will move according to the speed as well as the duration we have set.

2.2.3 Communication between KUKA and external computer

To receive data from external computer, listeners should be included in the executable file. The listener for the localization is as followed:

```
1. void ndt_poseCallback(const geometry_msgs::PoseStamped::ConstPtr& msg)
2. {
3.     current_ndt_pose_x = (*msg).pose.position.x;
4.     current_ndt_pose_y = (*msg).pose.position.y;
5.     std::cout<<"x"<<current_ndt_pose_x<<std::endl;
6.     movePlatform(current_ndt_pose_x,current_ndt_pose_y);
7. }
```

In this code, `(*msg).pose.position.x` is the value of x in the coordinate system and `(*msg).pose.position.y` is the value of y in the coordinate system. The data type should be the same as it in the localization package. Otherwise, the data transmission will be failed. In KUKA's package, `current_ndt_pose_x` is the global variable which represents the value of x while `current_ndt_pose_y` represents the value of y. Then the `movePlatform` function will be executed. As shown in the following picture, the position is changing and KUKA is receiving the information:



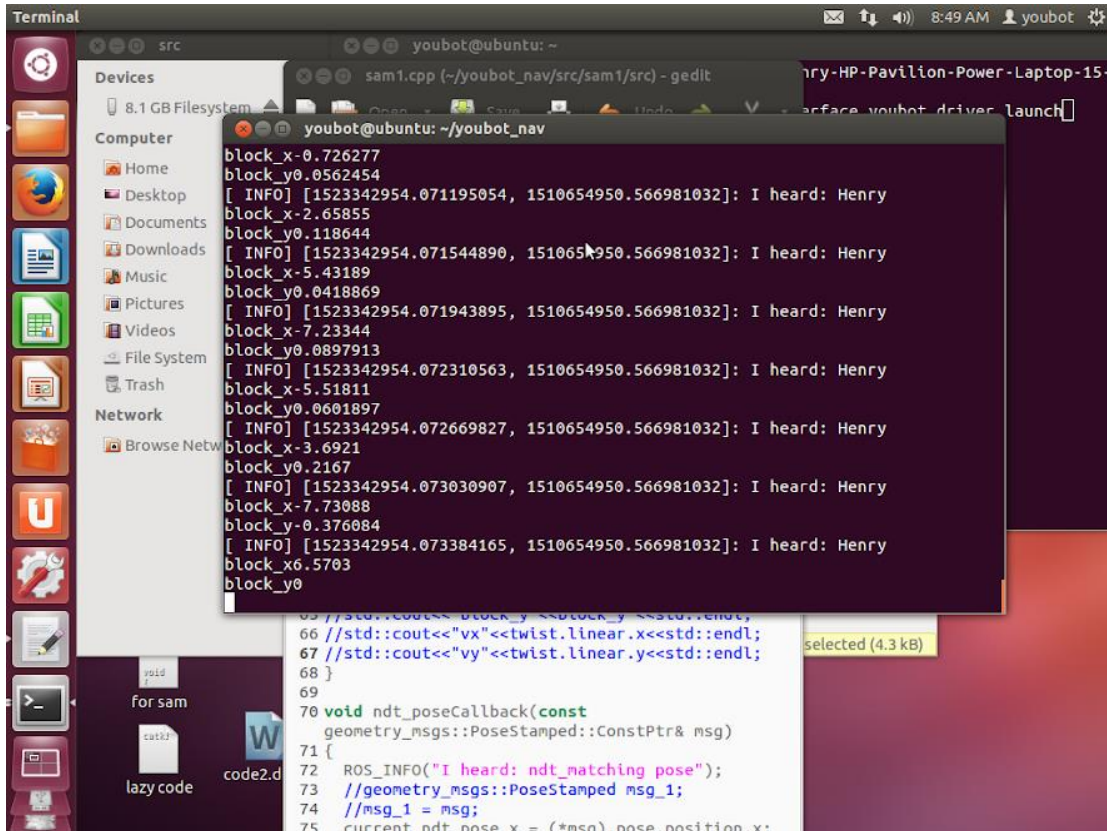
```
youbot@ubuntu: ~/youbot_nav
x-0.0574208
y-0.00305885
[ INFO] [1520919061.351470429, 1510654607.604839582]: I heard: ndt_matching pose
x-0.0605497
y-0.00842088
[ INFO] [1520919061.439344376, 1510654607.695571710]: I heard: ndt_matching pose
x-0.0565035
y-0.00333553
[ INFO] [1520919061.540267763, 1510654607.797488111]: I heard: ndt_matching pose
x-0.0588317
y-0.00867459
[ INFO] [1520919061.649232525, 1510654607.909927225]: I heard: ndt_matching pose
x-0.0602895
y-0.00784856
[ INFO] [1520919061.739952795, 1510654608.000901465]: I heard: ndt_matching pose
x-0.0588963
y-0.00282727
[ INFO] [1520919061.838202015, 1510654608.091506845]: I heard: ndt_matching pose
x-0.0642964
y-0.0101801
[ INFO] [1520919061.943131983, 1510654608.182106033]: I heard: ndt_matching pose
x-0.063339
y-0.00427393
[ INFO] [1520919034.840348300]: 4 EtherCAT slaves found
[ INFO] [1520919034.864486366]: BaseLeftFront Controller Type: 1632 Firmware
version: 200
[ INFO] [1520919034.870422941]: BaseRightFront Controller Type: 1632 Firmware
version: 200
[ INFO] [1520919034.893404777]: BaseLeftBack Controller Type: 1632 Firmware
version: 200
[ INFO] [1520919034.904744485]: BaseRightBack Controller Type: 1632 Firmware
version: 200
[ INFO] [1520919034.995033692]: Base is initialized.
[FATAL] [1520919034.999492148]: Not enough ethercat slaves were found to create
a YouBotManipulator!
[ERROR] [1520919035.000544307]: Arm "youbot-manipulator" could not be initialize
d.
[ INFO] [1520919035.001661418]: System has 0 initialized arm(s).
```

(picture of the changing positions)

To receive the central point of obstacles, another listener has been built in the package:

1. `void chatterCallback(const geometry_msgs::Point::ConstPtr& centroid)`
2. `{`
3. `block_x = (*centroid).y;`
4. `block_y=(*centroid).z;`
5. `std::cout<<"block_x"<<block_x <<std::endl;`
6. `}`

Where `(*centroid).y` is the x value of centroid and `(*centroid).z` is the y value. `block_x` and `block_y` are global variables that represent the corresponding x and y values.



The screenshot shows a terminal window with the following output:

```

block_x-0.726277
block_y0.0562454
[ INFO] [1523342954.071195054, 1510654950.566981032]: I heard: Henry
block_x-2.65855
block_y0.118644
[ INFO] [1523342954.071544890, 1510654950.566981032]: I heard: Henry
block_x-5.43189
block_y0.0418869
[ INFO] [1523342954.071943895, 1510654950.566981032]: I heard: Henry
block_x-7.23344
block_y0.0897913
[ INFO] [1523342954.072310563, 1510654950.566981032]: I heard: Henry
block_x-5.51811
block_y0.0601897
[ INFO] [1523342954.072669827, 1510654950.566981032]: I heard: Henry
block_x-3.6921
block_y0.2167
[ INFO] [1523342954.073030907, 1510654950.566981032]: I heard: Henry
block_x-7.73088
block_y-0.376084
[ INFO] [1523342954.073384165, 1510654950.566981032]: I heard: Henry
block_x6.5703
block_y0

```

The code editor shows the following C++ code:

```

65 //std::cout<< "block_y "<<block_y <<std::endl;
66 //std::cout<<"vx"<<twist.linear.x<<std::endl;
67 //std::cout<<"vy"<<twist.linear.y<<std::endl;
68 }
69
70 void ndt_poseCallback(const
  geometry_msgs::PoseStamped::ConstPtr& msg)
71 {
72   ROS_INFO("I heard: ndt_matching pose");
73   //geometry_msgs::PoseStamped msg_1;
74   //msg_1 = msg;
75   current_ndt_pose.x = (*msg).pose.position.x;

```

(picture of block position)

It should be pointed out that a necessary function should be included in the main function, which is:

1. `ros::spin();`

This powerful function enables the listener to receive real-time data whenever any one of the data value changes, which means, KUKA will get its instant position information as well as the obstacles' information during its movement, then execute movePlatform function again and again until it stops. With this function, speed value and direction can be calculated immediately, and path planning is achievable.

As mentioned in the previous content, some problems have been encountered when we tried to connect KUKA to the external computer, yet we solved them. To sum up, after the connection between KUKA and external computer is established, the process should be like this:

To start the driver of four wheels, in one terminal type:

1. `export ROS_MASTER_URI=http:// ahren-Inspiron-7537:11311`
2. `roslaunch youbot_driver_ros_interface youbot_driver.launch`

To run the package in KUKA, in another terminal type:

1. `export ROS_MASTER_URI=http:// ahren-Inspiron-7537:11311`
2. `cd youbot_nav`
3. `source devel/setup.bash`

```
4. rosrun sam1 sam1
```

Then the KUKA will receive data from external computer, calculate real-time speed and publish to the driver.

2.2.4 Real-time speed calculation and path planning

To do the speed calculation and path planning, it is essential to ensure that KUKA and Lidar use the same coordinate system. Since a map has been detected and built by Lidar, KUKA will use the coordinate system in Lidar as well.

Real-time speed calculation

To design how to calculate the real-time speed, we need to first analyze the requirements:

1. The total speed should not be fast to avoid damage of KUKA and Lidar
2. Both speed direction as well as duration can be calculated.
3. Lidar is accurate enough to tell the instant position.
4. After reaching the destination, KUKA should stop.

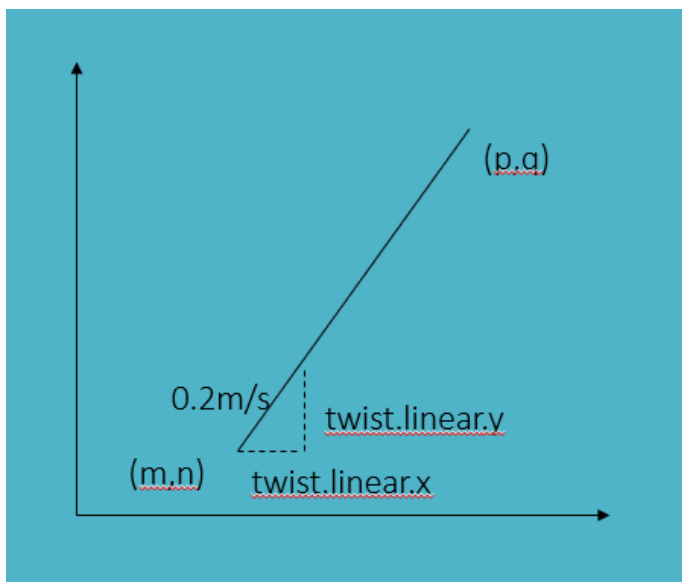
Therefore, 0.2m/s is set to be the total speed and speed in x and y-direction will be calculated in regard to the current position and the destination. The speed-direction can change from time to time so that we do not need to consider the speed duration. When KUKA reaches the destination, the program will end. The main code is as followed, where m, n is the current x and y position while p, q is the destination. As mentioned above, twist.linear.x is the speed in x-direction and twist.linear.y is the speed in the y-direction.

```
1. void movePlatform(double m, double n) {  
2.     double a;  
3.     geometry_msgs::Twist twist;  
4.     if(m==p&& n==q)  
5.     {  
6.         return;  
7.     }  
8.     if(obj_y[obj]==n)  
9.     {  
10. twist.linear.x=0.2;  
11. twist.linear.y=0;  
12. }  
13. else  
14. {  
15. a=abs(obj_x[obj]-m)/abs(obj_y[obj]-n);  
16. twist.linear.y=sqrt(0.04/(a*a+1));
```

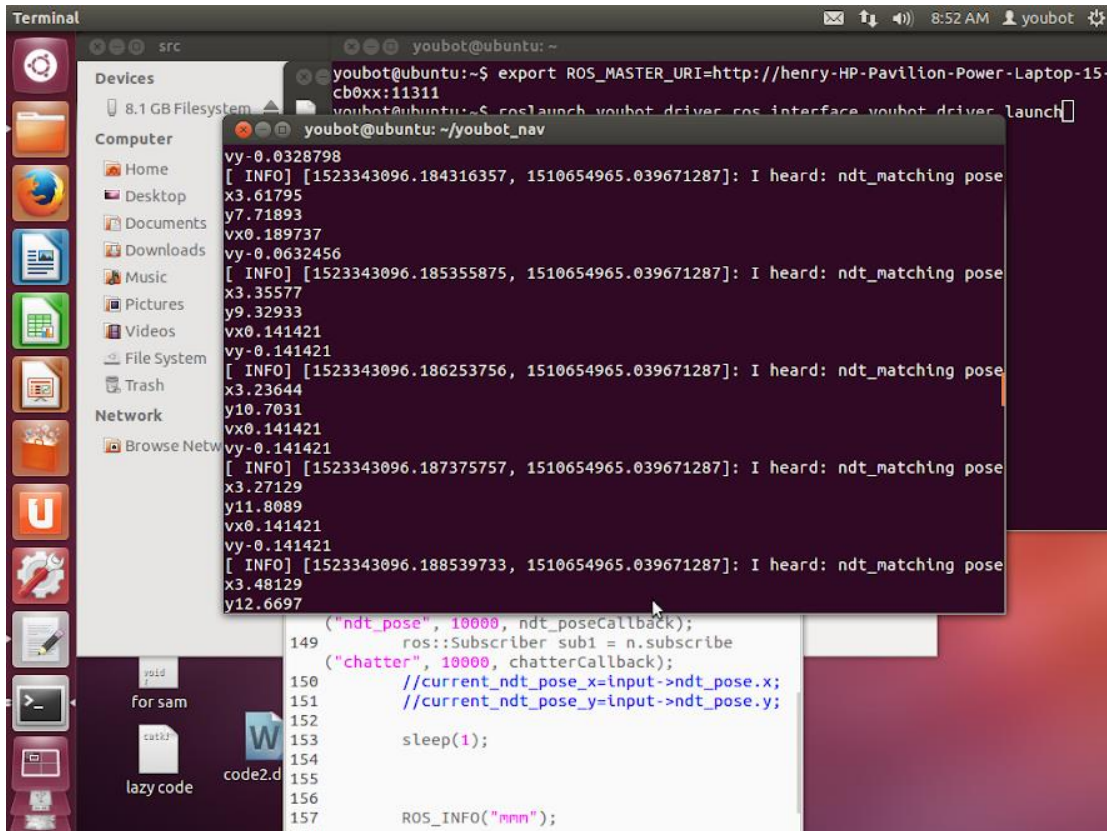
```

17. twist.linear.x=a*twist.linear.y;
18. }
19. if(obj_x[obj]<m) twist.linear.x=-twist.linear.x;
20. if(obj_y[obj]<n) twist.linear.y=-twist.linear.y;
21. platformPublisher.publish(twist);
22. }

