UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de
Ingeniería y Sistemas de Telecomunicación

PROYECTO FIN DE GRADO

A ROS NODE FOR PROCESSING IMAGES AND PROVIDING ROBOTS WITH LOCATION CAPABILITIES

WENJUN DUAN

Grado en Ingeniería Telemática
Julio 2018

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: A ROS NODE FOR PROCESSING IMAGES AND PROVIDING ROBOTS WITH LOCATION CAPABILITIES

AUTOR: WENJUN DUAN

TITULACIÓN: GRADO EN INGENIERÍA TELEMÁTICA

TUTOR: JOSÉ FERNÁN MARTÍNEZ ORTEGA

DEPARTAMENTO: DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: MARÍA PILAR OCHOA PÉREZ

TUTOR: JOSÉ FERNÁN MARTÍNEZ ORTEGA

SECRETARIO: VICENTE HERNÁNDEZ DÍAZ

Fecha de lectura:

Calificación:

El Secretario,

Resumen

Este Proyecto Final de Grado tiene como objetivo desarrollar un algoritmo de auto-localización para robots basado en el procesamiento de imágenes. Se han analizado varios algoritmos a lo largo de este proyecto y finalmente, uno que combina dos de ellos, uno que procesa imágenes y otro que maneja distancias a obstáculos, ha sido seleccionado y desarrollado y desplegado en una prueba de concepto para verificar su factibilidad y su rendimiento. Brevemente, la aplicación que se ha desarrollado y probado en un entorno simulado primero permite que el robot reconozca en qué habitación se encuentra a través de imágenes que el robot toma. Una vez que determina la habitación en la que se encuentra, descarga el mapa correspondiente y determina su posición exacta en la habitación midiendo las distancias a los obstáculos.

Las características de las imágenes se extraen con el algorithm Scale Invariant Feature Transform (SIFT) y se describen con un vector multidimensional altamente distintivo que hace que las características sean invariables con los cambios en la iluminación, la escala y el ruido. El uso de este algoritmo permitirá que el robot conozca la habitación en la que se encuentra. Posteriormente, el robot utilizará el algoritmo de Localización Adaptativa de Monte Carlo (AMCL) para finalizar la localización precisa. Al combinar SIFT y AMCL, la eficiencia del algoritmo de localización mejora mucho.

Los algoritmos SIFT y AMCL se ejecutan en Robot Operating System (ROS) y el robot y su proceso de localización se simula en el simulador Gazebo. Además de eso, considerando la alta capacidad de procesamiento de datos de Matlab, los algoritmos y la aplicación final se han desarrollado utilizando Matlab, que proporciona una biblioteca basada en ROS para controlar el robot utilizando Gazebo y procesar datos de ROS.

De acuerdo con los resultados en el capítulo posterior, tanto la localización local como la global se logran con éxito con estos algoritmos.

Abstract

This degree project aims to carry out self-localization algorithm based on image processing for robots. Several algorithms have been reviewed along this project and finally, one combining two of them, one processing images and other one managing distances to obstacles, has been selected and carried out and deployed in a proof-of-concept to check its feasibility and its performance. Briefly, the application that has been developed and tested in a simulated environment first lets the robot recognize in which room it is by taking pictures. Once it determines the room it is in, it downloads the corresponding map and determines its exact position in the room by measuring distances to obstacles.

The features of images are extracted with Scale Invariant Feature Transform (SIFT) and described with highly distinctive multi-dimensional vector making features invariant to changes in illumination, scale and noise. The use of this algorithm will let the robot know the room it is in. Afterwards, the robot will use Adaptive Monte Carlo Localization (AMCL) to finish accurate localization. By combining SIFT and AMCL, the efficiency of localization algorithm improves a lot.

The SIFT and AMCL algorithms are run on Robot Operating System (ROS) and the robot and its localization process is simulated in simulator Gazebo. Apart from that, considering the powering data processing capability of Matlab, the algorithms and the final application have been developed using Matlab, that provides a ROS based library to control the robot using Gazebo and process data from ROS.

According to the results in later chapter, both local and global localization are successful accomplished with these algorithms.

Index of contents

Index of figures

Index of tables

List of acronyms

AI – Artificial Intelligence
AMCL – Adaptive Monte Carlo Localization
API – Application Programming Interface
EKF – Extended Kalman Filter
MHT – Multi-hypothesis Tracking
PCA-SIFT – Principal Component Analysis- Scale Invariant Feature Transform
RGB – Red, green, blue
ROS – Robot Operating System
RPC – Remote Procedure Call
SIFT – Scale Invariant Feature Transform
SLAM – Simultaneous localization and mapping
SURF – Speeded Up Robust Features
TCP/IP – Transmission Control Protocol/Internet Protocol
UDP – User Datagram Protocol

Chapter 1: Introduction

Chapter 1 Introduction

1.1 Project's justification

This project aims to carry out self-localization algorithms for robots based on image processing and ROS.

Robots plays a more and more important role in industry and non-industry. As the concept of Industry 4.0 was put forward in Germany in 2013, it rapidly caused responses from various countries. There is no doubt that the developing of robots will promote Industry 4.0.

When it comes robots, the most basic capability of robots is localization. No matter what robots plan to finish, they need to know its current pose (location and orientation) and the target pose whenever robots is stopping or moving.

1.2 Objectives and proposed solutions

As section 1.1 shows, the objective of this project is to specify and carry out algorithms to process images so that the robot can be aware of its location.

Several algorithms and approaches have been analyzed in the extensive scientific literature about that concern. The proposed solution herein is to combine an image based algorithm like SIFT and a distance based algorithm like AMCL to get the robot's self-localization. The resulting application will be verified using the Gazebo simulator and controlled by ROS and Matlab. Firstly, use SIFT's characteristics, invariant to changes of illumination, scale, and noise, to get the features of current image that the robot takes and match it with the pictures in database while the robot is moving, to check which room the robot is. Once the match is successful, extract this room's map from a map database and finish the accurate localization with AMCL.

The tested robot is TurtleBot which is simulated in 3D simulator, Gazebo. Meanwhile, use ROS to develop and debug algorithms and Matlab to control TurtleBot and process data.

1.3 Structure of the project report

This paper is divided into several different chapters as follows:

- Chapter 1: Introduction
  A brief introduction of the degree project, including the justification, objective, proposed solutions and structure of the paper.
- Chapter 2: State of the art
  The current state of localization field of knowledge, and the details of ROS, Gazebo, TurtleBot and the algorithms we use, AMCL and SIFT
- Chapter 3: Specification and design restrictions
  The description of functions developed and the general framework build

between Matlab and ROS, SIFT and AMCL

- Chapter 4: Description of the solution
  The details of the application that has been developed, including the process of localization, including how to build the maps, create reference database, general localization with SIFT, and accurate localization with AMCL.
- Chapter 5: Results
  The results of image processing and matching with SIFT, and local and global localization with AMCL. In this chapter, the tests that has been run to verify correctness of the system are described.
- Chapter 6: Budget
  The cost of developing the proof-of-concept, i.e. the final application from this degree project, is estimated, considering the funding schema of the European research program H2020.
- Chapter 7: Conclusion
  The conclusion of what this paper did, and some future work which can make the localization algorithms improved, are presented in this chapter.

Chapter 2: State of the art

Chapter 2 State of the art

2.1 Robot

2.1.1 Industry 4.0 and robots

Robots are created to replace human beings to complete monotonous, boring and dangerous work. Nowadays, the application area of robot is wider and wider, extending from 95% industry applications to non-industry applications in many fields. Meanwhile, there are more kinds of robots like miniature robot.

The concepts of Industry 4.0 was proposed in Germany in 2001, which was officially launched at the Hannover Messe in 2013. Its purpose is to increase the level of German manufacturing through the application of new technologies such as Internet of Things, and to establish smart factories with adaptability, resource efficiency and ergonomics[1]. At present, various countries are promoting the developing plan for Industry 4.0.

Robots plays an important role in the era of Industry 4.0. They are no longer independent units, but need to communicate with other smart devices, to sense the environment, and even to integrate complex information to make decisions. Industry 4.0 requires the integration of many technologies of intelligent robots into larger production systems[2]. Therefore, it is particularly important to develop intelligent robots to promote Industry 4.0.

When it comes to developing robots, programming robots becomes a significant challenge to faster adoption of robots in industry. An approach is required to develop application for almost any robot and robot from almost any manufacturer. That is why ROS is so important to Industry 4.0. As a software platform for robots, the main goal of ROS is to provide code reuse support for robot research and development. Such a characteristic will largely satisfy the needs of a wide range of robot developers. In the future, this open and inclusive open source software platform will be highly competitive.

2.1.2 ROS

ROS (Robot Operating System) is an open source operating system for writing robot software. It collects tools, libraries, and conventions and aims to simplify the task of creating complex and robust robot behavior across a wide variety of robotics platforms[3]. It has become the reference middleware, available for most of the robots manufactured nowadays, that simplifies development of robotics applications for almost all robot.

Compared to other software frameworks for robots, there are four main advantages of ROS:

- Loosely coupled mechanism facilitates the organization of robot software framework.
- The most abundant function library for robot makes it faster to build a robot prototype.
- Very convenient data recording, analysis, and simulation tools to facilitate debugging.
- Academic and industrial standards to facilitate learning and communication.

ROS is focused on maximizing code reuse in the robotics research and development. To support this, ROS has the following characteristics:

- Distributed process: It is programmed in the form of minimum units of executable processes(nodes), and each process runs independently and exchanges data systematically.
- Package management: Multiple processes having the same purpose are managed as a package so that it is easy to use and develop, as well as convenient to share, modify, and redistribute.
- Public repository: Each package is made public to the developer's preferred public repository (e.g., GitHub) and specifies their license.
- API: When developing a program that uses ROS, ROS is designed to simply call an API and insert it easily into the code being used.
- Supporting various programming languages: The ROS program provides a client library to support various programming languages. The library can be imported in programming languages that are popular in the robotics filed such as Python, C++, and Lisp as well as languages such as JAVA, C#, Lua, and Ruby. In other words, you can develop a ROS program using a preferred programming language.

These characteristics of ROS have allowed users to establish an environment where it is possible to collaborate on robotics software development on a global level. Reusing a code in robotics research and development is becoming more common, which is the ultimate goal of ROS.

To maximize user reusability, ROS is developed in the form of nodes, which are the smallest unit executable program that is subdivided according to its purpose. The node exchanges data with other node through message and eventually a large program is established[4]. The key concept here is message communication between nodes. Message communication is illustrated in Figure 1.

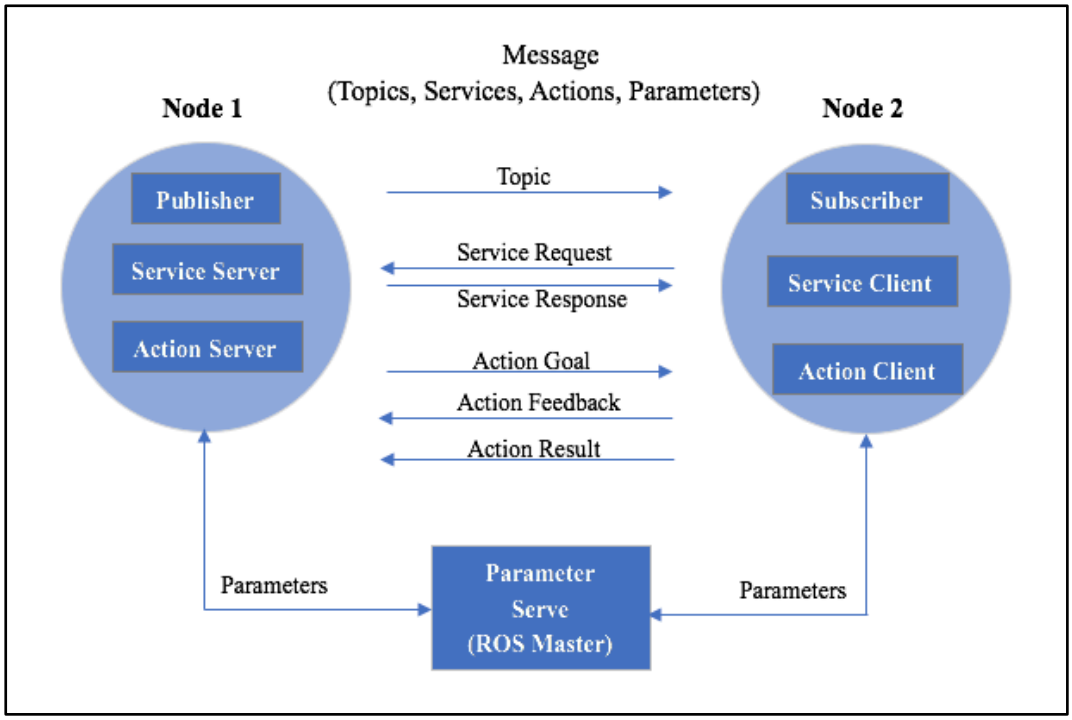

Figure 1: Message Communication between Nodes

Here are the description of main elements in Figure 1:

- Node
  A node is the smallest unit of processor which run in ROS. You can consider it as an executable program. Every single node in ROS has its own function, such as sensor driver, obstacle recognition, motion controller and so on.
- Message
  Messages are sent and received between nodes. The types of message are variable such as integer, floating point and Boolean. Users can also use structures such as simple data structures that messages are included in messages, and message arrays that list message. The underlying transmitted depends on TCP/IP or UDP.
- Topic
  The nodes publish or subscribe message on topics. After that the publisher node register its topic in the master and publishes messages on the topic, the subscriber node who want to receive this topic need to send request to the master. If the connection is built successfully, the subscriber node can directly exchanges messages as a topic with the publisher node.
- Publisher and Subscriber
  Publish refers to sending data in the form of message that corresponds to the content of the topic, and subscriber is to receive interesting data. Both of them needs to register in the master, and then they can exchange data directly. Meanwhile, a single node can become multiple publishers and subscribers.
- Service
  Service is synchronous two-way message communication between a service client and a service server. The service client requests a service corresponding

to a specific task, and the service server is responsible to the service response.