```



As mentioned in the localization, in the Rviz simulation, the position is changing so we can use it to check whether the speed is changing corresponding to different positions. The result is shown below where x , y are position values in x and y -direction and v_x , v_y are speed values in x and y -direction.



```
youbot@ubuntu:~$ export ROS_MASTER_URI=http://henry-HP-Pavilion-Power-Laptop-15-cb0xx:11311
youbot@ubuntu:~$ roslaunch youbot_driver ros_interface youbot_driver.launch

youbot@ubuntu: ~/youbot_nav
vy-0.0328798
[ INFO] [1523343096.184316357, 1510654965.039671287]: I heard: ndt_matching pose
x3.61795
y7.71893
vx0.189737
vy-0.0632456
[ INFO] [1523343096.185355875, 1510654965.039671287]: I heard: ndt_matching pose
x3.35577
y9.32933
vx0.141421
vy-0.141421
[ INFO] [1523343096.186253756, 1510654965.039671287]: I heard: ndt_matching pose
x3.23644
y10.7031
vx0.141421
vy-0.141421
[ INFO] [1523343096.187375757, 1510654965.039671287]: I heard: ndt_matching pose
x3.27129
y11.8089
vx0.141421
vy-0.141421
[ INFO] [1523343096.188539733, 1510654965.039671287]: I heard: ndt_matching pose
x3.48129
y12.6697

149 ("ndt_pose", 10000, ndt_poseCallback);
    ros::Subscriber sub1 = n.subscribe
150 ("chatter", 10000, chatterCallback);
151 //current_ndt_pose_x=input->ndt_pose.x;
152 //current_ndt_pose_y=input->ndt_pose.y;
153
154     sleep(1);
155
156
157     ROS_INFO("mmm");
158
```

Path planning without obstacles

After successfully reaching one destination, it is not difficult to give a certain pathway to KUKA. To be more specific, KUKA can move to the desired positions one after another or it can move a certain shape like a square. The desired positions are set in the main function as shown below, where `obj_x[]` and `obj_y[]` are the x and y position in the pathway:

1. `int main(int argc, char **argv) {`
2. `obj_x[0]=3;obj_y[0]=3;`

```

3.  obj_x[1]=4;obj_y[1]=-1;
4.  obj_x[2]=6;obj_y[2]=-1;
5.  ros::init(argc, argv, "sam1");
6.  ros::NodeHandle n;
7.  platformPublisher = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);
8.  ros::Subscriber sub1 = n.subscribe("ndt_pose", 10000, ndt_poseCallback);
9.  ros::Subscriber sub2 = n.subscribe("chatter", 10000, ndt_poseCallback);
10. ros::spin();
11. return 0;
12. }

```

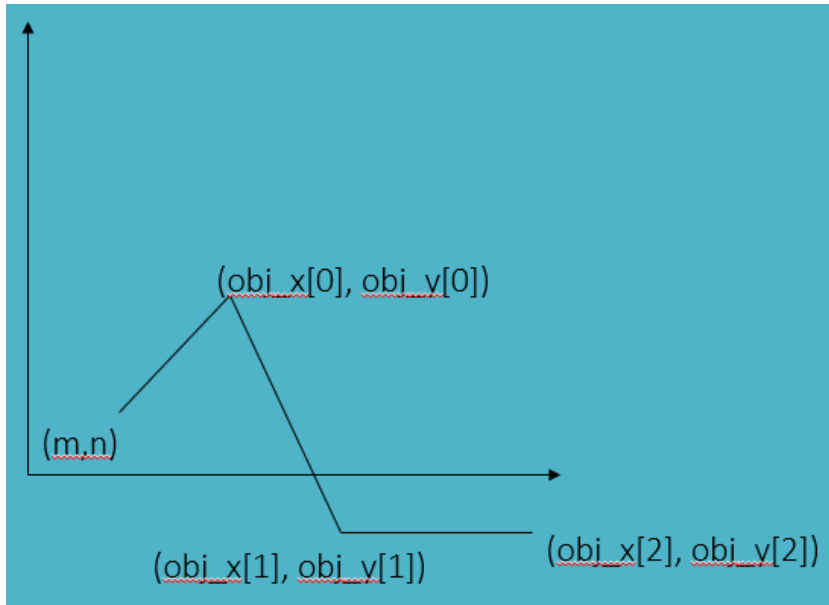
When KUKA reaches one point, it will automatically head towards the next point until the destination. The total speed is always 0.2m/s. The movePlatform function and an example of pathway is shown below:

```

1. int times_n=3;
2. double obj_x[3],obj_y[3];
3. int obj=0;
4.
5. void movePlatform(double m, double n) {
6.     double a;
7.     geometry_msgs::Twist twist;
8.     if(m==obj_x[obj]&& n==obj_y[obj])

```

```
9. {
10.  if(obj==times_n-1) return;
11.  obj++;
12.  movePlatform(m,n);
13.  return;
14. }
15. if(obj_y[obj]==n)
16. {
17. twist.linear.x=0.2;
18. twist.linear.y=0;
19. }
20. else
21. {
22. a=abs(obj_x[obj]-m)/abs(obj_y[obj]-n);
23. twist.linear.y=sqrt(0.04/(a*a+1));
24. twist.linear.x=a*twist.linear.y;
25. }
26. if(obj_x[obj]<m) twist.linear.x=-twist.linear.x;
27. if(obj_y[obj]<n) twist.linear.y=-twist.linear.y;
28. platformPublisher.publish(twist);
29. }
```



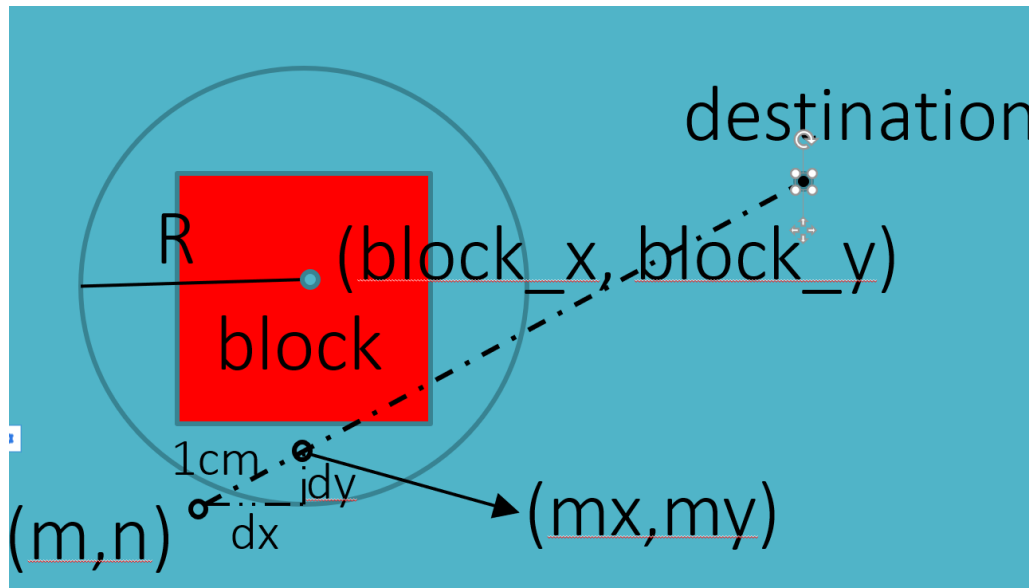
Path planning with obstacles

In the previous sections, all the missions can be fulfilled by implementing localization and direct control of KUKA. In this section, clustering is needed to detect the block and publish the center point to KUKA, just as introduced in the communication between KUKA and external computer. Therefore, we have the parameters as following:

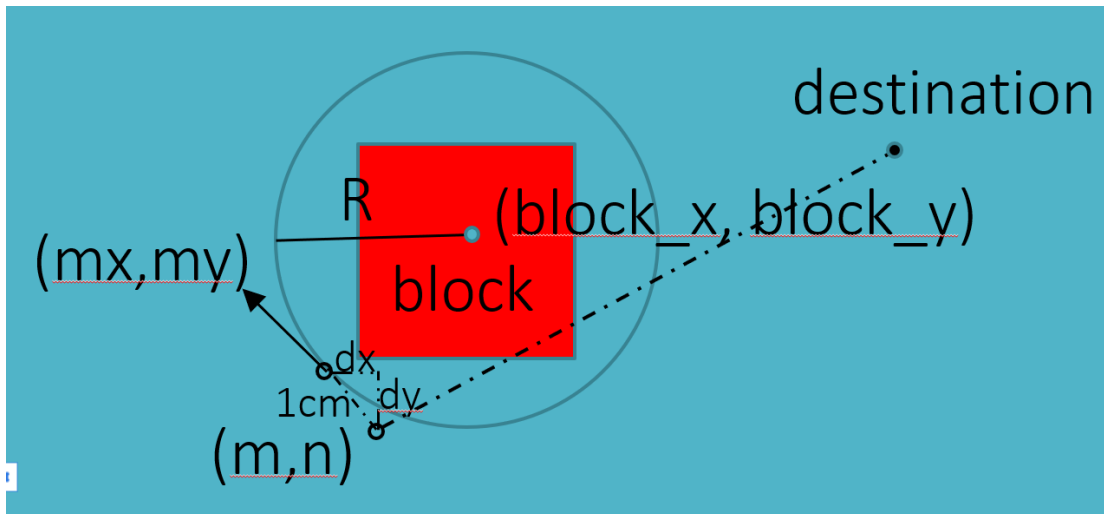
1. Current position (m, n)
2. The center point of the obstacle $(block_x, block_y)$
3. Destination point $(obj_x[obj], obj_y[obj])$

To simplify the detection, it is assumed that the size of obstacle has been know before. Therefore, a circle, which is regarded as the forbidden area, has been drawn and the

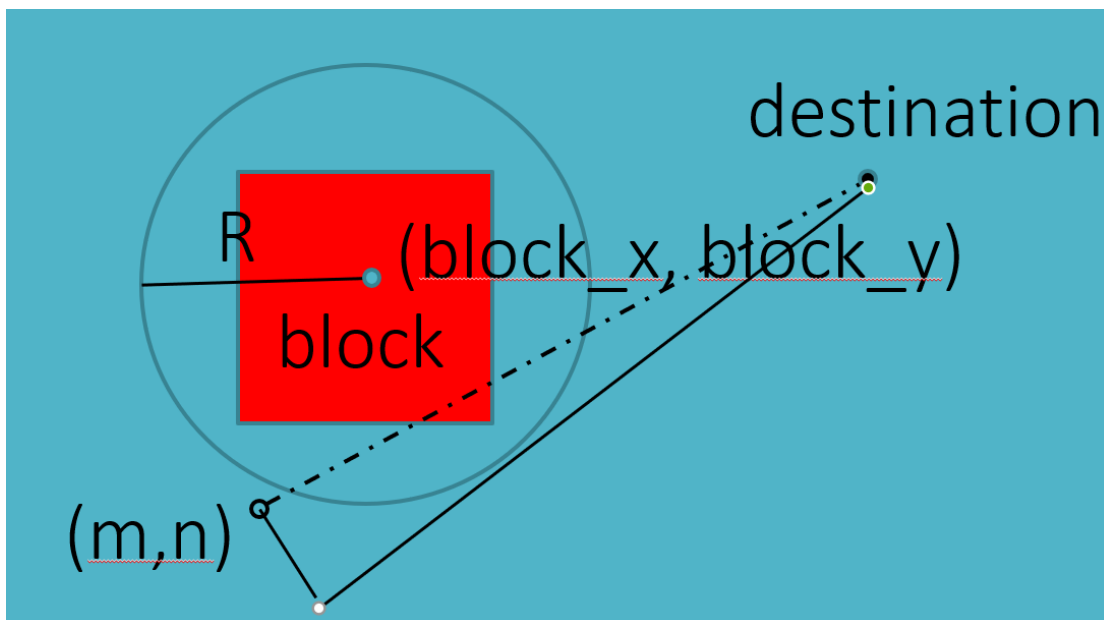
center point is the same as the obstacle. The size of KUKA should also be considered during the calculation of R . The minimum distance of the arc and the block should be bigger than 15cm to ensure the safety of KUKA. An illustrating picture is below:



A point, which is 1cm before KUKA on its pathway, is set to determine whether KUKA will enter the forbidden area or not. If this point (mx, my) is in the area, KUKA will try to move to the left perpendicular to the original direction, which is shown in the following picture:



In this case, the point after turning left is also in the area. Therefore, KUKA will turn right to avoid the obstacle and after (mx, my) is not in the forbidden area, KUKA will directly head towards the destination, just as shown below:

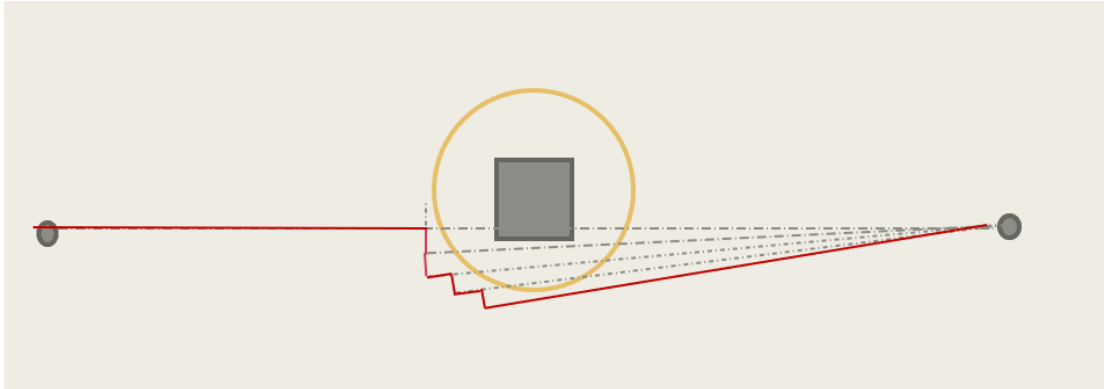


The frequency for determining whether the point (mx, my) is in the area or not is 2HZ, which means KUKA will decide its direction every 1 cm. The code is shown below:

```
1. void movePlatform(double m, double n) {
2.     double a,b;
3.     geometry_msgs::Twist twist;
4.     double mx,my;
5.
6.     if(obj_y[0]==n)
7.     {
8.         mx=obj_x[0]<m?m-0.01:m+0.01;
9.         my=n;
10.        twist.linear.x=0.2;
11.        twist.linear.y=0;
12.    }
13. else
14. {
15.     a=abs(obj_x[obj]-m)/abs(obj_y[obj]-n);
16.     double dy=sqrt(0.0001/(a*a+1));
17.     double dx=a*dy;
18.     mx=obj_x[0]<m?m-dx:m+dx;
19.     my=obj_y[0]<n?n-dy:n+dy;
20.     twist.linear.y=sqrt(0.04/(a*a+1));
21.     twist.linear.x=a*twist.linear.y;
```

```
22. }
23. if(obj_x[obj]<m) twist.linear.x=-twist.linear.x;
24. if(obj_y[obj]<n) twist.linear.y=-twist.linear.y;
25.
26. if((mx-block_x)*(mx-block_x)+(my-block_y)*(my-block_y)<R*R)
27. {
28.     double temp=twist.linear.x;
29.     twist.linear.x=twist.linear.y;
30.     twist.linear.y=-temp;
31.     double nx,ny;
32.     ny=obj_x[0]<m?n+dx:n-dx;
33.     nx=obj_y[0]<n?m-dy:m+dy;
34.     if((nx-block_x)*(nx-block_x)+(ny-block_y)*(ny-block_y)<R*R)
35.     {
36.         twist.linear.x=-twist.linear.x;
37.         twist.linear.y=-twist.linear.y;
38.     }
39. }
40.
41. platformPublisher.publish(twist);
42. ros::Duration(0.5).sleep();
43. }
```

If the frequency is high enough, KUKA will move around the forbidden area like an arc, the following picture gives an overall view of avoiding obstacles:



The following picture shows the changing of speed direction when simulating on Rviz:

```
Terminal
youbot@ubuntu: ~
src
youbot@ubuntu: ~
sam1.cpp (~/youbot_nav/src/sam1/src) - gedit
try-HP-Pavilion-Power-Laptop-15-
youbot@ubuntu: ~/youbot_nav
vx-0.189737
vy0.0632456
[ INFO] [1523343248.924264092, 1510654937.376477335]: I heard: ndt_matching pose
x27.984
y-0.604634
block_x-10.1165
block_y-0.718777
vx-0.189737
vy0.0632456
[ INFO] [1523343249.023796647, 1510654937.467082076]: I heard: ndt_matching pose
x28.0293
y-0.626788
block_x-10.1165
block_y-0.718777
vx-0.189737
vy0.0632456
[ INFO] [1523343249.128323626, 1510654937.577748048]: I heard: ndt_matching pose
x28.0564
y-0.680989
block_x-10.1165
block_y-0.718777
vx-0.189737
vy0.0632456
[ INFO] [1523343249.224630779, 1510654937.671968093]: I heard: ndt_matching pose
64 std::cout<<"block_x"<<block_x<<std::endl;
65 std::cout<<"block_y"<<block_y <<std::endl;
66 std::cout<<"vx"<<twist.linear.x<<std::endl;
67 std::cout<<"vy"<<twist.linear.y<<std::endl;
68 }
69
70 void ndt_poseCallback(const
geometry_msgs::PoseStamped::ConstPtr& msg)
71 {
72   ROS_INFO("I heard: ndt_matching pose");
73   //geometry_msgs::PoseStamped msg_1;
```

Because we have encountered quite a lot of problems when connecting the external computer with KUKA, there is not enough time to code a comprehensive solution for avoiding obstacles. However, this method can fulfill most of our needs and is able to integrate all the three parts together.

Advantages of this path planning method:

1. Easy and efficient to avoid a block.
2. 100 percent sure to avoid the block.
3. Good path planning, no need to drive extra path.

Disadvantages of this path planning method:

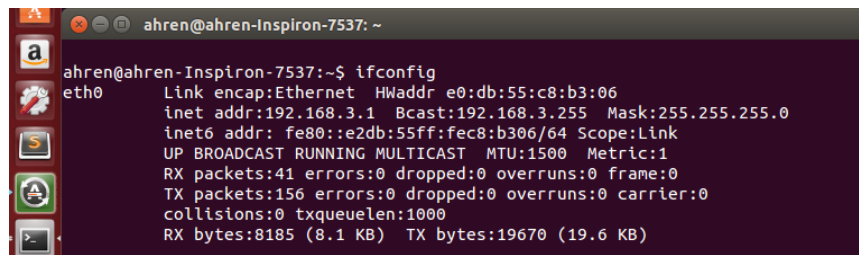
1. This method can only avoid one block or blocks that are far from each other.
2. It is required to know the size of block in advance and code the size.
3. Sometimes there will be some interruption when detecting blocks.

2.4 Integration

2.4.1 Methodology applied

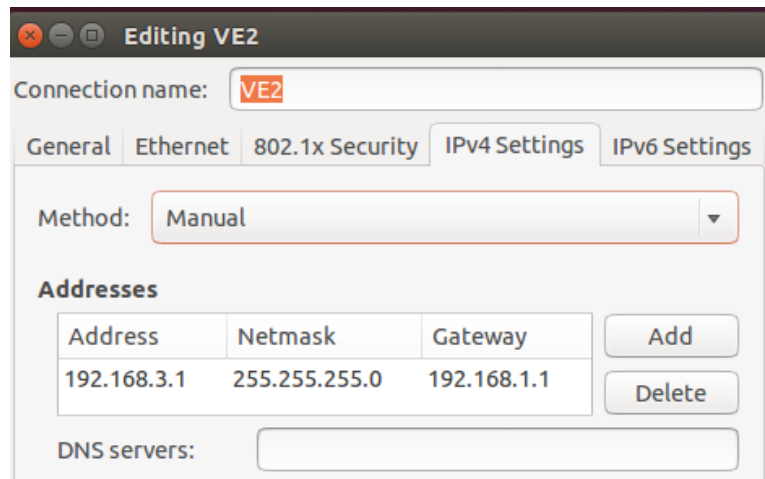
First, check the IP of PC and Kuka by typing command

Ifconfig



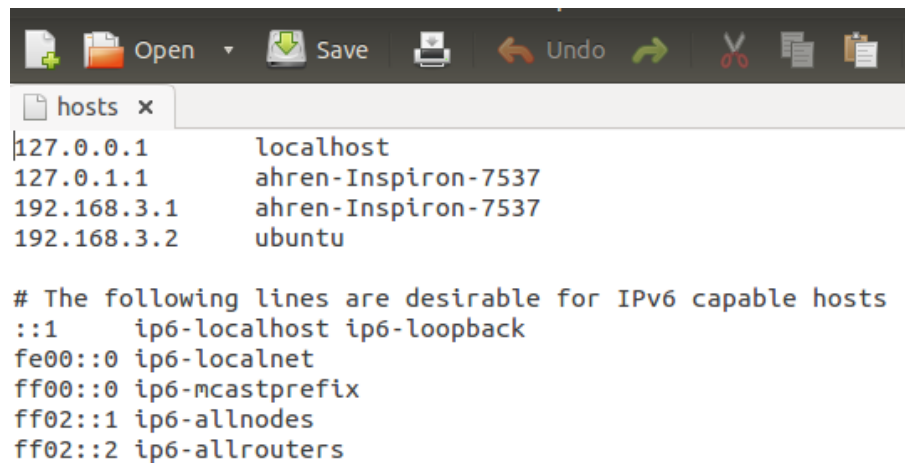
```
ahren@ahren-Inspiron-7537:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr e0:db:55:c8:b3:06
          inet addr:192.168.3.1  Bcast:192.168.3.255  Mask:255.255.255.0
          inet6 addr: fe80::e2db:55ff:fec8:b306/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:41  errors:0  dropped:0  overruns:0  frame:0
          TX packets:156  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8185 (8.1 KB)  TX bytes:19670 (19.6 KB)
```

Then, change their IP to the same network segment(e.g.192.168.3.xxx) to build a LAN.



Because my PC and ubuntu cannot be pinged by directly using IP, I bind the IP address and Hostname of the computer to make the multi-computer connection more convenient.

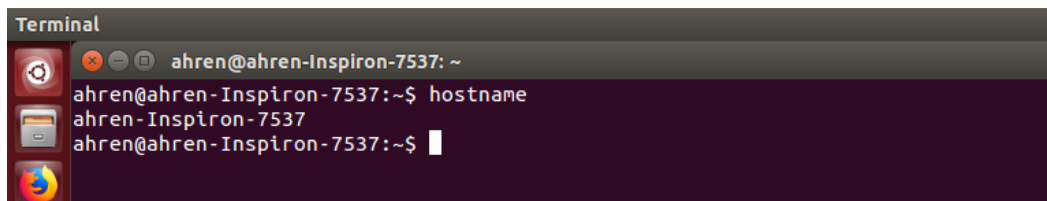
```
gksu gedit /etc/hosts/
```



(figure: hostname of PC is ahren-Inspiron-7537 and hostname of kuka is ubuntu.

Hostname can be checked by typing)

```
hostname
```



Restart and upgrade the network

```
sudo /etc/init.d/networking restart
```

In order to synchronize different computers, we need to synchronize their time.

```
apt-get install chrony
```

Install the ssh server and start it & Check the situation of ssh

```
sudo apt-get install openssh-server & ps -e|grep ssh
```

In the case, my PC is seemed as Master computer.

```
ssh ahren-Inspiron-7537
```

Roscore plays an important role in topic and node communication, so, all of the communicate should be commanded by one roscore. In ubuntu, we need to export master roscore which is run in “ahren-Inspiron-7537”.

```
export ROS_MASTER_URI=http:// ahren-Inspiron-7537:11311
```

3 Experiment design & data and error analysis

3.1 Section test

3.1.1 Localization

The localization test is designed to roughly measure the performance of the localization in Autoware.

- i. Carry the kuka robot to barrow and install the LiDAR on kuka.
- ii. Follow previous `ndt_matching` procedure and LiDAR driver procedure to let all equipment standby.
- iii. After opening the LiDAR and test the connection between PC and LiDAR.
- iv. Move the kuka scan the environment of FJ005. At the same time, observe the `rviz`.
- v. Compare it with real situation pose.

Data and result analyzation

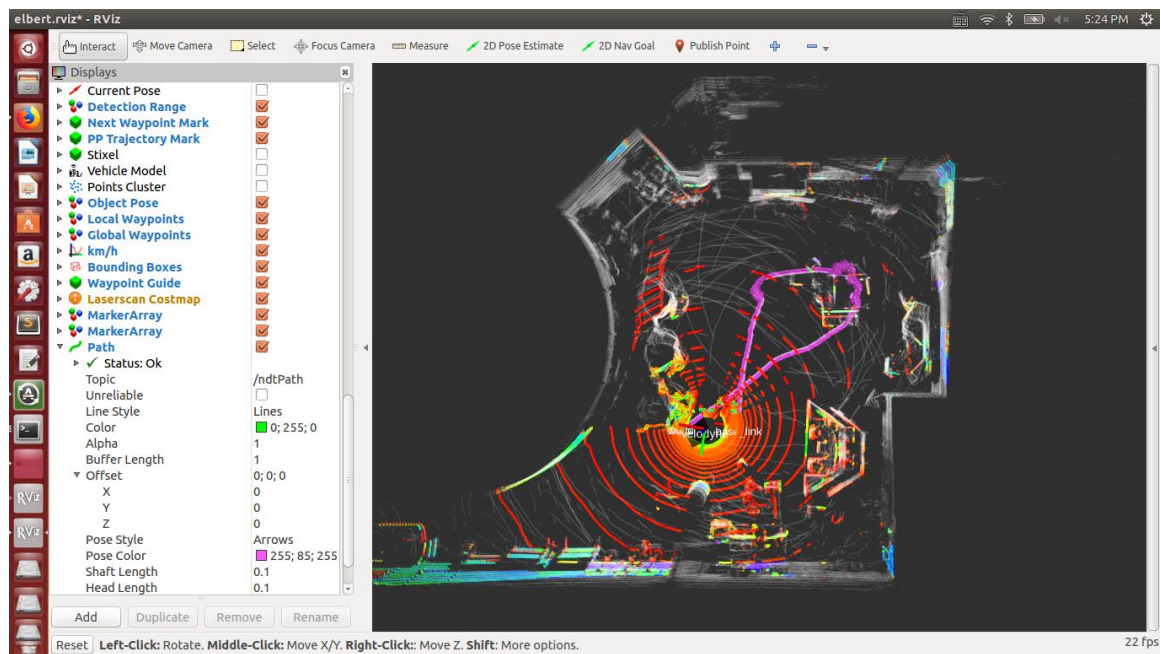
Even though we do not have third-party sensor to test its accuracy, the trail is almost same as what we observed. Some experiment result will be introduced in following.