- Service Server
  A service server is a server that receives a response as an input, and transmits a request as an output. Both request and response are messages. After receiving the service request, the server executes the specified service and delivers the result to the service client.
- Service Client
  A service client is a client that receives a request as an input, and transmits a response as an output. Both request and response are messages.
- Action
  Action is a message communication method that is used in the case of a two-way request like service. The difference between them is that a long request is required after processing the request in action, and midway feedback values are required in action.
- Action Server
  Action server is a server that receives target from action client as the input and transmits the results and feedback as the output. After receiving the target value from the client, it is responsible to perform the actual action.
- Action Client
  Action client is a client that receives target from action client as the input and transmits the results and feedback as the output. It delivers the target to the action server, receives the results and feedbacks and outputs the next indication or cancellation target.
- Parameters
  Parameters are used in the nodes that are set by default and can be read and written externally as needed. In particular, it is very useful owing to changing the setting value at the real time by using an external writing function.
- Parameter Server
  Parameter server is used to register every parameter when it used in a function package. It is also a function of the master node.
- Master
  The master is responsible for connections and message communication for node-to-node, similar to a name server. Roscore is its running command. When you run the master node, you can register the name of each node and get information as needed. Without the master node, access cannot be established between nodes.

Here is an example about how ROS runs.

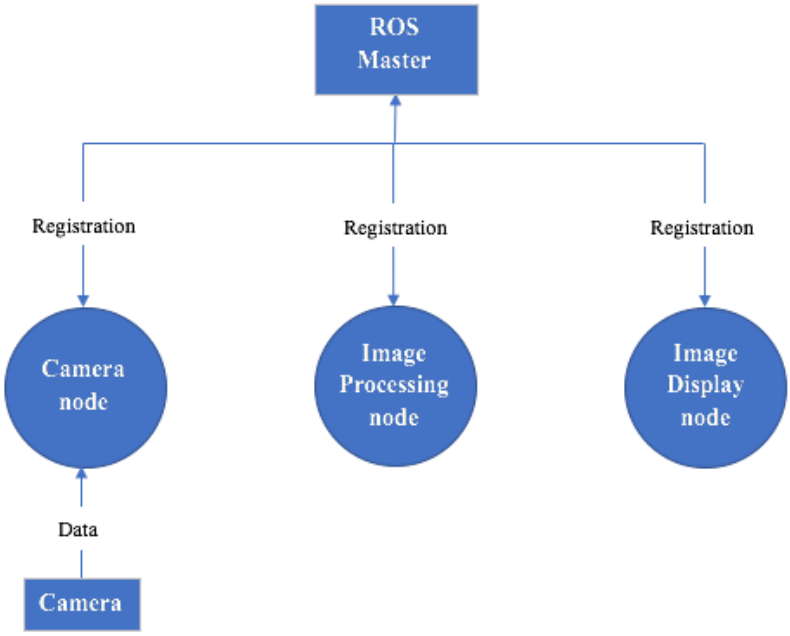

Figure 2: Example of ROS

In this example, there is a camera installed on the robot. The expected function is to make the real time image visible in computer screen. At first, we design three nodes, *Camera* node used to communicate with camera, *Image Processing* node used to process the images data obtained from camera node, and *Image Display* node used to display the image in computer screen.

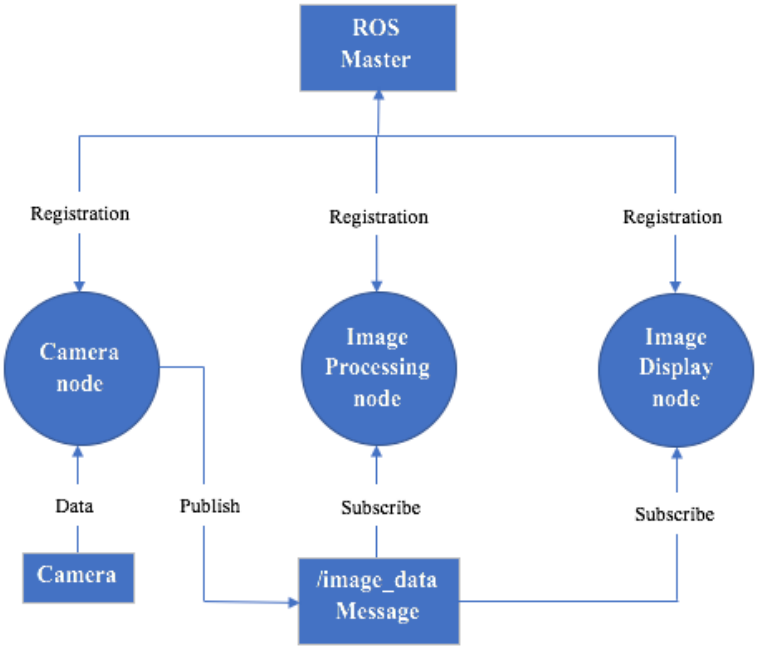

Figure 3: Communication based on topics

As Figure 3 shows, after registering in ROS master, *Camera* node declares that it will publish a topic named */image_data*. The other two nodes (*Image Processing* node and

Image Display node) declare that they subscribe this topic. Therefore, once the *Camera* node receives the data sent by camera, it immediately sends the *image_data* directly to the other nodes.
If *Image Processing* node wants to actively get the data received by *Camera* node, the following model based on ROS service will be used.

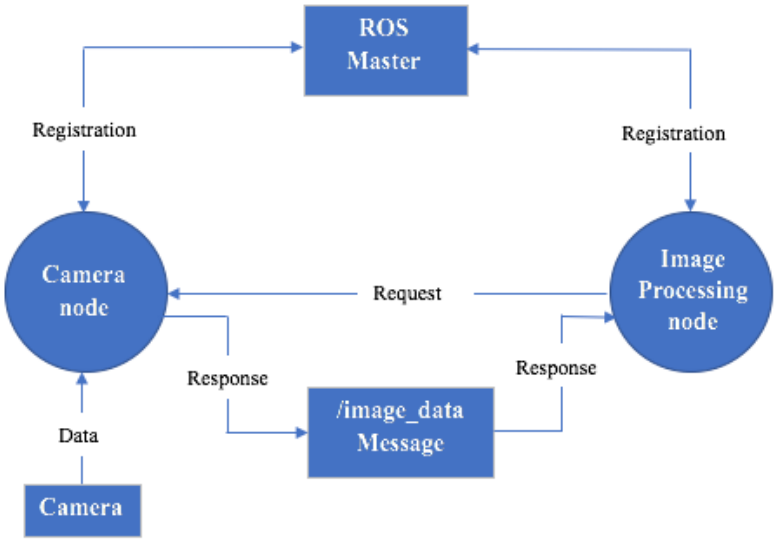

Figure 4: Communication based on service

Nodes can register a specific service in ROS master. In this example, *Camera* node will receive data from camera and then send them to *Image Processing* node after the first request from *Image Processing* node to *Camera* node.
As a beginner, you can start ROS from the Beginner Level of [5], which is the best tutorial for beginners.

2.1.3 Gazebo

To create embedded application for robots, robot simulators are developed which doesn't depend on real robots. More importantly, the most applications that simulators use can be transferred to real robots without big changes. If you don't have a robot, you can simulate a robot in simulator. If you have a real robot, you can test the application in simulators first to avoid the unnecessary loss of time and money[6].
Well-designed simulators can quickly test algorithms, design robots, perform learning tests, and train AI system with realistic scenarios. In this project, we use Gazebo simulation platform to simulate the robot.
Gazebo is an open source software platform and everyone can use it to develop plugins using model components. It has abundant models in the model database, and you can also create your own model using the Building editor. Gazebo supports ROS and is able simulate various robots in complex indoor and outdoor environment accurately and efficiently. Especially, it is used to develop robots which aim to interaction, lifting or grabbing objects, or any other activity that needs recognition and localization in 3D.
You can start the connection between ROS and Gazebo from [7].

2.1.4 TurtleBot

TurtleBot is a low-cost, personal robot kit with open-source software, which was created at Willow Garage by Melonee Wise and Tully Foote in November 2010. You can build TurtleBot and drive it around one house and see the 3D scene seen by the TurtleBot, and develop applications that hold different functions as you like[8].

When powered by ROS software, TurtleBot can handle vision, localization, communication and mobility in an easy way. It can autonomously move anything on top of it to wherever that item needs to go, avoiding obstacles along the way. TurtleBot's open source hardware, driven by the open source ROS software, is an ideal starting point for any developer transitioning into robotics.

Meanwhile, there are powerful ROS packages of Turtlebot simulated by Gazebo, which is quite useful when developing algorithms of robots. You can download it from official website of ROS[9].

2.2 Localization for robot

2.2.1 The status of localization (algorithms)

As the most basic ability for intelligent robots, autonomous navigation is constantly attracting attention in the industry. When it comes to autonomous navigation of mobile robots, it can be simply attributed to three problems: *where I am, where I am going* and *how could I go there*. Localization is the most basic step to solve *where I am* and this is what this project aims at.

Localization is the process for robots to ensure its location and orientation in the working environment. To be more precise, use some input information, including prior environment map information, real time pose estimation of robots and observation values from sensors, to generate a more accurate estimation of robots' current pose[10].

The localization method of robots depends on what sensors are used. Localization sensors for mobile robots include odometry, camera, lidar, ultrasound, infrared, microwave radar, gyroscope, compass, accelerometer and etc. The corresponding localization technology for robots can be divided into two categories: absolute localization and relative localization. Absolute localization uses navigation marks, active/passive identification, map matching, GPS and so on technology to achieve self-localization, whose result is relatively accurate. Relative localization is called dead reckoning method, which infers current pose of robots by calculating robots' direction and distance relative to initial pose.

- Localization technology based on map matching

  Localization problem of robots based on map matching mainly focused on analyzing possible robot's location on the map. The key point is that the robot can perceive the location information of local environment and match location environment information in the known map. In addition, robots' localization based on map matching needs to combine with other localization methods to achieve goals.

- Localization technology based on landmarks

  Landmarks have obvious characteristics which can be detected by sensors installed on robots. Landmarks designed manually have fixed location in the three-

dimensional space where the robot is. Therefore, the core mission of localization for robots is to recognize landmarks reliably and quickly calculate actual location of the robot in the known map. The actual localization accuracy mainly depends on accurate identification of landmarks and the accuracy and speed of extracting environmental location information.

- Dead reckoning localization technology based on probability estimation

There are many uncertainties during localizing. For example, the uncertainty of robot itself, the accumulation of odometer errors, the noise of sensors, and the complexity and unknowns of the robot's environment. In short, due to the existence of these uncertain factors, localization becomes more complicated. In recent years, many researchers have applied probability theory to robots' localization. The core idea is to use the data collected so far as known conditions, and then recursively estimate the posterior probability density of the state space. The method based on the probability estimate of particle filtering is more promising to implement. Particle filtering, also known as the sequence Monte Carlo, is a new filtering algorithm developed in the middle and late 1990s. Its core idea is to represent probability distributions with random samples. Dallert combined the particle filter algorithm with the robot motion and perceived probability model to propose the idea of robot Monte Carlo localization. The core idea is to use a set of filters to estimate the possible location of the robot, that is the probability of being at that location. Each filter corresponds to a position, and each filter is weighted using observations, which in turn increases the probability of the most likely location.

The pose of mobile robot is usually represented by triples (t_x, t_y, θ), where (t_x, t_y) represent the location of the robot under the world coordinate frame (translational component), and θ represents the motion direction of robot (rotational component). Pose estimate mainly involves three analysis methods under different assumption:

1- Given initial pose and the map

The localization under this condition is called Local Localization or Pose Tracking, using a relative localization method. The process of pose estimate is showed as Figure 5. Most study is under this situation which most pose estimate algorithms are based on.

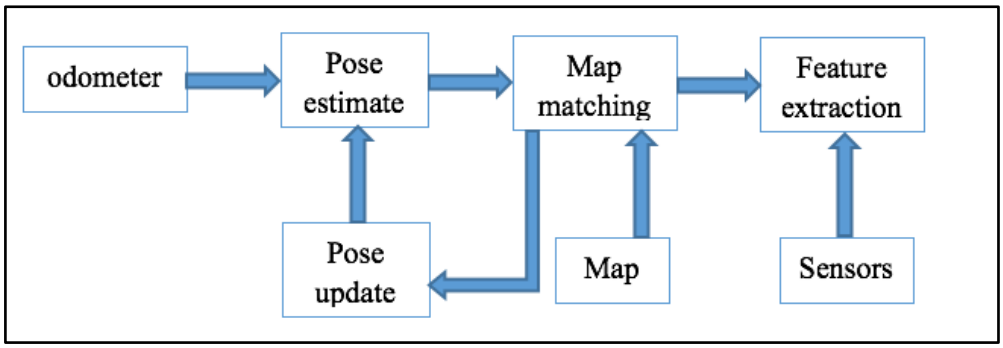

Figure 5: Process of local localization

2- Given map but not initial pose

This kind of localization is called Global Localization. The method is necessary when the robot needs to reset or loses direction, or suffers kidnapped. In order to find out the current pose of robot, the commonly assistive method was to add some artificial beacons in the environment to identify location, such as wireless transceiver, geometric beacon, satellite navigation, barcode technology and infrared or ultrasound receiving system. Another method is to identify natural landmarks with advanced sensors like video camera and three-dimensional vision system. But totally autonomous robots should self-identify the environment and self-localize with sensors belonging to itself. Therefore, developing global localization algorithms is one important mission for autonomous mobile robots.

3- Unknown map and initial pose

The ultimate target for localization of autonomous mobile robots is to complete this kind of self-localization. Robots need to create the map of environment constantly and calculate current pose with the map. That is to say, robots must not only complete the pose estimate but also create the map at the same time. This is called SLAM (Simultaneous Localization and Map Building). This is an important study direction of autonomous mobile robots currently.

The real pose of mobile robots has uncertainty owing to robots manufacturing errors, odometer and other sensors effected by measurement noise and errors in motion control. Therefore, probability theory is an important tool for localization. In this project, we aim to achieve local and global localization with AMCL, one algorithms based on probability theory. Apart from that, considering the efficiency of robots' self-localization, we use SIFT to recognize approximate location of robot before running AMCL. The details of AMCL and SIFT can be found in following sections and the details of combination between them can be found in following sections.

2.2.2 AMCL

To localize the robot, AMCL algorithm uses particle filtering to estimate its pose. Let us have a look of Particle Filters first.

Particle filters

The particle filter based on Monte Carlo methods use a set of particles to represent probability distribution. It can be used on any form of state space model. The core idea of the particle filter is to express probability distribution by extracting random state particles from the posterior $bel(x_t)$ [1], so it is sequential importance sampling. In particle filters, particles are the samples of the posterior distribution and are denoted

$$\chi_t = x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]}$$

[1] $bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t})$

At time t, each particle $x_t^{[m]}$ is a hypothesis which represents what the true world state may be. M is the number of particles in the particle set χ_t.

As one kind of Bayes filter algorithms, the particle filter algorithm constructs the belief $bel(x_t)$ recursively from $el(x_{t-1})$. This means particle filter constructs the particle set χ_t from the set χ_{t-1}. The key steps of particle filter algorithm are showed in Table 1.

1:	Algorithm Particle filter $(\chi_{t-1,}\ u_t, z_t)$
2:	$\bar{\chi}_t = \chi_t = \phi$
3:	for m = 1 to M do
4:	sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5 :	$w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6:	$\bar{\chi}_t = \bar{\chi}_t + < x_t^{[m]}, w_t^{[m]} >$
7:	end for
8 :	for m = 1 to M do
9:	draw i with probability $\propto w_t^{[i]}$
10:	add $x_t^{[i]}$ to χ_t
11:	end for
12:	return χ_t

Table 1: Key steps of Particle filter algorithm

The details are descripted as following.

1. At line 4, a hypothetical state $x_t^{[m]}$ is generated based on the particle $x_{t-1}^{[m]}$ and the control u_t. The sample is generated from the m-th particle in χ_{t-1}.
2. At line 5, the *importance factor* $w_t^{[m]}$ is calculated for each particle $x_t^{[m]}$. It is used to incorporate the measurement z_t into the particle set. $w_t^m = p(z_t \mid x_t^m)$ means the measurement probability z_t under the particle $x_t^{[m]}$.
3. Line 8 to line 11 implement resampling, or to say, importance resampling. When drawing the particles, the probability of each particle is with respect of

its importance weight. After resampling, the set of M particles is transferred into another particle set of the same size. With the importance, the distribution of the particles change: they were distributed with $\overline{bel}(x_t)$ before the resampling, and distributed according the the posterior

$bel(x_t) = \eta\, p(z_t \mid x_t^{[m]})\, bel(x_t)$ after the resampling.

MCL

Now we turn to the popular localization algorithm called *Monte Carlo Localization*, which represents the belief $bel(x_t)$ by particles. MCL (Monte Carlo Localization) is used to estimate robots' position and orientation using known environment map and data from ranger sensor and odometry sensor. It is available to both local and global localization. It is easy to implement and runs well in many global localization problems. Although it exists just for a short time, it has become one of the most popular localization algorithms in robotics.

The particles represent the distribution of the possible pose for the robot. Every particle represents a likely state of the robot. The particles will converge into one location when the robot is moving and senses different parts of the environment using range sensors. The basic MACL algorithm is showed in Table 2.

1:	Algorithm MCL $(\chi_{t-1,}\ u_t, z_t, m)$
2:	$\chi_t = \chi_t = \phi$
3:	for m = 1 to M do
4:	$x_t^{[m]} = sample\_motion\_model(u_t, x_{t-1}^{[m]})$
5:	$w_t^{[m]} = measurement\_model(z_t, x_t^{[m]}, m)$
6:	$\bar{\chi}_t = \bar{\chi}_t + < x_t^{[m]}, w_t^{[m]} >$
7:	end for
8:	for m = 1 to M do
9:	draw i with probability $\propto w_t^{[i]}$
10:	add $x_t^{[i]}$ to χ_t
11:	end for
12:	return χ_t

Table 2: Key steps of MCL

Samples from the motion model at line 4. The starting points are obtained from present belief. Then use measurement model to ensure the importance weight of each particle. To obtain initial belief, randomly generate M particles from $p(x_0)$, and assign the uniform importance factor M^{-1} to each particle.

AMCL

Adaptive monte Carlo Localization (AMCL) is the variant of MCL, which dynamically adjust the number of particles based on KL-distance to so that the particle distribution converges to the true distribution of robot state with high probability based on all past sensor and motion measurements. You can find more about KL-distance and more details about AMCL in [11].

The following table illustrates the comparison between MCL/AMCL and other localization algorithms based on particle filter.

Algorithms / Characteristics	EKF	MHT	Coarse(topological) grid	Fine (metric) grid	MCL/AMCL
Measurements	landmarks	landmarks	landmarks	raw measurements	raw measurements
Measurement noise	Gaussian	Gaussian	any	any	any
Posterior	Gaussian	Mixture of Gaussians	histogram	histogram	particles
Efficiency (memory)	++	++	+	–	+
Efficiency (time)	++	+	+	–	+
Ease of implementation	+	–	+	–	++
Resolution	++	++	–	+	+
Robustness	–	+	+	++	++
Global localization	no	no	yes	yes	yes

Table 3: Comparison among particle filter localization algorithms

As we can see in Table 3, MCL/AMCL do not depend on natural and artificial landmarks but raw measurements of the environment. Although its efficiency and resolution are not as good as EKF's and MHT's, MCL/AMCL is easier to implement and has better robustness, which are important characteristics of localization. More importantly, it can be used in global localization.

2.2.3 SIFT

In order to narrow down the distribution range of particles in AMCL, we use SIFT algorithm to ensure a certain area where the robot is, so that the efficiency of localization algorithm will be improved.

SIFT (Scale Invariant Feature Transform) aims to detect and describe local features of images in computer vision. This algorithm gets features of pictures and matches feature points by finding feature points (interest points or corner points) and calculating descriptors of these points about scale and orientation.

The key problem of image matching is to match the pictures of the same target, taken at different time, resolutions, illuminations and poses. SIFT aims to describe an image to a vector set of local features. The vector shows invariance about translation, scaling and rotation. It also has certain invariance to illumination change, radiation and affine transformation.

Features of SIFT

- SIFT features are local features of images, invariant to rotation, scaling and brightness, and stable in affine transformations, viewing angle changes, and noise.
- Suitable for fast and accurate matching in database of massive characteristics owing to its great distinctiveness and abundant information.
- Multiplicity, which means even a few objects can generate lots of SIFT feature vectors.
- Optimized SIFT algorithm can satisfy certain speed requirements.
- Extensibility, which means it can easily combine with other feature vectors.

Implementation steps

1. Build scale spaces

   This step is an initialization operation. Scale space theory aims to simulate the features of image data under multi-scale..

   The Gaussian kernel is one and only linear kernel that implements transformation between different scale spaces. For a two-dimensional image, the scale space of it is defined as

   $$\mathrm{L}(\mathrm{x},\mathrm{y},\partial) = G(\mathrm{x},\mathrm{y},\partial) * I(\mathrm{x},\mathrm{y}),$$

   where $\mathrm{G}(\mathrm{x},\mathrm{y},\partial) = \frac{1}{2\pi\,\partial^2} e^{-(x^2+y^2)/2\,\partial^2}$ is Gaussian function with variable scale. To be more effectively in detecting stable feature points, DoG (Difference of Gaussian) scale space is applied. DoG is generated by convoluting differential Gaussian kernels in different scale with image.

   $$\begin{aligned}\mathrm{D}(\mathrm{x},\mathrm{y},\partial) &= \big(G(x,y,k\,\partial) - G(x,y,\partial)\big) * I(x,y)\\ &= L(x,y,k\partial) - L(x,y,\partial)\end{aligned}$$

   For an image I, the image established under different scale also divided to several octaves to ensure corresponding feature points at any scale. The first octave is the original image, and any other octave is generated by down-sampling from its previous image.

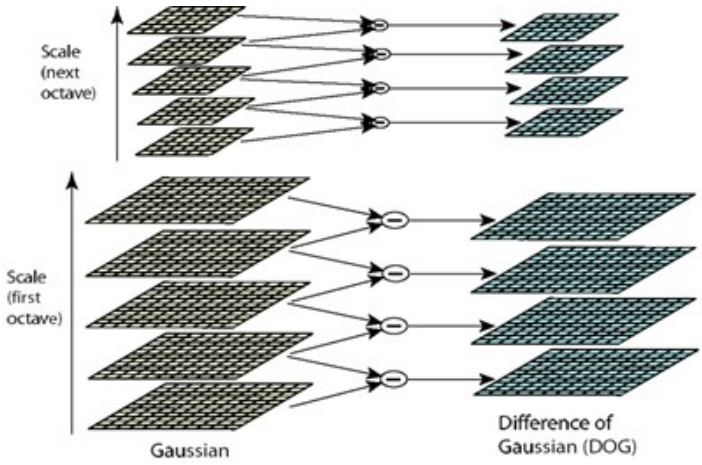

Figure 6: Image pyramid

2. Detect extreme points in DoG scale space
   To find the extreme points in the scale space, every sampled point should be compared with all its neighbors to check that under its image domain and scale domain, if it is larger or smaller than the neighboring points. As you can see in Figure 7, the middle point is compared with 8 neighboring points in the same octave and 26 points in neighboring octaves. After the comparation, the point is considered as a feature point of the image in this scale if its value is maximum or minimum.

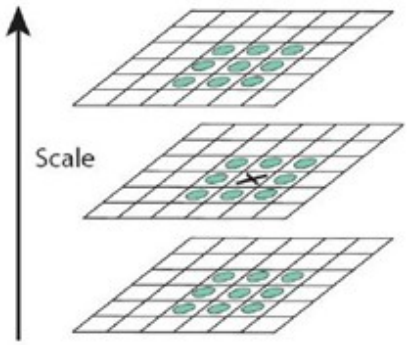

Figure 7: Extreme points detecting

3. Remove bad feature points
   This step essentially aims to eliminates pixels with very asymmetrically local curvature. In order to determine the feature points' position and scale, use three-dimensional quadratic function to fit the DoG function. Simultaneously, to enhance matching stability as well as noise immunity, remove low-contrast key points and unstable points owing to edge response.
   The Taylor expansion of spatial space function is as follows

$$\mathrm{D}(\mathrm{x}) = D + \frac{\partial D^T}{\partial\, x} x + \frac{1}{2}\, x^T \frac{\partial^2 D}{\partial x^2}\, x$$

   Get the derivative of this formula and obtain its exact position. Then we get

$$\bar{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x}.$$

- In the detected feature points, there are two kinds of bad points need to remove as description before. To remove low contrast points, take the value of D(x) at the extreme point of DoG space, and only take the first two items:

$$D(\bar{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\bar{x}$$

  For one feature point, if its $|D(\bar{x})| \geq 0.03$, it will be saved.