Colorful concentric circle: it means that the LiDAR is scanning the environment. The distance between LiDAR and barrier is directly proportional to scanning accuracy.

Pink path line: it means the trail of the robot. A lot of localization pose can form the path line. We made a circle of the desks in FJ005 and the `Rviz` shows the trial of localization

which is almost same as the real situation. Roughly, the localization is accurate and workable.

Grey structure: it is the high accuracy pcd map which is generated by using “ndt_mapping”. It shows the rough structure of the lab and by using cluster it can centric the pose of barrier.



Error analysis

In localization part, the error can be affected by these following factors

1. Ethernet cable and usb data transmission rate. Because of the quantity of data which is scanned by LiDAR need be transmated by net cable, it is necessary to consider the transmission rate. In the project, we use CAT-6 net cable and the rate of CAT-6 can achieve 250Mbps.

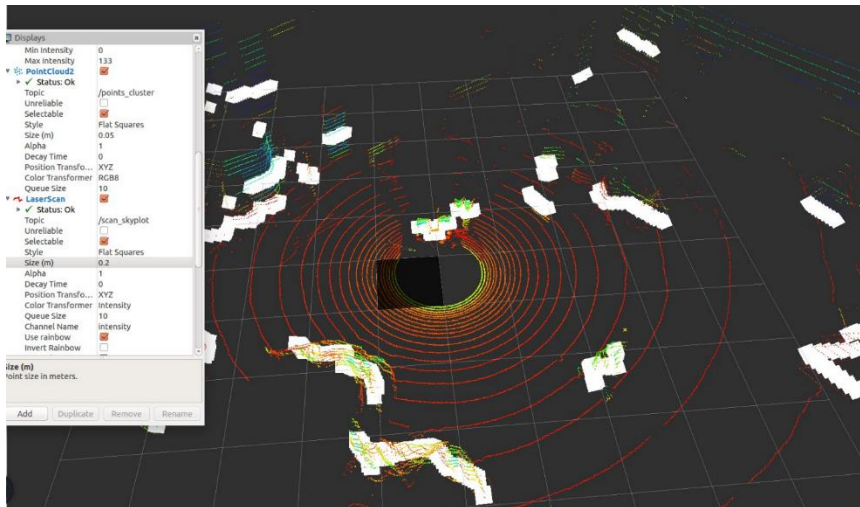
2. In our project, odom and imu were not used. So, comparing with multi-sensors project, the project still a little weak in stability and accuracy. The prediction is also not that accurate.

3.1.2 Clustering and navigation

3.1.2.1 Cost mapping Method

Laserscan Experiments

Laserscan is the crucial part of the method 1. Laser scan is operated on RVIZ. RVIZ is the software that shows the result of clustering or localization by simulation. In the previous graph of laser scan, the outcome is not the targeted result because the bag is collected around the campus. If the data is collected in the room fj005. The following result can be received:



Laserscan Test

Data Analysis

From the diagram above, we can see that the cluster of white spot is the scanning result of the scan. The scanning is in the 2D form, which is used for the construction of 2D map.

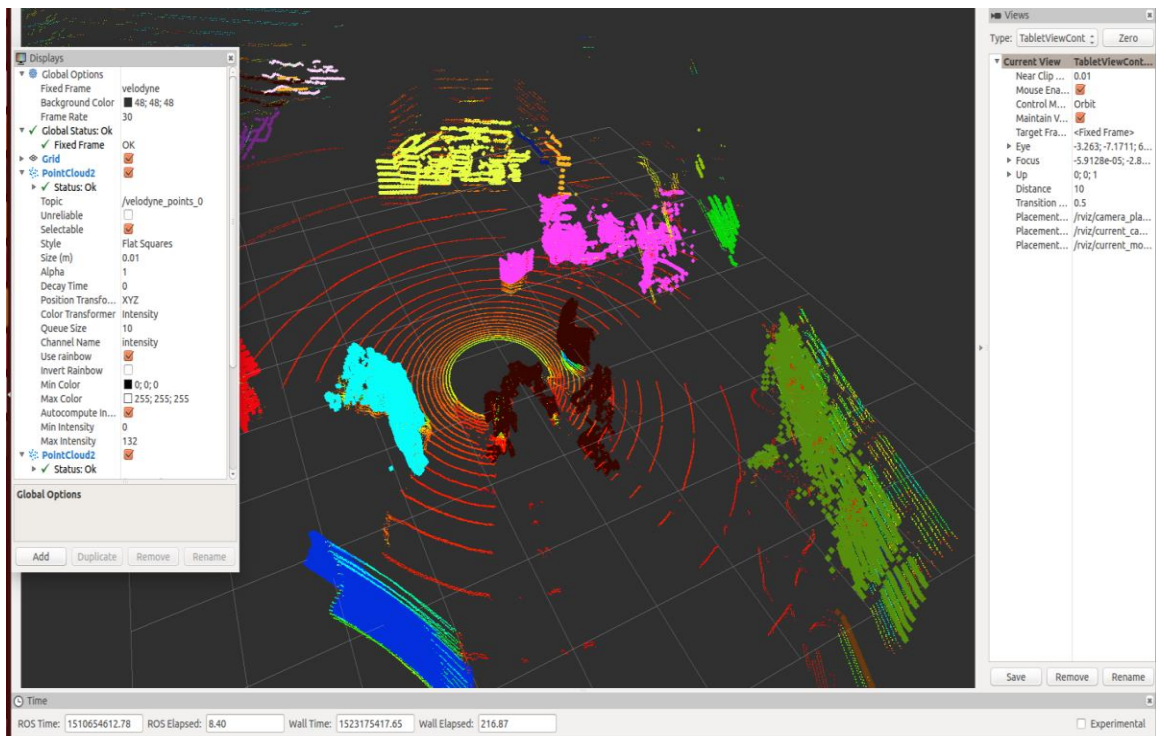
The laser scan in the form targets on the boundary of the object if the object is large. If the object is small enough, the cluster will occupy the entire object. The size of the white spot can be reduced if we change the sizing value from 0.2 m to 0.1m.

3.1.2.2 Clustering Method

Euclidean/Centroid Parts Experiments

The experiments mainly carry out in RVIZ as well. Running the bag and implementation of cpp files results in the simulation of the operation if the formula is applied.

The simulation experiment requires detailed explanation. The graph posted in the methodology part is based on the bag and PCD file collected from the environment around campus. In the final year project, the demanded environment is the static room fj001.



The coloring of the point cloud presents the distinction among cluster. Each cluster will be assigned one color. Some cluster only contains part of the clustering. The possible reason for the outcome is that the object in front of another target blocks the scanning process.

The red concentric arcs indicate that the lidar is on the operation, which represents the zone that lidar is scanning. The range of scan is 360 degrees.

Centroid Publication Test

Data Analysis

The result of cluster method

```
[ INFO] [1523175833.309380707, 1510654735.174504003]: I heard: [4271034],[1073741824],[-44054368]
[ INFO] [1523175833.309809368, 1510654735.174504003]: I heard: [4271034],[-2147483648],[-44054368]
[ INFO] [1523175833.310276780, 1510654735.174504003]: I heard: [4271034],[536870912],[-44054368]
```

The result of bounding box method

```
[ INFO] [1523175748.183126001, 1510654650.044926195]: I heard: [4271034],[536870912],[-44054368]
[ INFO] [1523175748.183347381, 1510654650.044926195]: I heard: [4271034],[0],[-44054368]
[ INFO] [1523175748.183553373, 1510654650.044926195]: I heard: [4271034],[-1073741824],[-44054368]
[ INFO] [1523175748.183766904, 1510654650.044926195]: I heard: [4271034],[-536870912],[-44054368]
[ INFO] [1523175750.174122770, 1510654652.029917151]: I heard: [4271034],[-1610612736],[-44054368]
[ INFO] [1523175750.179626672, 1510654652.029917151]: I heard: [4271034],[1610612736],[-44054368]
[ INFO] [1523175750.179760150, 1510654652.029917151]: I heard: [4271034],[0],[-44054368]
```

Cluster method and bounding box method have the similar result since bounding box is the extension of clustering. Based on the primary observation, the output is in three dimensional. On the contrary, the preset guidance is based on 2D. As the result, the receiver of the path control only receives the coordinate of x and y.

Clustering Test

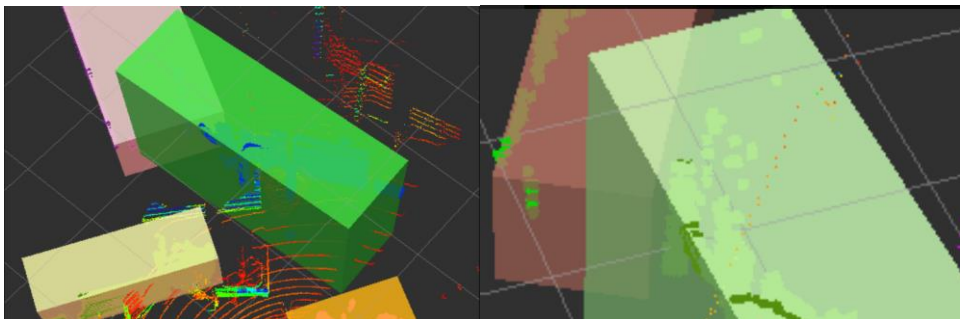
Data Analysis

There are two methods for selection, but which one is more optimal to apply in the actual process needs further investigation.

Certainly, there are several drawbacks for the bounding box method and euclidean cluster method.

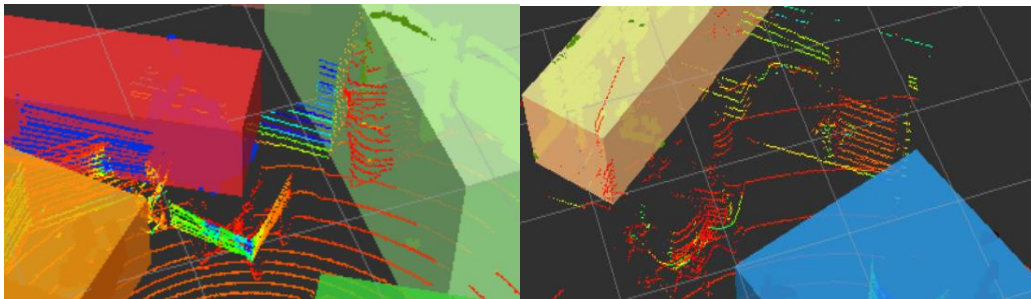
1.Overlap

In the bounding box experiment, when two clusters are adjacent to each other, bounding boxes might overlap in the running process. In comparison, Euclidean cluster has better classification between clusters.



Delay

In the experiment, both Euclidean cluster and bounding box method have the problem of delay. The problem of delay occurs if the relative position between the robot and detected objects changes drastically. Two situations are possible: robot moves at high speed or the detected objects move at high speed. The possibility that the robot shifts at high speed can be deleted because the mapping process is based on the low speed of movement. Otherwise, the map cannot be formed. Therefore, the detected object moves at high speed cause the delay. The possible reason for the delay is that centroids of cluster change in each iteration. Therefore, if the objects move, the relative distance between the robot and the calculated centroids of detected objects changes. The additional calculation requires time. As the result, the delay occurs.



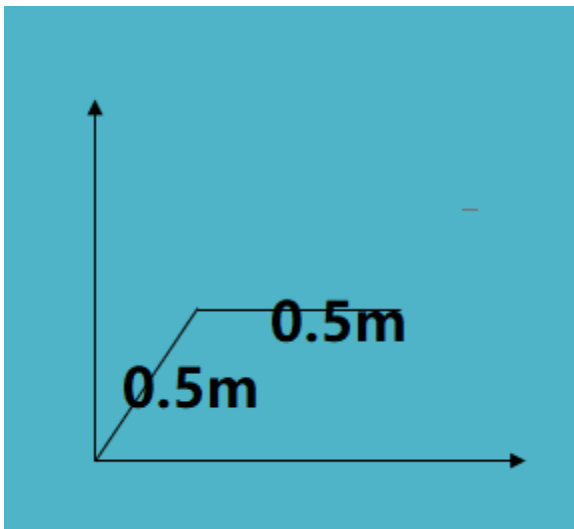
3.1.3 Path tracking and autonomous control

To examine the accuracy as well as the operability of the whole project, several experiments related to KUKA and Lidar have been conducted in FJ005. It is not possible to show the calculation process in KUKA's computer because the wires of monitor and

keyboard are not long enough to reach the moving KUKA. Instead, we can only provide the measurement and results of the experiments. The details are as followed:

1.Accuracy of KUKA and Lidar

Before conducting all the experiments, we would like to know how accurate KUKA can execute the command, that is, the accuracy of magnitude as well as the direction of KUKA's speed. A pathway has been designed to show KUKA's capability, as is shown below:



KUKA will first move 0.5m with 60 degrees forward then turn right for the next 0.5m. The speed is always 0.05m/s. This pathway can examine both the magnitude and the direction of KUKA. Furthermore, it can show whether there is the deviation from changing the direction.