- Remove points owing to edge response
  There is strong edge response of Gaussian difference function. Some points should be removed, which have a large principal curvature in a certain direction but a small main curvature in vertical direction. The details of this step can be found in [12].

4. Specify direction for feature points
   Use distribution characteristics of gradient direction of key points' neighboring pixels to assign the directional parameters to every key point, and then the operator of key points will be invariant to rotation.
5. Generate descriptors for key points
   In the previous step, the key point with the main direction has been obtained. The next step is to sample the neighborhood of the key point to form a descriptor of the local image, and then use a certain measurement method to match the descriptors.
6. Match with SIFT
   After generating the descriptors of two images, match descriptors of images in all scales. If two feature points are matched successfully in 128-dimensional, they act as matched points.

SIFT has unparalleled advantages in extracting the invariant features of images, but there are still some drawbacks of it such as not being able to extract features in the case that images have high edge blur. The comparison between SIFT and some other algorithms based on SIFT is showed in Table 4.

Algorithms / Characteristics	Time	Scale	Rotation	Blur	Illumination	Affine
SIFT	common	best	best	common	common	good
PCA-SIFT	good	good	good	best	good	best
SURF	best	common	common	good	best	good

Table 4: Comparison among SIFT, PCA-SIFT and SURF

As Table 4 shows, SIFT does not act well in efficiency, blur and illumination, but holds the best performance in changes of scale and rotation.
More details of SIFT can be found in [13] and [14].

Chapter 3: Specification and design restriction

3.1 Synoptic design specification

The aim of this final year project is to carry out ROS algorithms to achieve robot's self-localization. To be faster, image-based localization method will be used. Instead using a real robot, we are going to use a simulated robot named TurtleBot and simulate it in Gazebo.
First, we use SIFT to process real time images that the TurtleBot simulated with Gazebo obtains to resolve which room it is. The Gazebo environment where TurtleBot moves is built as three similar rooms with different objects. Once the TurtleBot decides which room it is, extract this room map that previously built and stored in a maps database. Then use that map to finish accurate localization with AMCL.
Meanwhile, Robotics System Toolbox in Matlab enables Matlab to communicate with ROS, including connecting to a ROS network, collecting data, sending and receiving messages, and deploying code to a standalone system[15]. Considering the powerful capability of processing data of Matlab, we choose this collaboration framework between ROS and Matlab.

3.2 Connection between SIFT and AMCL

As before mentioned in the introduction of SIFT in 2.2.3, it aims at obtaining features and matching key points of pictures. Its feature is not only scale invariance, but good adaptabilities in changes of rotation angle, image brightness and shooting angle. Therefore, we choose SIFT to detect the features of environment to help TurtleBot decides which room it is, so that the range of accurate localization with AMCL will decrease. This project considers a small use case with three rooms to be detected. Imagine that there are much more rooms in localization environment, the first step for TurtleBot, deciding which room it is with SIFT, will significantly reduce the running time of AMCL. Accordingly, the overall efficiency of localization algorithm will improve a lot.
You can find the details of the connection between them in section 4.1

3.3 Connection between ROS and Matlab.

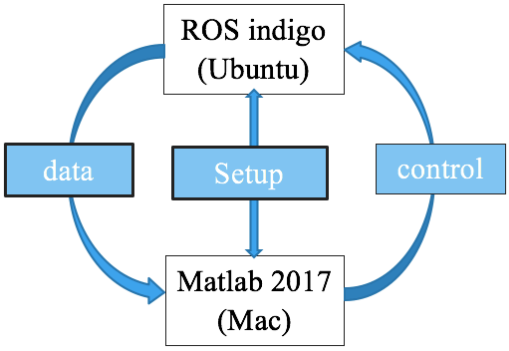

Figure 8:Connection between ROS and Matlab

The Figure 8 shows the deployment that has been used. As we can see in Figure 8, after the connection between ROS and Matlab is set up successfully, Matlab running in a Mac is used to control ROS, running in Ubuntu, by sending commands[16]. TurtleBot will act under the control commands. After that, ROS will send sensor data to Matlab, which are going to be processed in Matlab so that we will get useful information, such as laser data, odometry data, image data, etc.

The details about the commands and configuration is deeply explained in following sections.

Chapter 4: Description of the solution

4.1 General schematic for this localization method

There is some basic work needed to be done before running the core algorithms. The first thing is to create a 3D model for simulation in Gazebo, which contains three different rooms and the robot we use, TurtleBot. After setting the environment and connecting Matlab to TurtleBot in Gazebo, the three maps relative to three different rooms are build, which make the motion of TurtleBot and location results visible in Matlab. Another thing is to take pictures of each room representing the rooms' features.

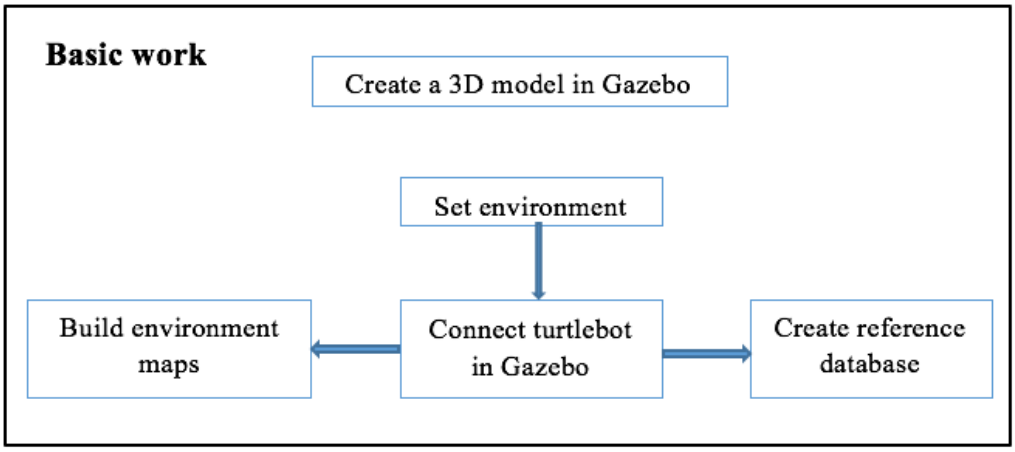

Figure 9: Basic work in this project

To do self-localization, TurtleBot takes a picture at current location. Then this picture is matched with pictures in database one by one using SIFT. If the match fails, TurtleBot will continue to move and take pictures until the match succeeds. After that one room is chosen, its map is extracted from a map database and used to AMCL. AMCL will be used to finish the accurate localization while TurtleBot keeps moving.

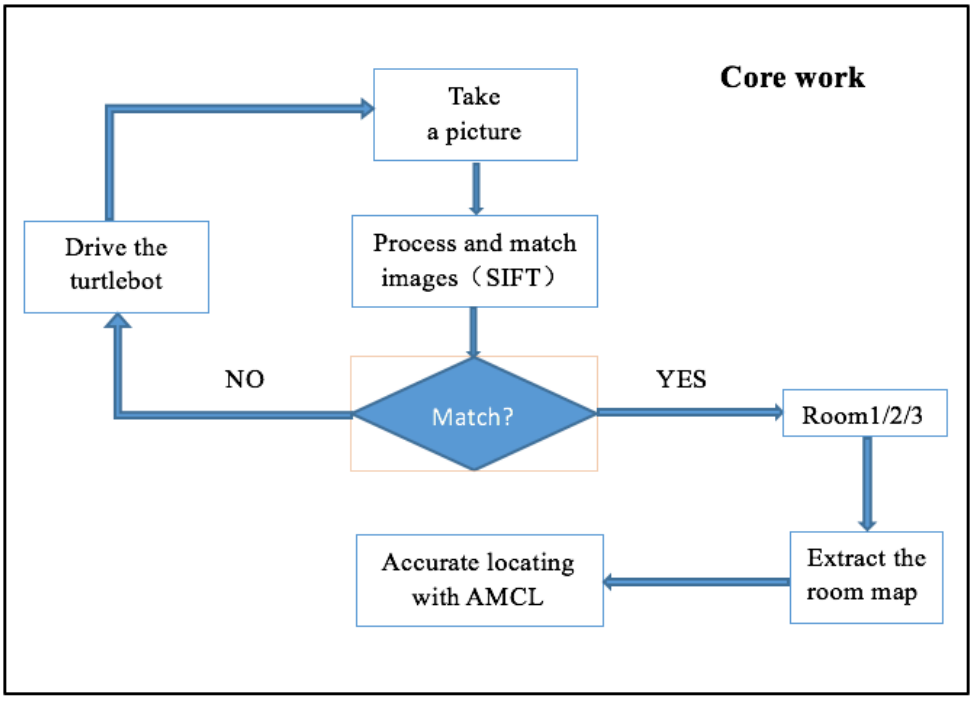

Figure 10: Core work in this project

4.2 Create 3D model for simulation in Gazebo

In this project, the expected Gazebo environment contains several rooms containing different objects[17], which means the rooms hold different characteristics so that TurtleBot can decide which room it is by matching real time pictures with pictures in database.
In this project, Building Editor in Gazebo[18] has been used to create a building containing three room as Figure 11 shows.

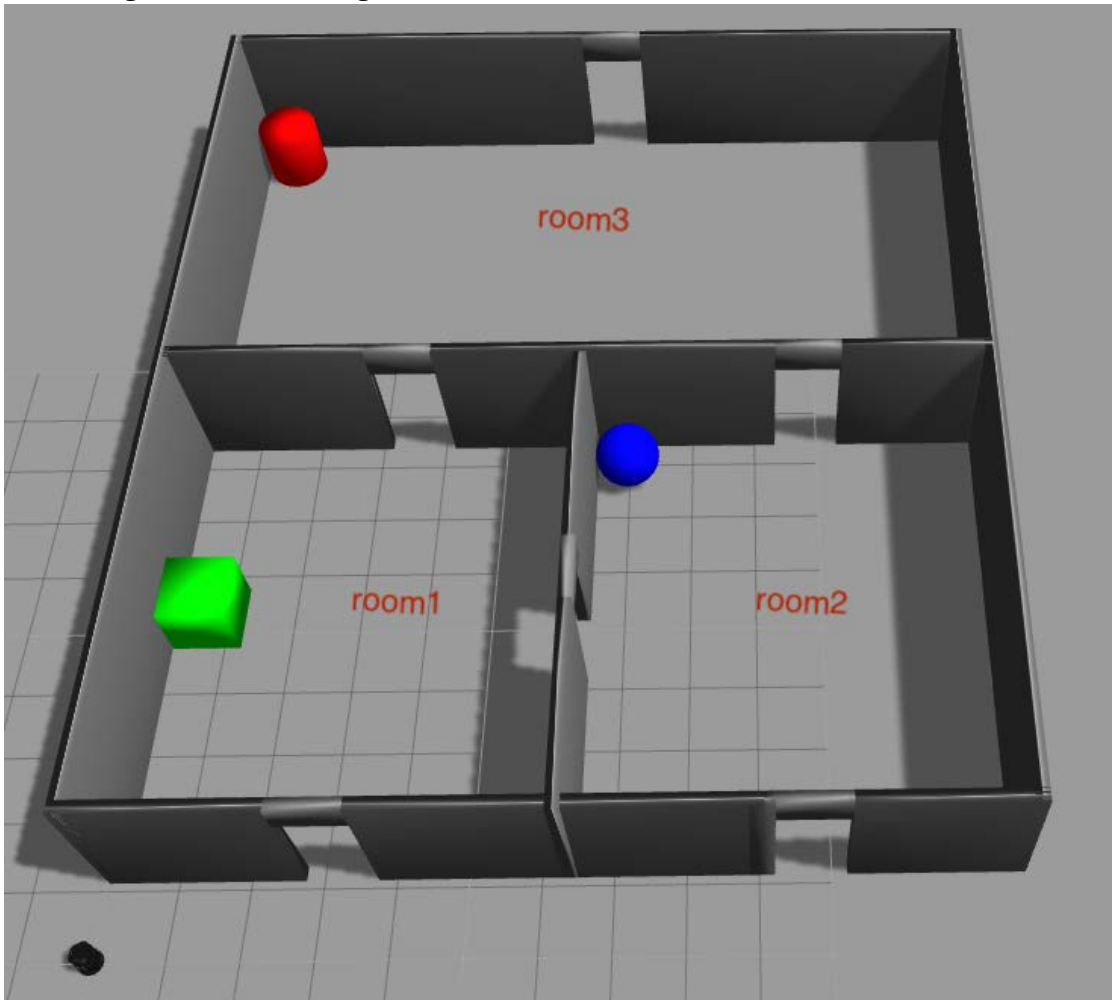

Figure 11: 3D model in Gazebo

4.3 Set environment

To connect Matlab to ROS, the two computers must be connected to the same network. The desktop computer where ROS and Gazebo are installed runs Ubuntu and uses the wired network, and the laptop where Matlab is installed connects to the same network wirelessly.
ROS_IP and ROS_HOSTNAME are optional environment variable that sets the declared network address of a ROS node or tool. The options are mutually exclusive, if both are set ROS_HOSTNAME will take precedence. Use ROS_IP if you are specifying an IP address, and ROS_HOSTNAME if you are specifying a host name.

The environment variable ROS_MASTER_URI is required when *roscore*[2] is not running on localhost. ROS_MASTER_URI is used by ROS nodes to locate the master. In this project, we check the IP address of the two computers in terminals, and set the ROS_IP and ROS_MASTER_URI in Matlab and Ubuntn terminal.

4.4 Connect TurtleBot in Gazebo

As each ROS version corresponds to one or two Ubuntu versions, we installed the ROS indigo in Ubuntu 14.04, which is the most convenient ROS version for beginner and has the most users. You can find the installation details and resources in [19]. Meanwhile, indigo uses Gazebo 2 which is the default version of Gazebo on Ubuntu 14.04 and is recommended. Apart from that, gazebo_ros_pkgs, the set of ROS packages for interfacing with Gazebo, also needs to install from [20].
Start up ROS master and start Gazebo with the model built before in Ubuntu terminal. Then connect TurtleBot with Matlab.

4.5 Build environment maps

In order to make the process of TurtleBot motion and the results of localization visible in Matlab, the environment maps are created under Matlab control.
The general idea is creating a desired path for each room that TurtleBot follows to move. A desired path is a set of waypoints defined explicitly. During the motion of TurtleBot, use the range sensors reading and known poses of TurtleBot to create the map[21].
Before the map building, ROS publishers and subscribers are to be created to communicate with Gazebo and get transformation tree from ROS, which allows to find the pose of TurtleBot at the time of lase sensor reading is observed. The tf system in ROS keeps track of multiple coordinate frames and maintains the relationship between them in a tree structure. In the tf tree, base_link coordinate frame is rigidly attached to the mobile robot base, and map is a world fixed frame with its Z-axis pointing upwards. In this project, get the transformation between the map and base_link frames to obtain the position and orientation at the time of sensoring reading.

[2] A collection of nodes and programs that are pre-requisites of a ROS_based system. It must be running in order for ROS nodes to communication.

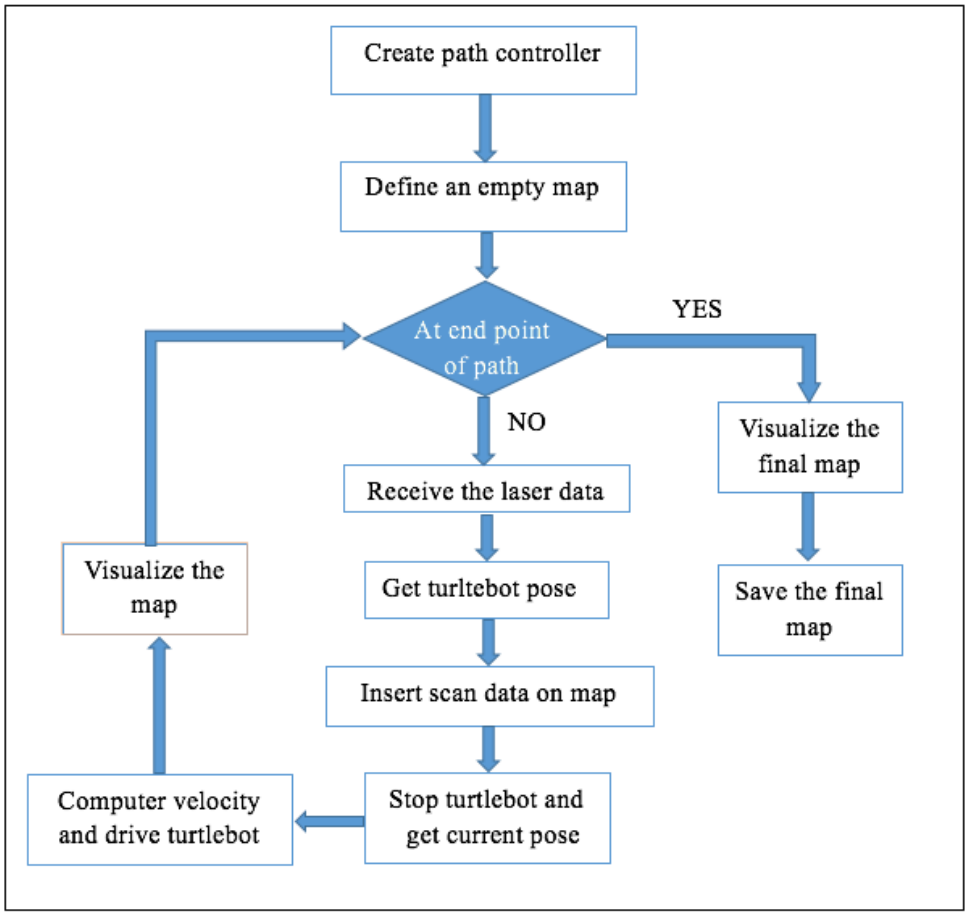

Figure 12: Process of building maps

The details of map building is showed in Figure 12. According the position of each room, the path that TurtleBot will follow is created. Based on the path defined, create the path following using *robotics.PurePursuit*[3] object to drive the TurtleBot along the path. Meanwhile, define a goal point and a goal radius to stop the robot at the end of the path. Then define an empty map with a resolution of 20 cells per meter using *robotics.OccupancyGrid*[4] to capture sensor readings. The map size depends on its size in Gazebo.

After these steps, a while loop will build the map of the room while driving the robot. Firstly, judge if the distance between robot current position and final position is larger than the goal radius. If that, receive a new laser sensor reading and get robot pose at the time sensor reading. The robot position and orientation are obtained from the transformation between the map and base_link frames. Then extract the laser scan data and insert them in the map. The TurtleBot should be stopped before using the driver controller to calculate the linear and angular velocity of the robot and publish it to drive TurtleBot. After that, visualize the map and update the distance to goal point, and return the loop until the distance between robot current position and final position is less than goal radius.

[3] A object class in Robotics Systems Toolbox in Matlab

[4] A object class in Robotics Systems Toolbox in Matlab

Build the maps of three room in turn with different path relative to them. The maps are show as Figure 13, Figure 14, Figure 15.

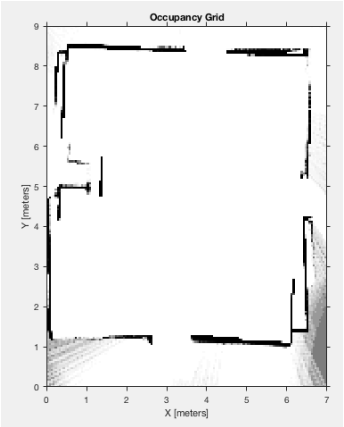

Figure 13: The map of room1

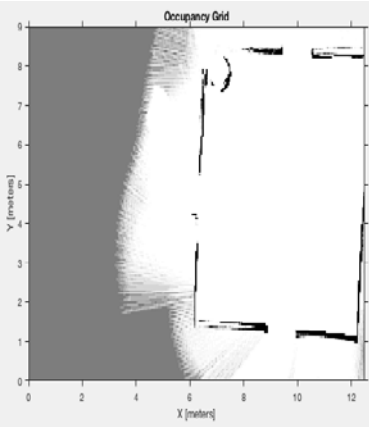

Figure 14: The map of room2

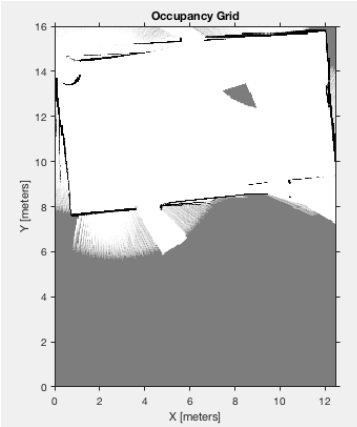

Figure 15: The map of room3

4.6 Create reference database

According to the introduction of SIFT in 2.2.3, the database of rooms' pictures should be created before matching process. After subscribing ***/camera/rgb/image_raw*** image topic, wait for the image data and then display the image with *imshow*[5].
After taking a picture of box in room1, drive the TurtleBot with teleoperation in Matlab, and take pictures of sphere and cylinder in room2 and room3. The three pictures each of which represents a room are showed as Figure 16, Figure 17, Figure 18.

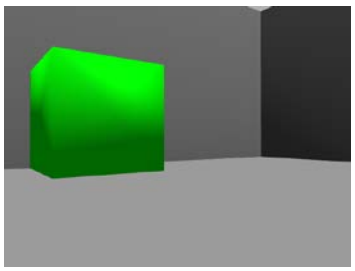

Figure 16: Green box in room1

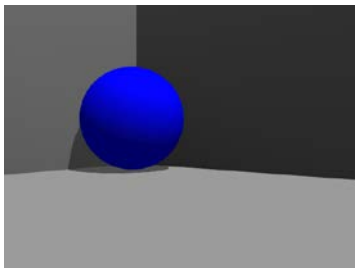

Figure 17: Blue sphere in room2

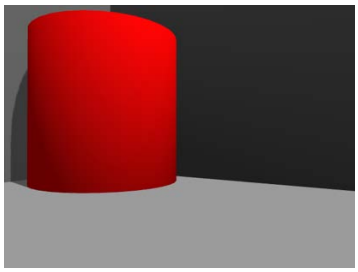

Figure 18: Red cylinder in room3

4.7 General location using SIFT

The main algorithms of robot's self-localization can be run after finishing all basic work. The first step for TurtleBot is to take a picture at current position and process the picture with SIFT algorithm to check if it is matched with any picture in database.
The process of SIFT is showed as Figure 19.

[5] A function in Robotics Systems Toolbox in Matlab

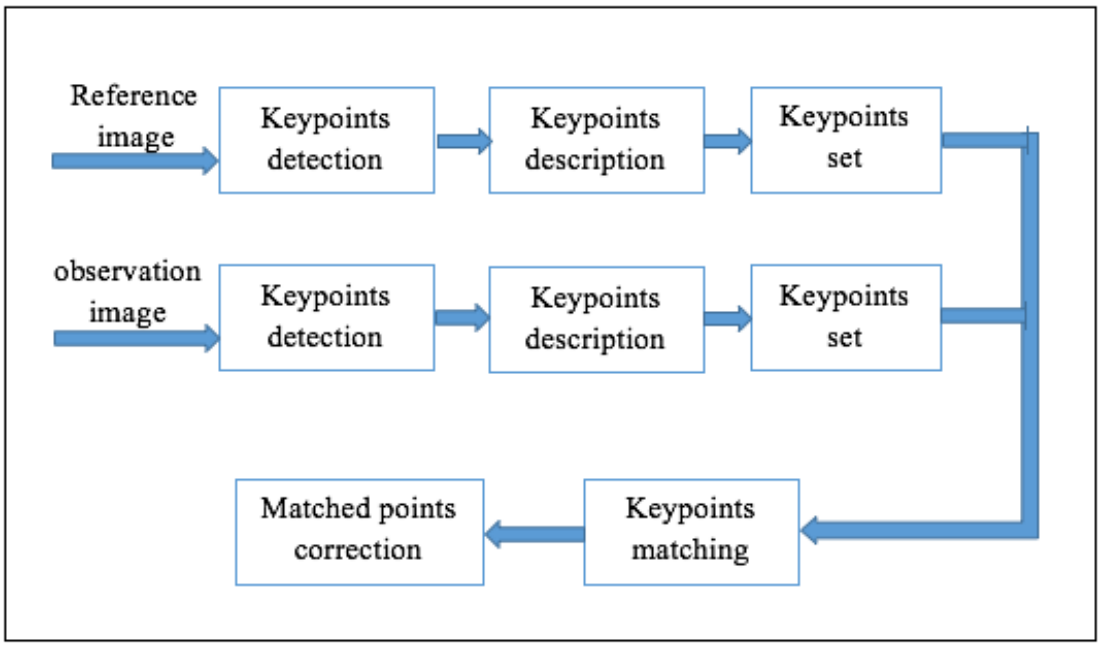

Figure 19: Process of SIFT

Key points detection

Key points are local extremal points with direction information detected under different scale spaces. Gaussian kernel is the only one that can produce multi-scale space. The scale space of an image L (x, y, ∂), is defined as the convolution operation between the original image I (x, y) and a variable two-dimensional Gaussian function G (x, y, ∂).

$$L(x, y, \partial) = G(x, y, \partial) * I(x, y)$$

$$G(x, y, \partial) = \frac{1}{2\pi \partial^2} e^{-(x^2+y^2)/2\partial^2}$$

For giving RGB images, use different values of ∂ to do Gaussian smoothing after converting RGB images to gray images. Then the images are down-sampled to get groups of different size (octave) and each of groups have some images(intervals). After getting the DoG Pyramid, localize the accurate key points and assign orientation for key points.