After the measurement, the results are quite satisfied because the direction is almost the same as designed and there is the only about 0.5cm deviation between the final destination and desired destination, which means the error should be around 0.5%, and that is acceptable for this project.

Next, an experiment related to both KUKA and Lidar is conducted. There are four aims of this experiment:

- a. Check whether there is any delay or any problem between the communication among Lidar, external computer and KUKA.
- b. Examine the accuracy of Lidar's detection ability
- c. Show that KUKA can drive according to Lidar's signal
- d. Measure the coordinate system in Lidar for further experiment

To fulfill these missions, codes introduced in real-time speed calculation has been implemented. A one-meter square has been designed as its pathway and the code is as followed:

```
1.  obj_x[0]=0;obj_y[0]=0;
2.  obj_x[1]=1;obj_y[1]=0;
3.  obj_x[2]=1;obj_y[2]=1;
4.  obj_x[3]=0;obj_y[3]=1;
5.  obj_x[4]=0;obj_y[4]=0;
```

It is necessary to drive KUKA to the original point first because it is difficult to place KUKA to original point accurately every time. This pathway can also show the direction of x and y coordinates. During the experiment, there are plenty of problems encountered related to the communication yet we solved them. In the experiment, KUKA moved according to the designed pathway and changed its direction when it reached one midpoint. The error of the movement is less than 1cm and this may be because:

- a. The floor is not flat so there will be some influence
- b. Lidar may not be 100% accurate all the time
- c. Data transmission has some delay

Overall, the results are quite satisfying and further experiments can be planned and conducted. The video will be played in the final presentation.

3.2 Communication and connection test

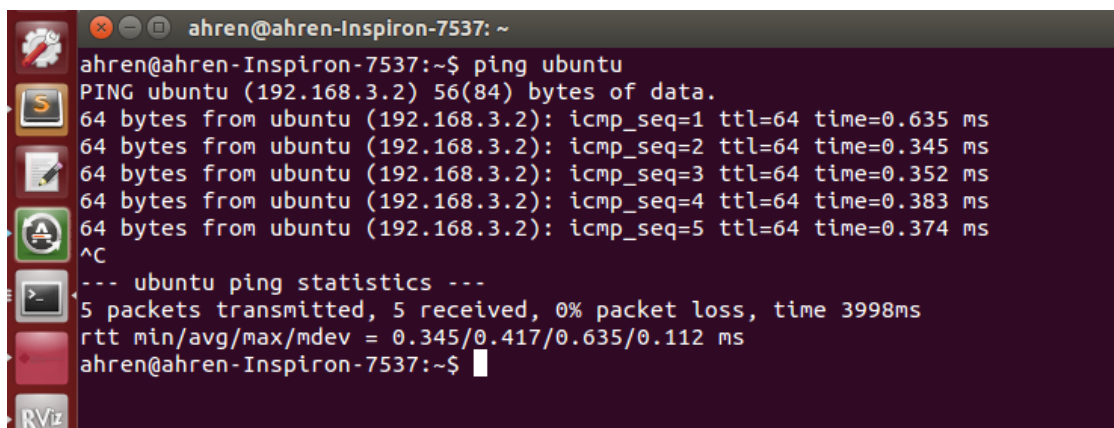
Test mainly contains three steps: ping test, topic test and final test.

3.2.1 Ping test

- i. Objectives: test the hardware that is workable to achieve multi-computer communication
- ii. Test procedure

After checking the hardware connecting is well established and computers are in right IP, the operator can open a new terminal and define the master computer by typing “ssh ahren-Inspiron-7537” Then, a new terminal should be opened to formally start the ping test. Typing “ping ubuntu” After typing “Ctrl + c”, the information passing efficiency can be checked.

- iii. Result analysis: ping testing is success without any package loss



```
ahren@ahren-Inspiron-7537: ~  
ahren@ahren-Inspiron-7537:~$ ping ubuntu  
PING ubuntu (192.168.3.2) 56(84) bytes of data.  
64 bytes from ubuntu (192.168.3.2): icmp_seq=1 ttl=64 time=0.635 ms  
64 bytes from ubuntu (192.168.3.2): icmp_seq=2 ttl=64 time=0.345 ms  
64 bytes from ubuntu (192.168.3.2): icmp_seq=3 ttl=64 time=0.352 ms  
64 bytes from ubuntu (192.168.3.2): icmp_seq=4 ttl=64 time=0.383 ms  
64 bytes from ubuntu (192.168.3.2): icmp_seq=5 ttl=64 time=0.374 ms  
^C  
--- ubuntu ping statistics ---  
5 packets transmitted, 5 received, 0% packet loss, time 3998ms  
rtt min/avg/max/mdev = 0.345/0.417/0.635/0.112 ms  
ahren@ahren-Inspiron-7537:~$
```

3.2.2 Topic test

- i. Objective: test the ros topic deliverability
- ii. Test procedure

In this part, I program a sample publisher and subscriber to test the information transport via topic. If the subscriber successfully receives the data which is sent by the publisher, it will print “kuka testing” in the terminal. Because the publisher and subscriber are easy to be written by following the roswiki, explanation is not necessary.

(<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>)

Publisher :

```
1. #include "ros/ros.h"
2. #include "std_msgs/String.h"
3. #include <sstream>
4. int main(int argc, char **argv)
5. {
6.   ros::init(argc, argv, "talker");
7.   ros::NodeHandle n;
8.   ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
9.   ros::Rate loop_rate(10);
10. int count = 0;
11. while (ros::ok())
12. {
13.   std_msgs::String msg;
14.   std::stringstream ss;
15.   ss << "wifi kuka testing " << count;
16.   msg.data = ss.str();
17.   ROS_INFO("%s", msg.data.c_str());
18.   chatter_pub.publish(msg);
19.   ros::spinOnce();
20.   loop_rate.sleep();
```

```
21. ++count;
22. }
23. return 0;
24. }
```

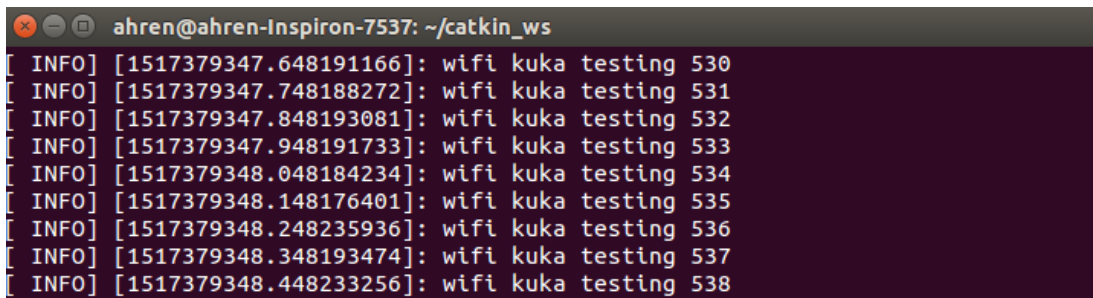
Subscriber :

```
1. #include "ros/ros.h"
2. #include "std_msgs/String.h"
3. void chatterCallback(const std_msgs::String::ConstPtr& msg)
4. {
5.     ROS_INFO("I heard: [%s]", msg->data.c_str());
6. }
7. int main(int argc, char **argv)
8. {
9.     ros::init(argc, argv, "listener");
10.    ros::NodeHandle n;
11.    ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
12.    ros::spin();
13.
14.    return 0;
15. }
```

Cmake list: simply add these few lines to the bottom of CMakeLists.txt:

1. `include_directories(include ${catkin_INCLUDE_DIRS})`
2. `add_executable(talker src/talker.cpp)`
3. `target_link_libraries(talker ${catkin_LIBRARIES})`
4. `add_executable(listener src/listener.cpp)`
5. `target_link_libraries(listener ${catkin_LIBRARIES})`

iii. test result



```
ahren@ahren-Inspiron-7537: ~/catkin_ws
[ INFO] [1517379347.648191166]: wifi kuka testing 530
[ INFO] [1517379347.748188272]: wifi kuka testing 531
[ INFO] [1517379347.848193081]: wifi kuka testing 532
[ INFO] [1517379347.948191733]: wifi kuka testing 533
[ INFO] [1517379348.048184234]: wifi kuka testing 534
[ INFO] [1517379348.148176401]: wifi kuka testing 535
[ INFO] [1517379348.248235936]: wifi kuka testing 536
[ INFO] [1517379348.348193474]: wifi kuka testing 537
[ INFO] [1517379348.448233256]: wifi kuka testing 538
```

Data can be successfully passed from PC to kuka.

3.2.3 Project test

- i. Test objective: Evaluate the system that can meet the accuracy and stability requirement.
- ii. Test procedure

By using

Rostopic list

Rostopic echo

```
* /tf [tf2_msgs/TFMessage] 3 publishers
* /config/twist_filter [runtime_manager/ConfigTwistFilter] 1 publisher
* /brake cmd [runtime_manager/brake cmd] 1 publisher
* /ndt_pose [geometry_msgs/PoseStamped] 1 publisher
* /diagnostics [diagnostic_msgs/DiagnosticArray] 1 publisher
* /velodyne_points_0 [sensor_msgs/PointCloud2] 1 publisher
* /localizer_pose [geometry_msgs/PoseStamped] 1 publisher
* /config/car_dpm [runtime_manager/ConfigCarDpm] 1 publisher
```

The data type of the output “ndt_pose” can be found. According to ros wiki tutorials material, the subscriber sample can be written as follow

```
1. #include "ros/ros.h"
2. #include "geometry_msgs/PoseStamped.h"
3. void ndt_poseCallback(const geometry_msgs::PoseStamped::ConstPtr& msg)
4. {
5.   ROS_INFO("I heard: [ndt_matching pose]");
6. }
7. int main(int argc, char **argv)
8. {
9.   ros::init(argc, argv, "listener");
10.  ros::NodeHandle n;
11.  ros::Subscriber sub = n.subscribe("ndt_pose", 1000, ndt_poseCallback);
12.  ros::spin();
13.  return 0;
14. }
```

Initially, we test the connection between Euclidean cluster and path-finding function. The function will only present The result is shown as follows:

The data can pass thorough rosout to ubuntu and active the subscriber in the ubuntu. If the subscriber successfully receives the data which is sent by the publisher, it will print “ I heard ndt_matching pose” in the terminal.

In the real installation, the multi-computer connection is blocked with a lot of problems because of the unfamiliar with the roscore and master in multi-computer communication. There is only a little information that can be found on the internet about it, so I have to solve these problems by myself without enough past references.

If all of the members integrate their work together, the testing is required as well.

Initially, we test the connection between Euclidean cluster and path-finding function .The result is shown as follows:

```
youbot@ubuntu: ~/youbot_nav
block_x7.73349
[ INFO] [1523340306.620976001, 1510654990.999860941]: I heard: Henry
block_x9.12446
[ INFO] [1523340306.621614141, 1510654990.999860941]: I heard: Henry
block_x-0.991088
[ INFO] [1523340306.622268555, 1510654990.999860941]: I heard: Henry
block_x9.34792
[ INFO] [1523340306.622907184, 1510654990.999860941]: I heard: Henry
block_x7.11291
[ INFO] [1523340306.623534569, 1510654990.999860941]: I heard: Henry
block_x11.2091
[ INFO] [1523340306.624245973, 1510654990.999860941]: I heard: Henry
block_x10.3453
[ INFO] [1523340306.624904158, 1510654990.999860941]: I heard: Henry
block_x11.4787
[ INFO] [1523340306.625600057, 1510654990.999860941]: I heard: Henry
block_x-9.44616
[ INFO] [1523340306.626217874, 1510654990.999860941]: I heard: Henry
block_x7.59128
[ INFO] [1523340306.626840859, 1510654990.999860941]: I heard: Henry
block_x3.85708
[ INFO] [1523340306.627463914, 1510654990.999860941]: I heard: Henry
block_x-6.9431
```

The connection between cluster and pathfinding is correct.

```
block_x-0.726277
block_y0.0562454
[ INFO] [1523342954.071195054, 1510654950.566981032]: I heard: Henry
block_x-2.65855
block_y0.118644
[ INFO] [1523342954.071544890, 1510654950.566981032]: I heard: Henry
block_x-5.43189
block_y0.0418869
[ INFO] [1523342954.071943895, 1510654950.566981032]: I heard: Henry
block_x-7.23344
block_y0.0897913
[ INFO] [1523342954.072310563, 1510654950.566981032]: I heard: Henry
block_x-5.51811
block_y0.0601897
[ INFO] [1523342954.072669827, 1510654950.566981032]: I heard: Henry
block_x-3.6921
block_y0.2167
[ INFO] [1523342954.073030907, 1510654950.566981032]: I heard: Henry
block_x-7.73088
block_y-0.376084
[ INFO] [1523342954.073384165, 1510654950.566981032]: I heard: Henry
block_x6.5703
block_y0
```

Then, we add the y value of the centroid of block.