Key points description

In the above step, the key points holding main orientation are created. Then 16x16 neighborhood around the key point is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bit orientation histogram is created. So a total of 128 bit values is available. It is represented as a vector to form key point descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

Key points matching

The same previous steps are taken to both real time picture and pictures in database. Then the key point sets of them are extracted to match. Key points between two images are matched by identifying their nearest neighbors. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other

reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminates around 90% of false matches while discards only 5% correct matches.

Matched points correction

RANSAC (Random Sample Consensus) is a parameter estimation method to robustness. The initial values of the parameters in the designed target function are estimated by repeatedly extracting the minimum point set, and all data are divided into inlier and outlier by using these initial values. Finally, use all inlier points to recalculate and estimate parameters of the function.

4.8 Accurate localization using AMCL

Once a room map is chosen according to matching results, TurtleBot will accomplish self-localization in the room with AMCL[22]. The steps of AMCL are showed in Figure 20.

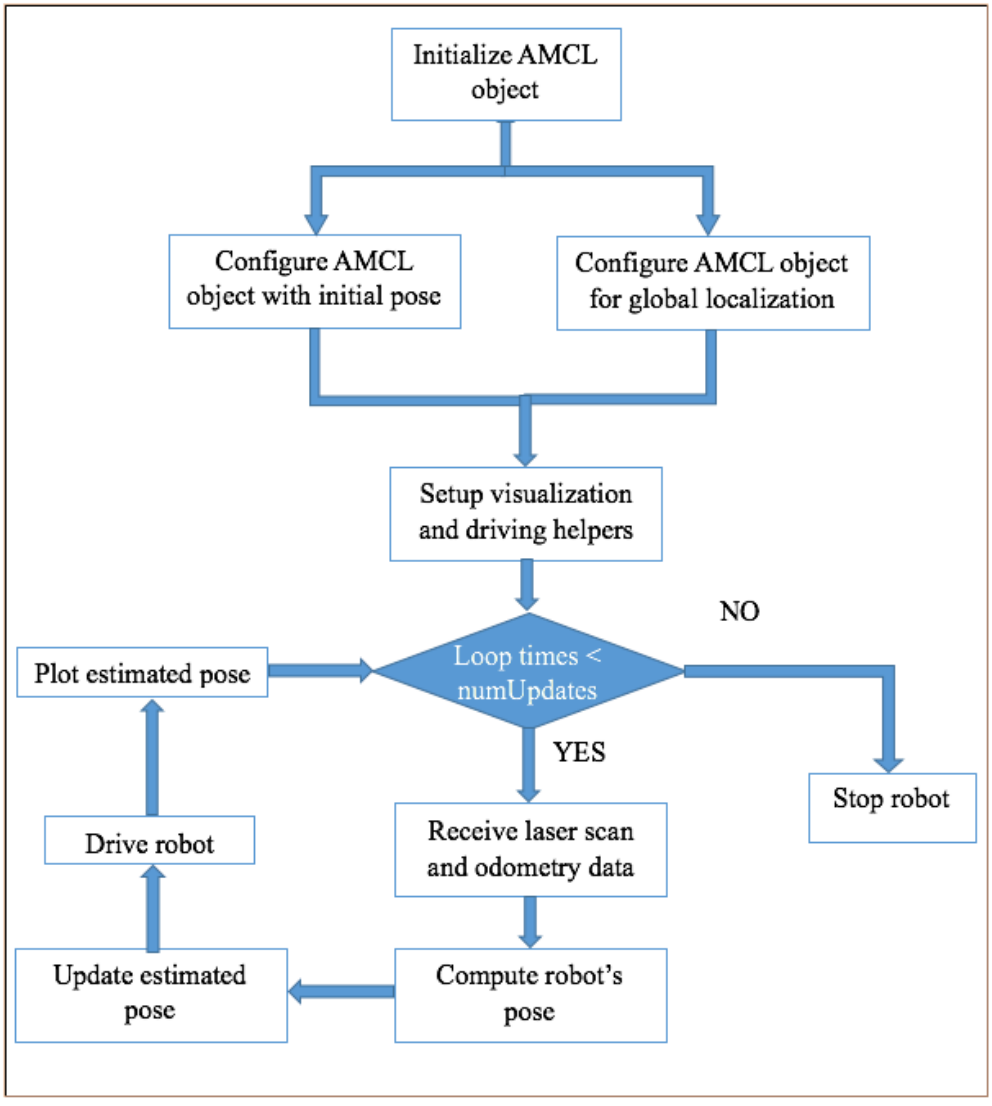

Figure 20: Process of AMCL

Initialize an AMCL object *amcl* using *robotics.MonteCarloLocalization*[6] and assign the motion model and sensor model properties in the *amcl* object. There are two kinds of localization, local localization and global localization. For local localization, initial pose estimate should be obtained by retrieving the TurtleBot's current true pose from Gazebo. Global localization needs significantly more particles compared to localization with initial pose estimate. If TurtleBot knows its initial pose with some uncertainty, such additional information can help AMCL localize faster with less particles. Then setup visualization to plot the map and update robot's estimated pose, particles and laser scan readings on the map. In the project, drive the TurtleBot randomly inside the room while avoiding obstacles using some classes in robot toolbox.

During the localization procedure, AMCL is updated with odometry and sensor readings at every time step when TurtleBot is moving around. After receiving data from ROS topics, update the estimated TurtleBot's pose and covariance using new odometry and sensor reading. When the loop time is less than update time, stop the robot and shut down ROS in Matlab.

[6] A object class in Robotics Systems Toolbox in Matlab

Chapter 5: Results

In this chapter, the environment of the programming framework is being showed. Besides, the running effect of the algorithm will also be showed.

The simulation environment is showed as below.

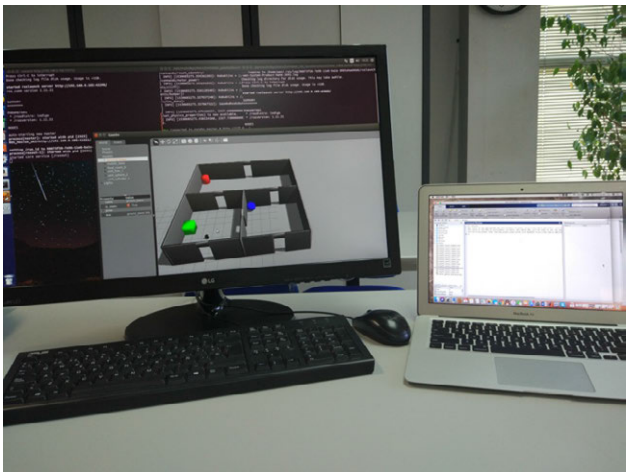

Figure 21: Simulation environment

ROS and Gazebo are installed in the left desktop computer and Matlab is installed in the right laptop. They are connected under the same network.

5.1 Test1: SIFT

The simulation environment is showed as below. The left computer is the desktop computer where Ubuntu, ROS and Gazebo are installed, and the right one is laptop where Matlab is installed.

To do self-localization, the first step is to decide which room TurtleBot is with SIFT. Drive TurtleBot with keyboard of MAC to room1/ room2/ room3 in turn. In every room, drive it to a casual position, then TurtleBot will take a picture of what it sees and match this picture with pictures in database in turn at the same time of moving randomly, until the match is successful.

In the Gazebo world we build, the objects in the three room are green box, blue sphere and red cylinder. The matching results are showed below.

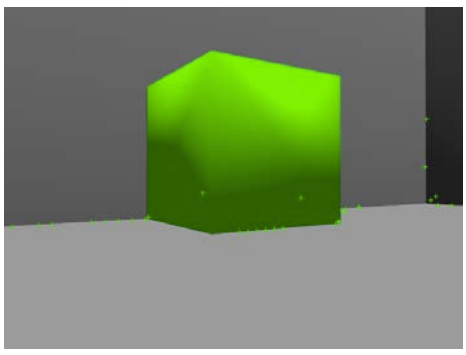

Figure 22: Green box in room1

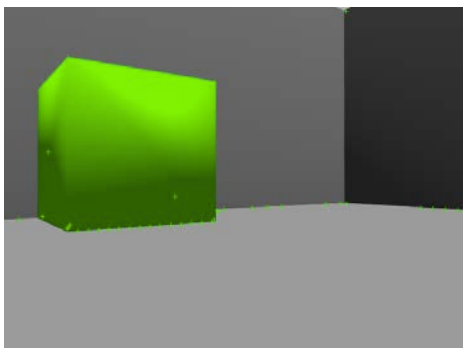

Figure 23: Real time image in room1

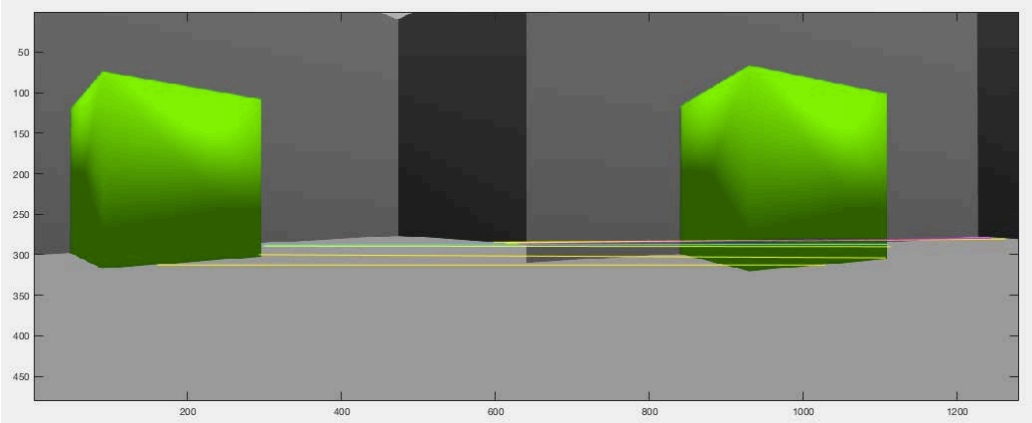

Figure 24: Matching result of room1

Figure 22 shows the green box picture saved in the database as the reference of room1. The green points on it are feature points detected and described by SIFT. Figure 23 is the real time picture that TurtleBot takes, which is matched successfully with Figure 22. Figure 24 shows the matching results of the two pictures. As we can see, there are seven matching points between them.

The matching results of the other two rooms are showed from Figure 25 to Figure 30.

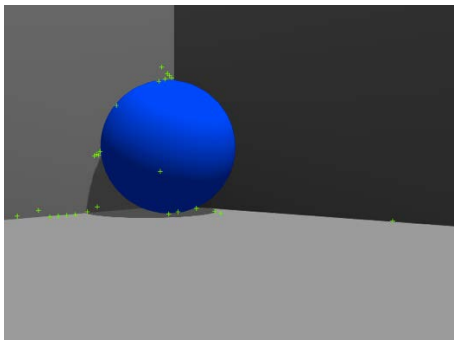

Figure 25: Blue sphere in room2

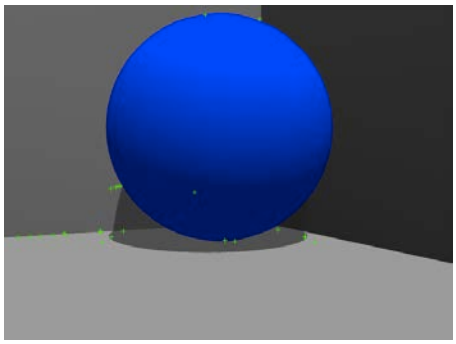

Figure 26: Real time image in room2

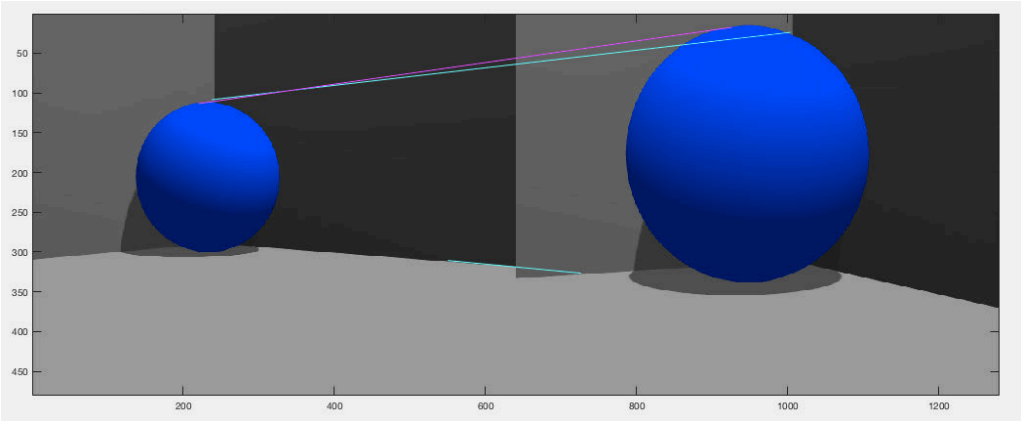

Figure 27: Matching result of room2

As Figure 27 shows, SIFT works well with different scaling of the box.

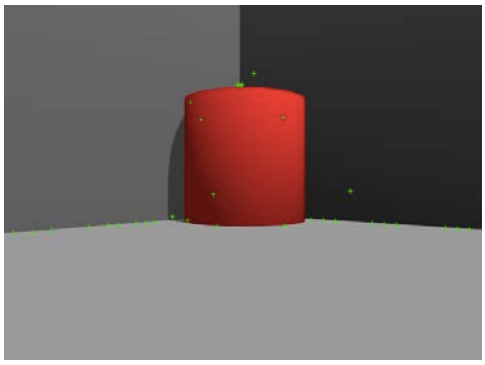

Figure 28: Red cylinder in room3

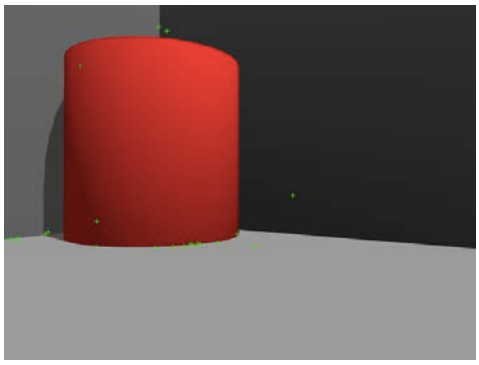

Figure 29: Real time image in room3

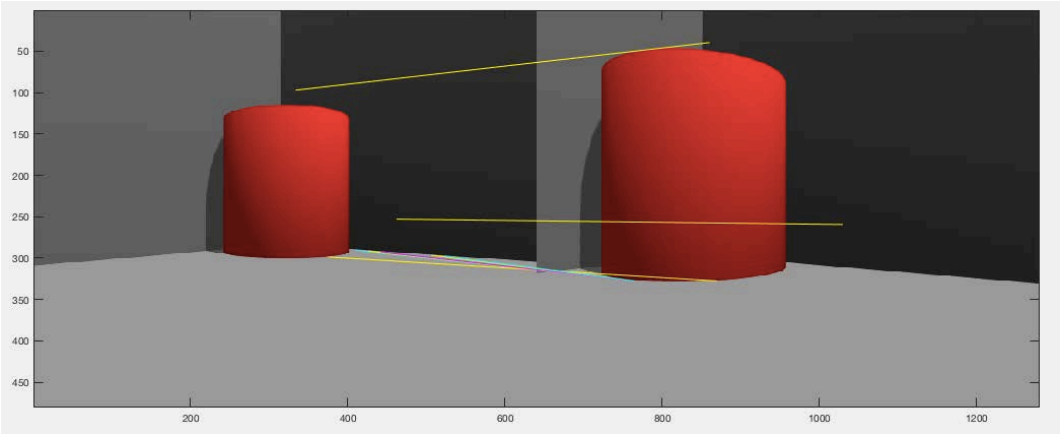

Figure 30: Matching result of room3

5.2 Test2: Local localization

After checking which room TurtleBot is with SIFT, the map of this room will be extracted from the map database we have built, and then used in AMCL.

For local localization, as we introduce before, the initial pose of Turtlebot should be given. In the simulation, we retrieve TurtleBot's initial true pose under base_link frame of ROS form Gazebo. Set amcl.GlobalLocalization to false and specify the lower and upper bound on the number of particles, which will be generated during the localization process.

The following pictures are the localization process in room1.

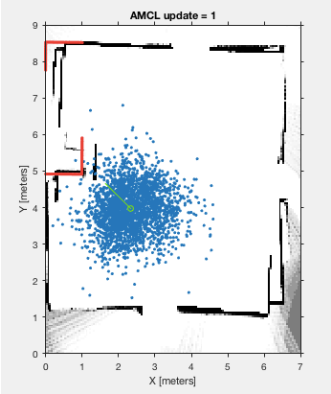

Figure 31: Update 1 in room1

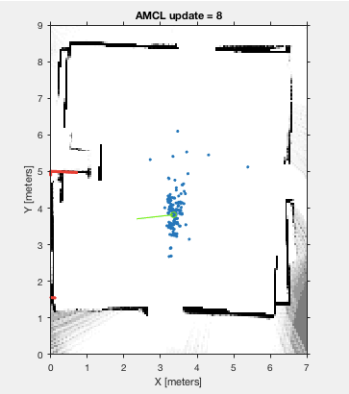

Figure 32:Update 8 in room1

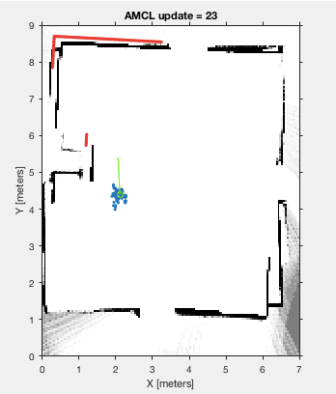

Figure 33: Update 23 in room1

In these pictures, the blue points are particles representing estimated locations of TurtleBot. The red lines are scanning results of laser on TurtleBot, and the green line represents the estimated orientation. AMCL is updated with data of odometry and laser while TurtleBot is moving.

Figure 31 is the first update of TurtleBot's poses. These particles are generated by sampling Gaussian distribution with mean equal to the initial pose gotten from Gazebo and covariance equal to amcl.InitialCovariance set before.

As we can see, the particles begin to converge to areas with higher likelihood after the eighth updates. After the 23 updates, the pose of TurtleBot gather at a certain area and we can get the pose of TurtleBot now.

Figure 34 shows the screenshot from Gazebo containing the real pose of Turtlebot.

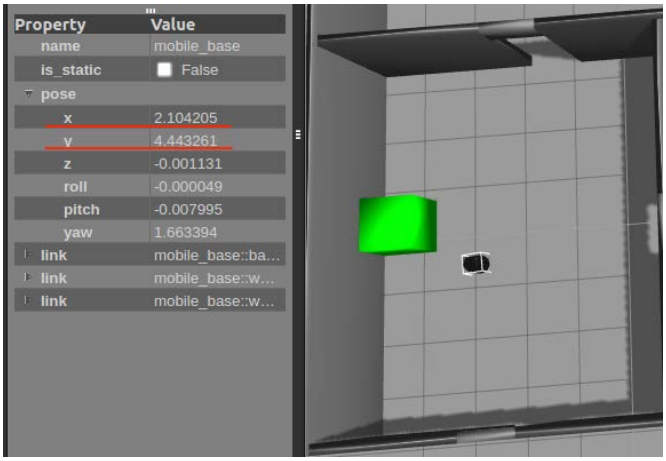

Figure 34: Real pose of TurtleBot in Gazebo

Comparing the real pose of Turtlebot in Figure 34 and the estimated pose in Figure 33, we can find that the localization is accurate in some extent.

The local localization results and real pose of TurtleBot in room2 are showed from Figure 35 to Figure 38.

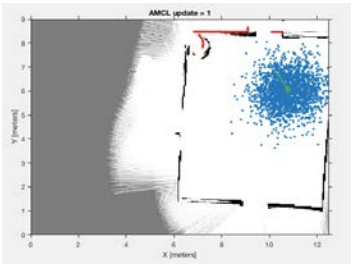

Figure 35: Update 1 in room2

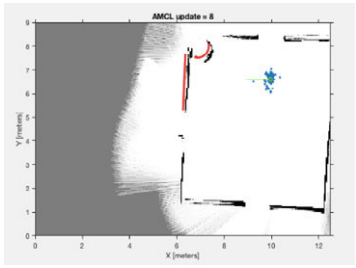

Figure 36: Update 8 in room2

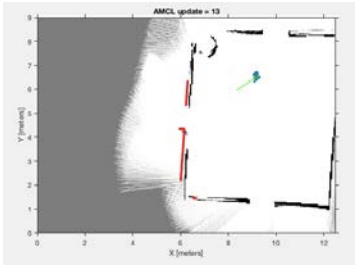

Figure 37:Update 13 in room2

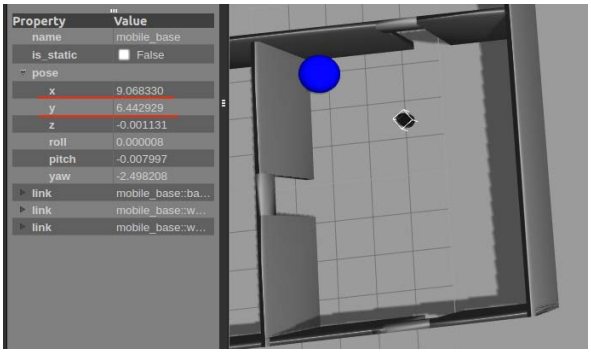

Figure 38: Real pose of TurtleBot in rooom2

The local localization results and real pose of TurtleBot in room3 are showed from Figure 39 to Figure 42.

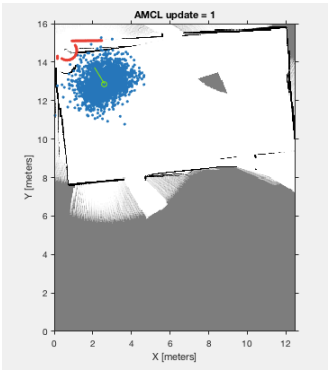

Figure 39: Update 1 in room3

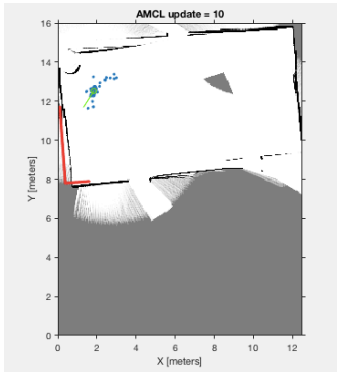

Figure 40: Update 10 in room3

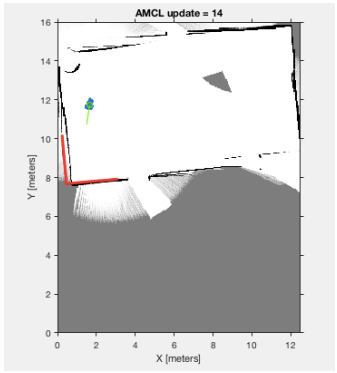

Figure 41: Update 14 in room3

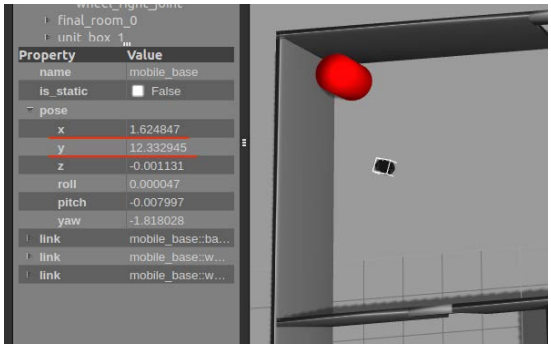

Figure 42: Real pose of TurtleBot in room3

As these pictures show, the global localization with AMCL works successfully.

5.3 Test3: Global localization

Like local localization, global localization will be implemented with AMCL after running SIFT. In global localization, AMCL tries to localize TurtleBot without initial pose. At the beginning, the algorithm assumes that TurtleBot has the same probability of any position in the room, and generates evenly distributed particles in the environment. Therefore, compared to local localization, global localization needs more particles.

Set the amcl.GlobalLocalization to ture and make the particle number limits bigger.

The following pictures are the global localization in room1.