However, when both clustering and localization are applied, the pathfinding function can only present or receive the data from localization.

```
youbot@ubuntu: ~/youbot_nav
vy-0.0328798
[ INFO] [1523343096.184316357, 1510654965.039671287]: I heard: ndt_matching pose
x3.61795
y7.71893
vx0.189737
vy-0.0632456
[ INFO] [1523343096.185355875, 1510654965.039671287]: I heard: ndt_matching pose
x3.35577
y9.32933
vx0.141421
vy-0.141421
[ INFO] [1523343096.186253756, 1510654965.039671287]: I heard: ndt_matching pose
x3.23644
y10.7031
vx0.141421
vy-0.141421
[ INFO] [1523343096.187375757, 1510654965.039671287]: I heard: ndt_matching pose
x3.27129
y11.8089
vx0.141421
vy-0.141421
```

iii. Test result

The subscriber successfully receive the data which is sent by the publisher , it prints “ I heard ndt_matching pose” in the terminal.

```
youbot@ubuntu: ~/catkin_ws
[ INFO] [1517379796.706524979, 1510654638.139648618]: I heard: [ndt_matching pos
[ INFO] [1517379796.819383388, 1510654638.261266622]: I heard: [ndt_matching pos
[ INFO] [1517379796.904884423, 1510654638.340915823]: I heard: [ndt_matching pos
[ INFO] [1517379797.016660507, 1510654638.453701043]: I heard: [ndt_matching pose]
[ INFO] [1517379797.104254671, 1510654638.542309190]: I heard: [ndt_matching pose]
[ INFO] [1517379797.218406765, 1510654638.656106136]: I heard: [ndt_matching pose]
[ INFO] [1517379797.305619390, 1510654638.746740952]: I heard: [ndt_matching pose]
[ INFO] [1517379797.409671376, 1510654638.843667692]: I heard: [ndt_matching pose]
[ INFO] [1517379797.502304322, 1510654638.938998821]: I heard: [ndt_matching pose]
[ INFO] [1517379797.627697355, 1510654639.063158777]: I heard: [ndt_matching pose]
[ INFO] [1517379797.707661297, 1510654639.143632650]: I heard: [ndt_matching pose]
[ INFO] [1517379797.799060628, 1510654639.236061701]: I heard: [ndt_matching pose]
```

a. Content of the data

The following is a set of output data

Seq: sequence means the data is the 128th group. (In order to improve the localization performance, the frequency should be high.)

Position: According to the coordinate system, it provides the coordinate point.

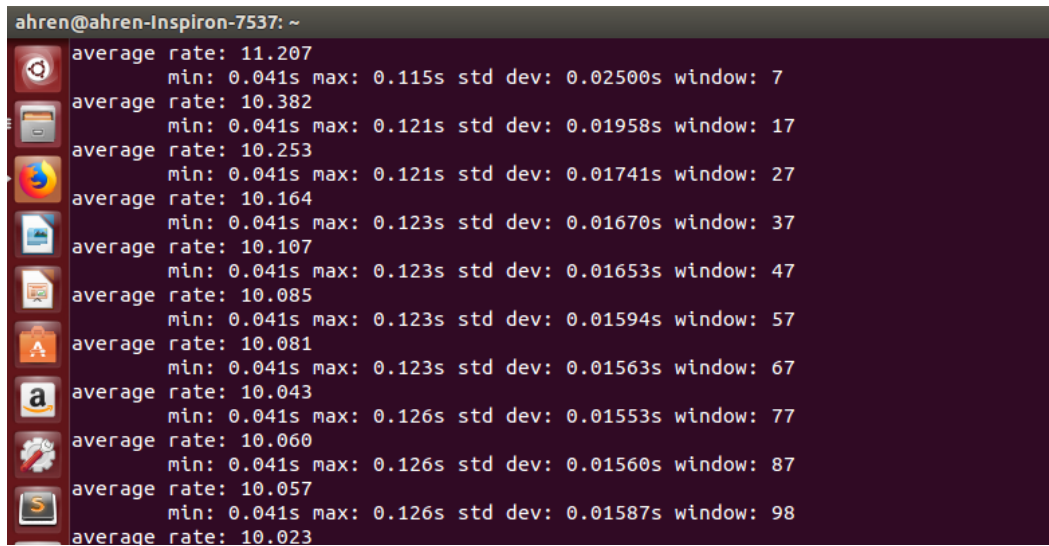
Orientation: It corresponds to the movement direction. According to the past velocity, the direction can be predicted. The data in the part is indicated by using quaternion which is a super complex number. Because we only use the position in our project, orientation and quaternion is negligible.

```
header:
  seq: 128
  stamp:
    secs: 1510654617
    nsecs: 872335104
  frame_id: /map
pose:
  position:
    x: 1.15532112122
    y: -0.0974764227867
    z: 0.039402961731
  orientation:
    x: -0.00699552956087
    y: -0.0147749852207
    z: 0.838270739626
    w: 0.545009109524
```

b. Frequency

Because path tracking part needs to receive two kinds of data: localization pose and center barrier pose. All these two data should corporate and change every moment. So, these two kinds of data should have the same frequency and the frequency should match the computer power. By typing the following command in terminal, the topic frequency can be shown.

1. rostopic hz -ndt_pose

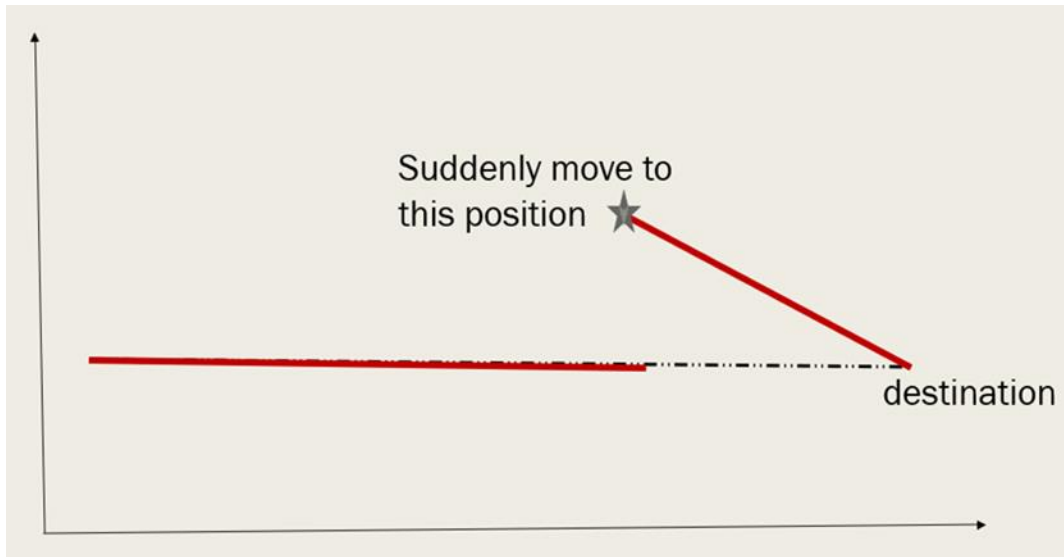
A terminal window with a dark background and a sidebar of application icons on the left. The terminal displays the output of the 'rostopic hz' command for the 'ndt_pose' topic. The output shows a series of statistics for each of the 10 published messages, including average rate, minimum/maximum values, standard deviation, and window size.

```
ahren@ahren-Inspiron-7537: ~  
average rate: 11.207  
min: 0.041s max: 0.115s std dev: 0.02500s window: 7  
average rate: 10.382  
min: 0.041s max: 0.121s std dev: 0.01958s window: 17  
average rate: 10.253  
min: 0.041s max: 0.121s std dev: 0.01741s window: 27  
average rate: 10.164  
min: 0.041s max: 0.123s std dev: 0.01670s window: 37  
average rate: 10.107  
min: 0.041s max: 0.123s std dev: 0.01653s window: 47  
average rate: 10.085  
min: 0.041s max: 0.123s std dev: 0.01594s window: 57  
average rate: 10.081  
min: 0.041s max: 0.123s std dev: 0.01563s window: 67  
average rate: 10.043  
min: 0.041s max: 0.126s std dev: 0.01553s window: 77  
average rate: 10.060  
min: 0.041s max: 0.126s std dev: 0.01560s window: 87  
average rate: 10.057  
min: 0.041s max: 0.126s std dev: 0.01587s window: 98  
average rate: 10.023
```

3.3 Field test

3.3.1 Self-adjustment test

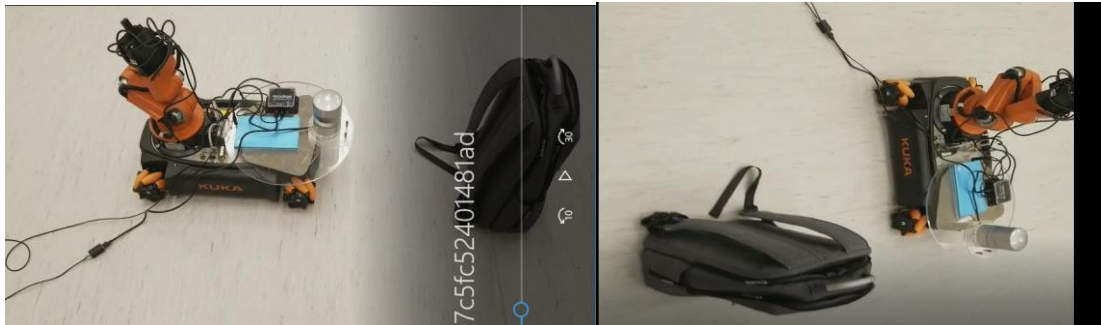
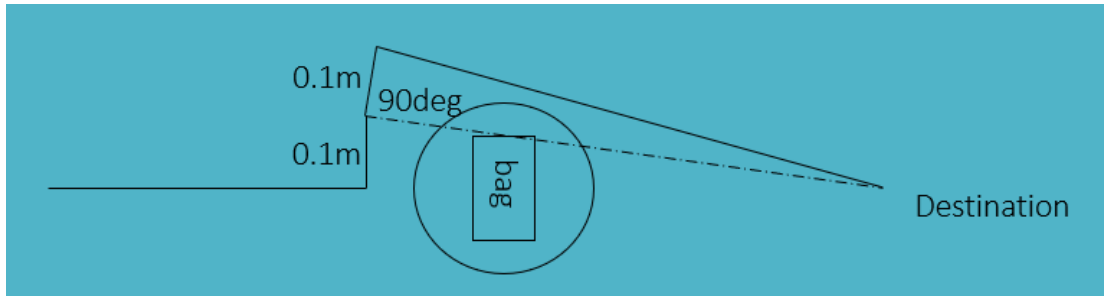
To show that our project can overcome some interruptions from the outside environment, an experiment has been designed to examine whether KUKA can drive to the destination when it is suddenly moved to another place. The main code is similar to the code in real-time speed calculation because, in that design, the speed direction is always towards the destination, which fulfills the requirement in this experiment. The pathway and the sudden movement, which position is random, are shown in the picture below:



The measurement shows that after the sudden movement, KUKA still headed towards the destination and stopped when it reached the destination. The error is less than 0.5cm, which means our design has a relatively high self-adjustment ability.

3.3.2 Obstacle avoidance test

Another experiment is designed to test whether the project is able to avoid an obstacle. The code is similar to the previous one. However, the frequency is set to 0.05hz and total speed is set to 0.05m/s, which means KUKA will judge whether to change the speed direction every 10cm. This can make it easy to observe and measure when and where KUKA changes its direction to conduct the error analysis. The pathway is shown below.



This pathway is almost the same compared to the desired one. However, KUKA should turn left around 2cm before the current pathway. The reason for this error may be:

- a. There exists some delay time among KUKA, Lidar and external computer.
- b. There may be some interference during Lidar's detection.
- c. The center point of the block is not quite accurate.

Despite of these errors, we can still conclude that the experiment of self-adjustment and avoiding obstacles are successful.

3.4 Problems occurred

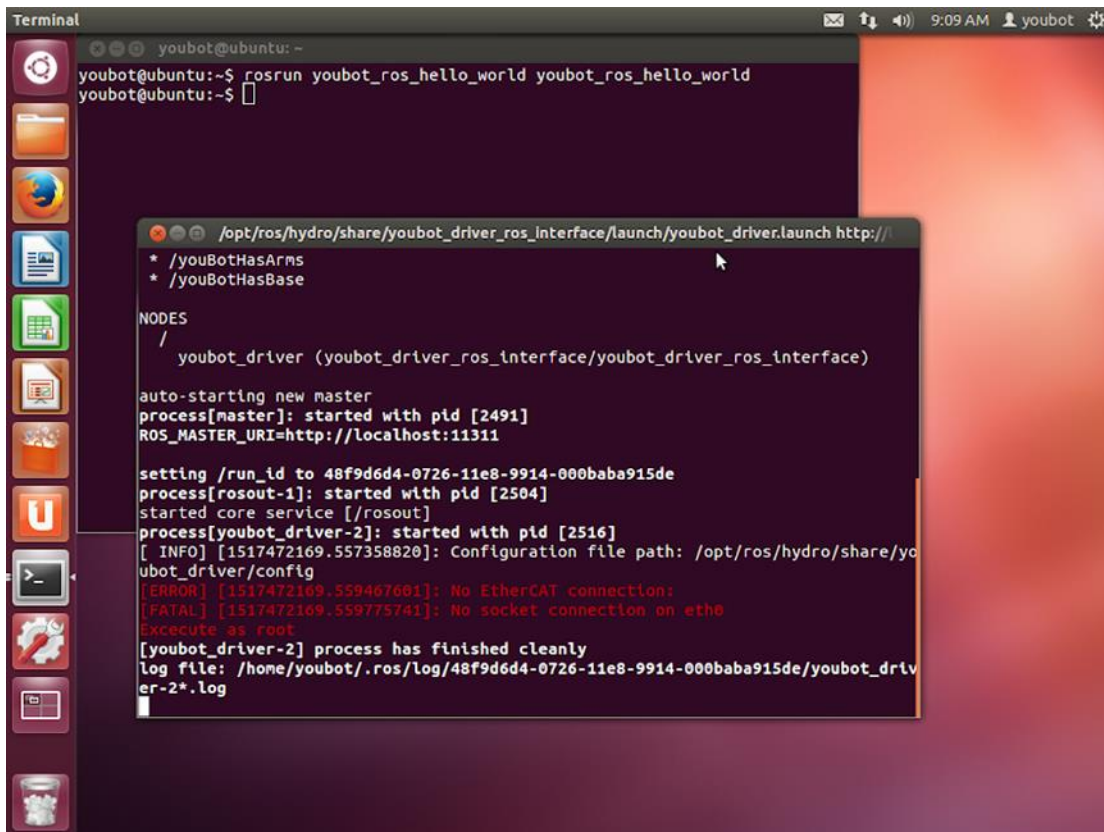
3.4.1 Problems about KUKA driver in KUKA's computer

During the connection between the external computer and KUKA, a lot of problems have been encountered because of the communication and permission. One of the problems is that the driver cannot start after KUKA connects with the switch. To solve it, many actions related to communication should be taken and here just introduces what should be modified in KUKA's computer except for IP address.

Originally, there is only one cable that connected with KUKA computer, which is shown in the picture



However, in this project, KUKA also need to connect with the switch so that it can receive information from the external computer, which means there need two cables. With these two cables, KUKA's computer will be confused which one should give the command to drive and which one should receive information from the external computer so KUKA's driver cannot start by command. The error shows in the following picture:

A screenshot of a Linux terminal window. The top terminal shows the command 'roslaunch youbot_driver youbot_driver.launch' being executed. A second terminal window is overlaid, showing the launch process. It starts with 'auto-starting new master', then 'process[rosout-1]: started with pid [2504]', and 'process[youbot_driver-2]: started with pid [2516]'. It then shows an error: '[ERROR] [1517472169.559467601]: No EtherCAT connection: [FATAL] [1517472169.559775741]: No socket connection on eth0'. The process then finishes cleanly with the message '[youbot_driver-2] process has finished cleanly' and a log file path is provided.

```
Terminal
youbot@ubuntu: ~
youbot@ubuntu:~$ roslaunch youbot_driver youbot_driver.launch
youbot@ubuntu:~$

/opt/ros/hydro/share/youbot_driver_ros_interface/launch/youbot_driver.launch http://localhost:11311
* /youBotHasArms
* /youBotHasBase

NODES
/
  youbot_driver (youbot_driver_ros_interface/youbot_driver_ros_interface)

auto-starting new master
process[master]: started with pid [2491]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 48f9d6d4-0726-11e8-9914-000baba915de
process[rosout-1]: started with pid [2504]
started core service [/rosout]
process[youbot_driver-2]: started with pid [2516]
[ INFO] [1517472169.557358820]: Configuration file path: /opt/ros/hydro/share/youbot_driver_ros_interface/config
youbot_driver/config
[ERROR] [1517472169.559467601]: No EtherCAT connection:
[FATAL] [1517472169.559775741]: No socket connection on eth0
Execute as root
[youbot_driver-2] process has finished cleanly
log file: /home/youbot/.ros/log/48f9d6d4-0726-11e8-9914-000baba915de/youbot_driver-2*.log
```

Therefore, it is necessary to modify the config file so that KUKA knows the function of each ethernet. Furthermore, we need to give KUKA the permission command driver by the following command:

1. `sudo setcap cap_net_raw+ep /opt/ros/indigo/lib/youbot_driver_ros_interface/youbot_driver_ros_interface`
2. `sudo ldconfig /opt/ros/indigo/lib`

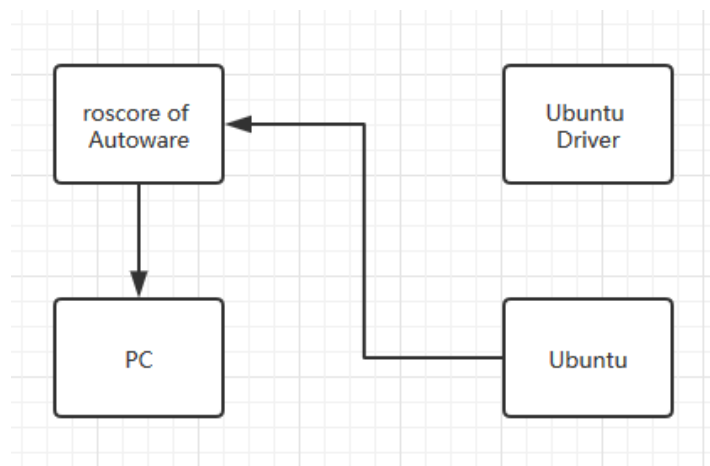
3.4.2 Confliction between youbot_driver and Autoware

Problem 1

Confliction between youbot_driver and Autoware

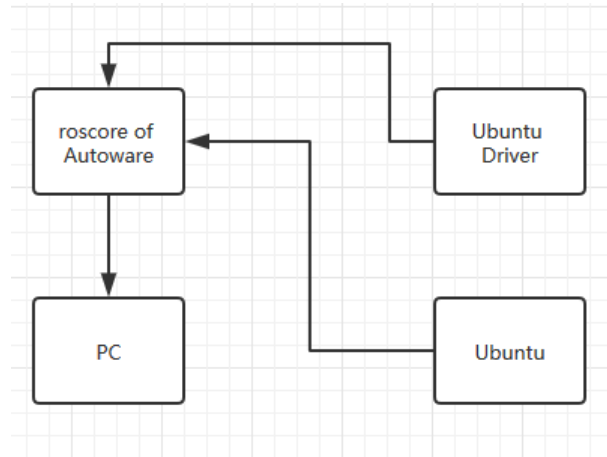
The situation is that the topic can be sent to Kuka computer. Meanwhile, the calculated velocity and position can be printed in the terminal (it means the raw data is readable and changeable). However, the youbot cannot move.

Finally, I find that the problem is a roscore and master problem.



(the figure shows the connection between each part, the ubuntu youbot,driver cannot connect with PC.) Before opening the driver, I only export controlling program but did not export the youbot_driver (export ROS_MASTER_URI=http:// ahren-Inspiron-7537:11311). So, it cannot connect with master roscore in pc and cannot be activated (after exporting driver's roscore, it can connect with PC. The PC can work)

Finally, the standard opening sequence or process is provided as follow.



Frist, the Autoware should be opened because the roscore in Autoware works as a master roscore.

1. `$ ssh ahren-Inspiron-7537`
2. `$ cd $HOME/Autoware/ros`
3. `$ ./run`

Then,we come to kuka computer. Frist, export the ubuntu driver and then export the Path tracking and control part.

1. `export ROS_MASTER_URI=http:// ahren-Inspiron-7537:11311`
2. `roslaunch youbot_driver_ros_interface youbot_driver.launch`

Finally, open a new terminal and it can work (refer to the appendix video).

1. `export ROS_MASTER_URI=http:// ahren-Inspiron-7537:11311`
2. `cd $HOME/youbto`

3. `source devel/setup.bash`
4. `roslaunch sam1 sam1`

3.4.3 Conflicting between youbot_driver and LiDAR driver

Conflicting between youbot_driver and LiDAR driver

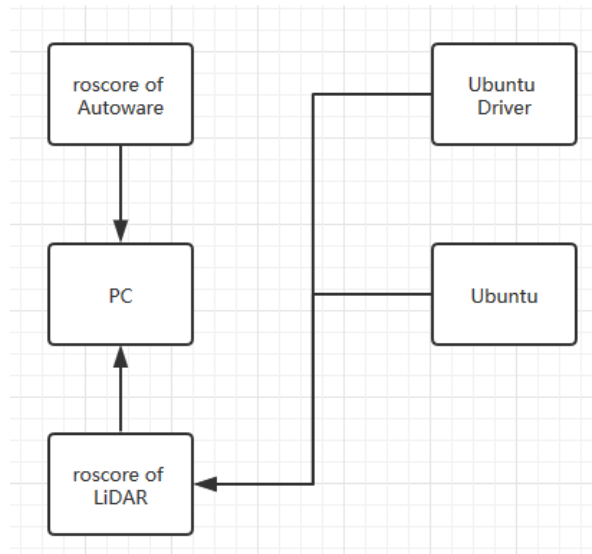
LiDAR is a sophisticated device and it also needs a driver to send scanning data to PC.

The problem occurs when integrating LiDAR, PC, and ubuntu is in process. In problem 1, I have mentioned that Autoware needs to be opened first and its roscore works as a master. However, the situation is that if I open Autoware first then open the LiDAR driver, the LiDAR cannot send data to my PC. If LiDAR driver is opened first, the LiDAR can send data to my computer but the ubuntu driver cannot work.

In background part, some operation rules about roscore are concluded.

1. first come first served
2. One computer only can own one roscore in localhost.

The ubuntu driver cannot work because LiDAR driver is opened first and the ubuntu driver's roscore is exported to LiDAR roscore. The connection between Autoware and ubuntu driver is broken. So, the ubuntu driver cannot be active by topic “ndt_pose”



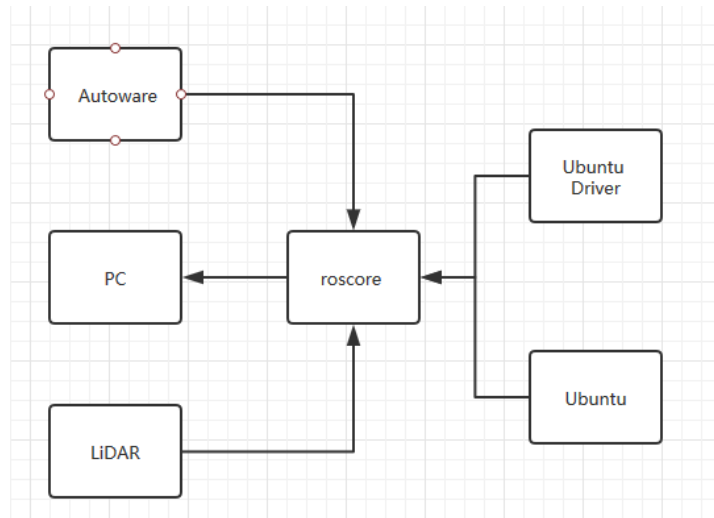
In order to solve the problem, we need to try to let the roscore in Autoware work as master. After trying many times, the idea that focuses on Autoware roscore is abandoned. By chance, I find that if roscore is opened before LiDAR drive and Autoware, they can share the roscore. In this way, the problem is easier to be solved.

Then the standard opening sequence or process is provided as follow.

Frist, the independent roscore should be opened firstly because the roscore works as a master roscore. Then open the LiDAR driver.

1. `$ ssh ahren-Inspiron-7537`
2. `$ roscore`
3. `$ cd 15_velodyne/`

4. `$ ./runVelodyne.sh`



After opening the LiDAR driver, we need to open the RVIZ and change the frame to velodyne. Then, add point cloud in topic. If the LiDAR successfully send data to PC, we can continue our process-open the Autoware platform.

1. `$ cd $HOME/Autoware/ros`

2. `$ ./run`

Then, come to kuka computer. First, we need to export the ubuntu driver and then export the Path tracking and control part.

3. `export ROS_MASTER_URI=http:// ahren-Inspiron-7537:11311`

4. `roslaunch youbot_driver_ros_interface youbot_driver.launch`

Then, open a new terminal and it works.

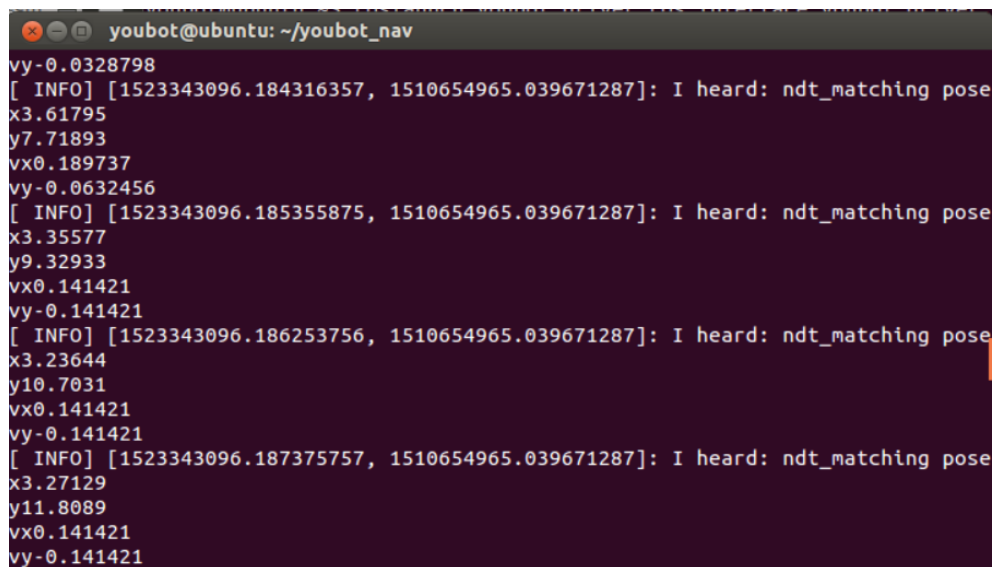
5. `export ROS_MASTER_URI=http:// ahren-Inspiron-7537:11311`

6. `cd $HOME/youbot`
7. `source devel/setup.bash`
8. `roslaunch sam1 sam1`

3.4.4 Discordance of subscriber

In the graph below , only information of robot velocity and current location are shown.

The function does not receive the information of centroid of blockade.



```
youbot@ubuntu: ~/youbot_nav
vy-0.0328798
[ INFO] [1523343096.184316357, 1510654965.039671287]: I heard: ndt_matching pose
x3.61795
y7.71893
vx0.189737
vy-0.0632456
[ INFO] [1523343096.185355875, 1510654965.039671287]: I heard: ndt_matching pose
x3.35577
y9.32933
vx0.141421
vy-0.141421
[ INFO] [1523343096.186253756, 1510654965.039671287]: I heard: ndt_matching pose
x3.23644
y10.7031
vx0.141421
vy-0.141421
[ INFO] [1523343096.187375757, 1510654965.039671287]: I heard: ndt_matching pose
x3.27129
y11.8089
vx0.141421
vy-0.141421
```

The possible reason for the discordancy might be the frequency used for the subscriber and publisher.

In the general, the format of publisher is shown as follows:

```
ros::Publisher advertise(const std::string& topic, uint32_t queue_size, bool latch = false);
```

Queue_size is the size of the outgoing message queue, which can be considered as the frequency of data published. Subscriber follows the same format.

For ndt mapping, the frequency is 1000 for queue_size in publisher and subscriber. For centroid, the frequency is 10 for queue_size in publisher and subscriber. If the frequency of subscriber of the centroid is 1000, the result is shown.

```
[ INFO] [1523342796.730621158, 1510654945.416209350]  
x2.01087  
y1.31498  
block_x-2.05371  
block_y0.106286  
vx0  
vy0.2
```

All the desired data are shown. The possible reason for the discoordinance might be the low processing speed of kuka, which means that kuka robot cannot manage too much data instantly. Because the frequency of ndt_mapping is high compared with centroid subscriber and publisher, kuka robot has higher possibility to manage the data from ndt mapping than the data published from euclidean cluster.

Discussion and conclusion

In conclusion, all the fundamental functions for unmanned indoor drive are basically fulfilled and experiment results are quite satisfied after integration. To be specific, several missions as followed have been accomplished in this project.

Localization is achieved by using `ndt_localization` which is contained in Autoware platform. Roughly, the accuracy and update frequency fulfill the requirement.

For the part of clustering, edges of objects are successfully detected. The corresponding centroids are published and received by path tracking function.

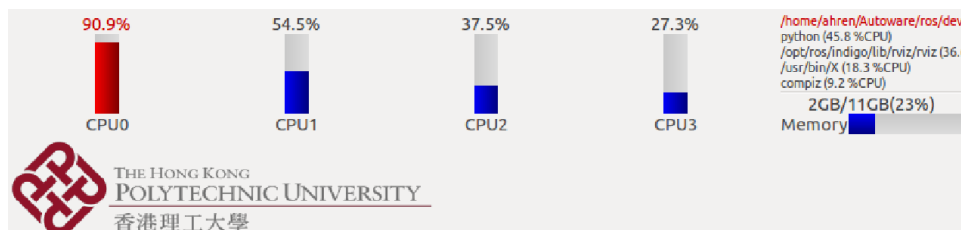
For path tracking and autonomous control, simple path planning is designed and tested according to the parameters from cluster and localization. Real-time speed control and obstacle avoidance have been achieved to directly control KUKA as well.

By using Switch and LAN, whole multi-computer communication can be stabilized and accurately achieved, which means integration is successful and all the above functions can be utilized on KUKA.

Future work

a. Further integration

Because of the time limitation, I only can use multi-computer communication as a compromise. If someone wants to continue work on this project, focusing on integrating all these parts into one computer to avoid using network cable might be a good direction. Before starting integrating, the computer in kuka should be updated because of the algorithm requirement.



(the figure shows my computer power load when running the localization part. If someone wants to integrate all these three parts into one computer, the computer should be strong enough to calculate these data.)

b. Simplify Autoware

In localization part, simplifying the Autoware platform might be another direction. Because the platform is developed for autonomous vehicle and it is a little over-quality, I want to simplify it to decrease the computer power required.

c. Take full advantages of the output

In the output part, orientation data is not used, so in the future, the project can be further improved by containing the direction part to get more accurate localization coordinate point.

d. Add more hardware to improve localization performance

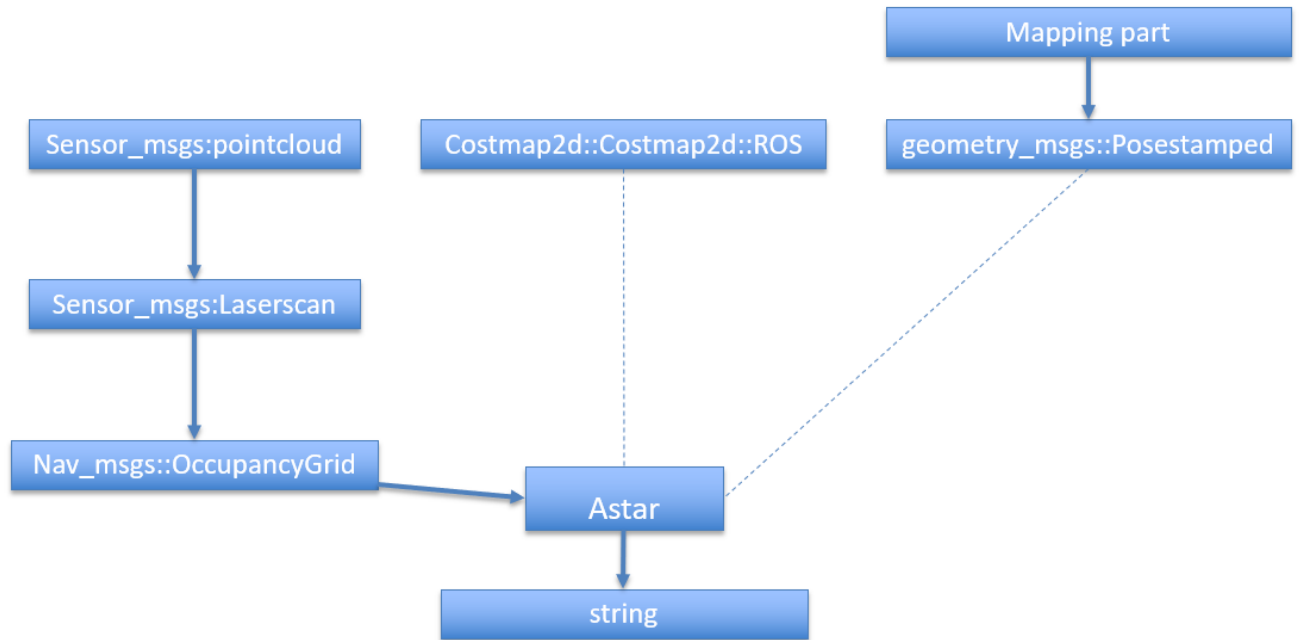
In the code of `ndt_mapping` and `ndt_matching` some addition hardware was mentioned such as imu and odom. In this project, we only use LiDAR to provide environment information. In the future, we can focus on cooperate different sensors to achieve better localization performance.

e. Guidance

The future work of the guidance is to find out the shortest path between starting point and destination that is randomly selected.

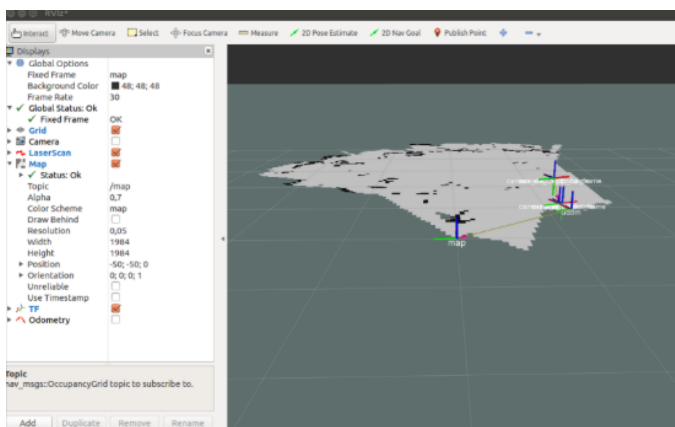
In the previous schedule, the timetable is too tight for the completion of two methods. Therefore, we choose to complete the second method instead of the first one.

Actually, part of the first method is finished. We have finished up the transformation from 3D to 2D. The extra process is required.



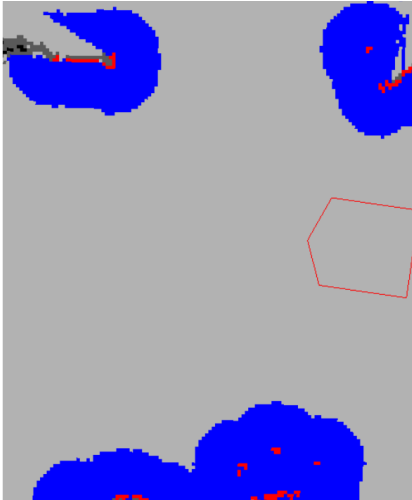
The shortest path is achieved by applying Astar. As mentioned before, Astar needs the information from occupancy grid map and costmap. Occupancy grid is the format that transforms the map into the grid form. The grid form is crucial to the storage of data. In each grid, the grid stores the information of the estimated distance from current grid to the destination grid, which is labeled as h . The distance from the starting grid to the current grid, which is stored in g . The combination of g and h is f . The grid also stores f .

The optimal outcome of occupancy map is shown in the form in this way:



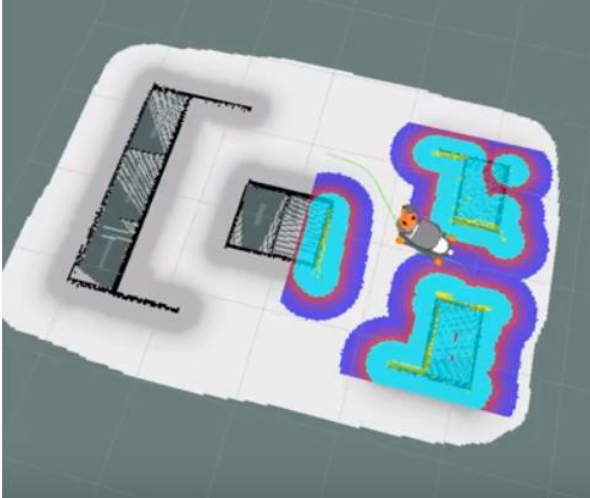
The cost map is added on the grid map. Cost map is required for the blockade avoidance.

At the blockade scanned in the grid map, the additional area, or cost area, should be placed around the blockade.



The red area on the costmap is the blockade. The blue area is the additional cost near to blockade. The blue area serves the same purpose as the area R in the pathfinding function. The cost area is used for robot turning. If there has no cost area, the robot may directly hit the edge of the blockade during the change of direction. As the result, a star implemented for the guide will demonstrate the shortest path based on the information given by gridmap and costmap.

The desired outcome should be the following:



The green line is the assigned path from the starting point to the destination. The blue area is the cost area. The red area is the edge of blockade.

f. Avoid more obstacles and reduce costs

Obstacle avoidance can also be further improved. In the real situation, obstacles may be close to each other or in some random shape so a better design is required for these complex situations. Furthermore, there are moving obstacles like human and dogs in real life, which needs prediction and decision-making. If these problems can be solved, it is not difficult to optimize path planning and reduce the costs for each obstacle.

Ref

Saarinen, J., Andreasson, H., Stoyanov, T., & Lilienthal, A. J. (2013, November). Normal distributions transform Monte-Carlo localization (NDT-MCL). In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on* (pp. 382-389). IEEE.

Cai, Z.S, Hong, B.R, & Wei, Z.H. (2005). Use LiDAR scanning to achieve robot localization27(5), 414-419.

Wolcott, R. W., & Eustice, R. M. (2015, May). Fast LIDAR localization using multiresolution Gaussian mixture maps. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on* (pp. 2814-2821). IEEE

Thrun, S. (2002). Probabilistic robotics. *Communications of the ACM*, 122-1847.

Fang, E. F., Scheibye-Knudsen, M., Jahn, H. J., Li, J., Ling, L., Guo, H., ... Ng, T. B. (2015). A research agenda for aging in China in the 21st century. *Ageing Research Reviews*, 24(Pt B), 197–205. <http://doi.org/10.1016/j.arr.2015.08.003>

Backplanes, O., & Cables, C. IEEE Standard for Ethernet 802.3-2012 Clause 9.1.

O. (2014). About ROS. Retrieved from <http://www.ros.org/about-ros/>

O. (2014). Topic. Retrieved from <http://wiki.ros.org/Topics>

Novi Commentarii academiae scientiarum Petropolitanae 20, 1776, pp. 189–207

Takahashi, S., Kaneko, R., Ishikawa, A., Ikenoue, M., & Mita, F. (1999). Population projections for Japan: Methods, assumptions and results. *Review of Population and Social Policy*, 8, 75-115.

Population structure and ageing. (2017, June). *Eurostat*. Retrieved April 17, 2018, from http://ec.europa.eu/eurostat/statistics-explained/index.php/Population_structure_and_ageing#Further_Eurostat_information

Mathew, K. (2015, June 10). Global Autonomous Car Technology Market Poised to Grow Over 10% Through 2035 Says TechSci Research. *PR Newswire*. Retrieved April 17, 2018, from https://global.factiva.com/ha/default.aspx#./!/?&_suid=152395862214507133698493140772

Wu, J. S., Zheng, W. S., Lai, J. H., & Suen, C. Y. (n.d.). Euler Clustering on Large-Scale Dataset(Vol. 14, Latex Class Files, Tech.). Retrieved November 25, 2017, from <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8030074>

Khankanapoka, K., & Chinnasarn, K. (n.d.). Path Finding of 2D&3D Game Real-Time Strategy With Depth Direction A*Algorithm For Multi-layer (Tech.).

Bertozzi, M., Broggi, A., Fascioli, A., & Nichele, S. (2000). Stereo vision-based vehicle detection. In *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE* (pp. 39-44). IEEE.

“PID Control,” in *The Control Handbook*, W. S. Levine, Ed. Piscataway, NJ: IEEE Press, 1996, pp. 198–209.

Kiam Heong Ang, Chong, G. and Yun Li (2005). PID control system analysis, design, and technology. IEEE Transactions on Control Systems Technology, 13(4), pp.559-576.

F. G. Shinskey, Feedback Controllers for the Process Industries. New York: McGraw-Hill, 1994.

Appendix