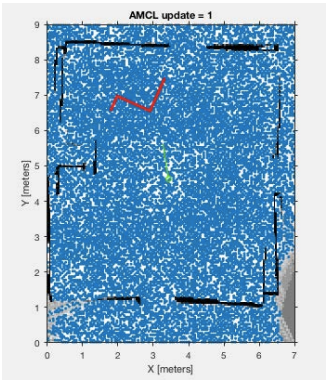

Figure 43: Update 1 in room1

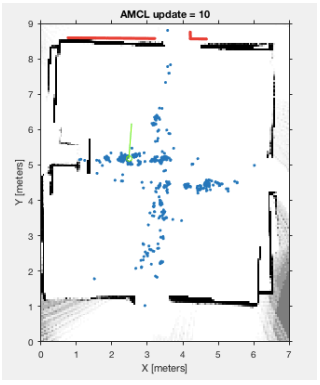

Figure 44: Update 10 in room1

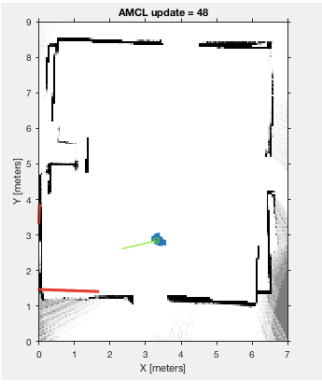

Figure 45: Update 48 in room1

As Figure 43 shows, the whole map is full of particles in the first step of global localization. The particles converge to a certain area after the 48 updates, which means global localization algorithm also needs more running time to ensure the actual pose of TurtleBot.

Figures 46 shows the real pose of TurtelBot in Gazebo.

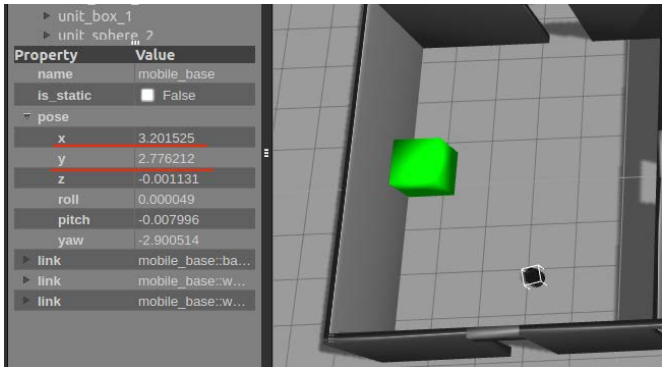

Figure 46: Real pose of Turtlebot in room1

After comparing the real pose and the final estimated pose in Figure 45, we can find the global localization works well.
The global localization results in the other two rooms are showed below.

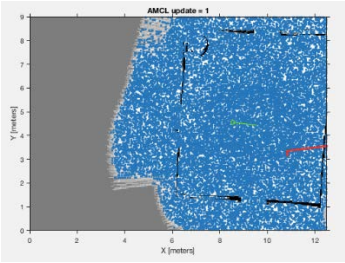

Figure 47: Update 1 in room2

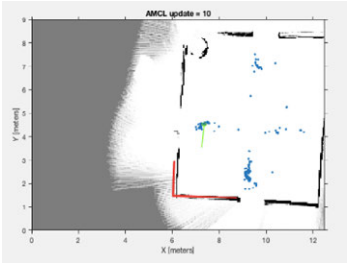

Figure 48: Update 10 in room2

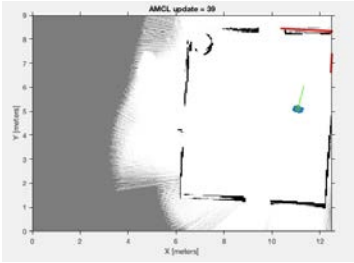

Figure 49: Update 39 in room2

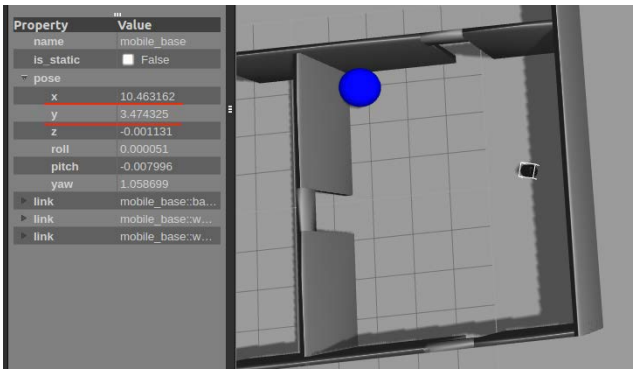

Figure 50: Real pose of TurtleBot in room2

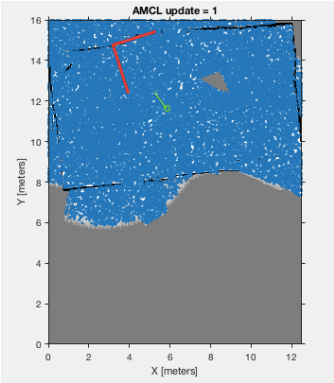

Figure 51: Update 1 in room3

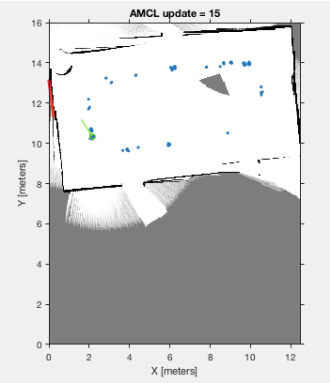

Figure 52: Update 15 in room3

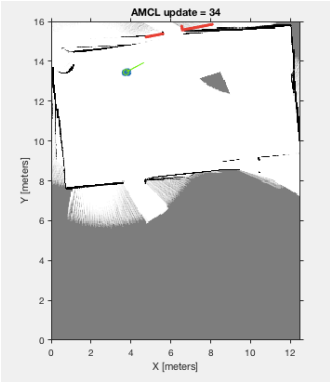

Figure 53: Update 34 in room3

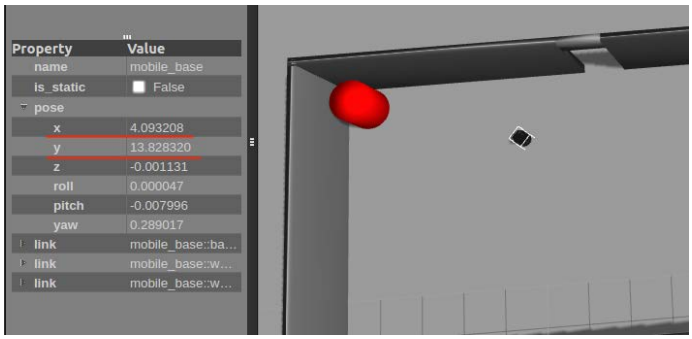

Figure 54: Real pose of TurtleBot in room3

One thing needed to note is that the global localization in room3 doesn't work successfully until third simulation. Figure 55 shows the layout of room3.

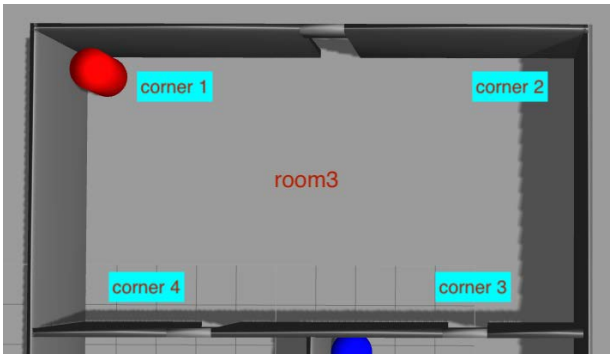

Figure 55: Layout of room3

As you can see, the areas around corner 2, 3 and 4 are quite similar. When do global localization, the particles are distributed everywhere, and the resampling area depends on the laser scan and odometry data. Thus, it is possible to misjudge which corner TurtleBot is.

Chapter 6: Budget

Chapter 6 Budget

In this chapter, the cost of developing the degree project detailed in the previous sections is being estimated and broken down. The costs have been grouped in two classes: direct costs and indirect costs.

6.1 Direct costs

In general, the direct costs contain personal cost and material cost. The following table shows the direct costs breakdown.

Personal		26 899,40 € (1 year)	11208,10 € (5 months)
Material	ROS	free	
	Gazebo	free	
	Matlab	800,00 € (1 year)	333,33 € (5 months)
	Computer	1000,00 € (5 year)	83,33 € (5 months)
Total		11624,76 €	

Table 5: Direct costs

As you can see in Table 1, ROS and Gazebo are free to download and use. For personal salary, it is about 26 899,40 euros per year, if we consider the tables provided by Universidad Politécnica de Madrid for a graduate professional. The cost of 5 months is then 11208,10 euros because it took 5 months to the author to develop the proof-of-concept described in previous sections. The Matlab's cost in 5 months is about 333,33 euros according to information in the Matlab official website. Except the laptop that the author owns, the cost of the desktop computer, a powerful one with large memory and high running speed to meet the requirements of ROS and Gazebo, is about 1000 euros, and the average in per 5 months is about 83,33 euros, considering a period of 5 years for amortization.

6.2 Indirect costs

The indirect costs include any other costs associated to any developing activity like insurances, electricity used in lighting, air conditioning, computer power and so on, and heating equipment, etc. In this case, they have been estimated as a 15% of the direct costs, according to the rule used for funding research projects in the H2020 European Research Program. That is to say, about 1743,71 euros.

In conclusion, the total cost of this project is 13368,47 euros.

Chapter 7: Conclusion

7.1 Personal conclusion of work

This project is intended to program based on ROS to achieve self-localization for robots. We combine SIFT and AMCL to solve this problem. SIFT aims at extracting and matching features between real time image from TurtleBot and pictures in database. After checking which room TurtleBot is with SIFT, use AMCL to finish accurate localization. Meanwhile, combine Matlab and ROS to improve the efficiency of processing data and make the localization process and results visible. According to the results in chapter 5, this localization method is quite successful.

When we use the keyboard of MAC to control TurtleBot to move, the reaction of Turtlebot is not pretty flexible. The reason might be that the ROS version we use is ROS Indigo released in 2014, and the Matlab version we use is Matlab 2017b released in 2017. The connection might be better if we choose more matched versions of ROS and Matlab.

In addition, Gazebo is a full-featured 3D physics simulator, but it requires high memory and graphics card, and it is not so easy to use for beginners. It is also suggested that the connection model between ROS and Matlab is more suitable for someone proficient in ROS. For beginners, it is better to learn ROS well first then try the connection model.

7.2 Future work

Considering the finished work and the problem met during the simulation, there are some future work which cloud help to improve the self-localization for robots.

- The global localization in room3 doesn't work successfully until the third simulation, because the areas around three corners of room3 are too similar. How to solve the similarity of the environment is also an important future work.
- As we illustrate before, the combination between SIFT and AMCL aims to improve the efficiency of localization algorithm. It is better to compare the efficiency of the two algorithms' combination and the efficiency of AMCL only to prove our hypothesis. In addition, try and compare other algorithms managing images and robot self-localization.
- There are only three rooms in Gazebo 3D model to test the localization algorithm. It is better to test the algorithm in mote complex environment, more rooms, more objects in the room, …. Even you can create the same objects but different colors or different sizes and then test and improve the algorithm.

References

[1] The forecast of the global Industry 4.0 and robot development status (1), http://robot.ofweek.com/2016-12/ART-8321202-8420-30079022.html
[2] The forecast of the global Industry 4.0 and robot development status (2), http://robot.ofweek.com/2016-12/ART-8321202-8420-30079044.html
[3] About ROS, http://www.ros.org/about-ros/
[4] YoonSeok Pyo et al, ROS Robot Programming, ROBOTIS Co., Ltd., 2017
[5] ROS Tutorials, http://wiki.ros.org/ROS/Tutorials
[6] We Want You to Learn TurtleBot in Simulation, http://learn.turtlebot.com/2015/02/03/1/
[7] Gazebo Tutorials, Connect to ROS, http://gazebosim.org/tutorials?cat=connect_ros
[8] What is a TurtleBot, https://www.turtlebot.com/
[9] Robots/TurtleBot, http: //wiki.ros.org/Robots/TurtleBot.
[10] A review of research on indoor localization methods for autonomous mobile robots, 李群明, 熊蓉, 褚健, 2003, 25(6): 560-567,573.
[11] S. Thrun, W. Burgard and D. Fox, Probabilistic Robotics. Cambridge, MA: MIT Press, 2005.
[12] The analysis of extracting SIFT's features, https://blog.csdn.net/abcjennifer/article/details/7639681
[13] Lowe D G. Distinctive image features from scale-invariant keypoints[J]. International journal of computer vision, 2004, 60(2): 91-110.
[14] Introduction to SIFT (Scale-Invariant Feature Transform), https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
[15] Robotics System Toolbox, https://www.mathworks.com/help/robotics/index.html
[16] Connect to a ROS Network https://www.mathworks.com/help/robotics/examples/connect-to-a-ros-network.html
[17] Building a world, http://gazebosim.org/tutorials?tut=build_world
[18] Building editor, http://gazebosim.org/tutorials?tut=building_editor
[19] cn/ indigo/ Installation, http://wiki.ros.org/cn/indigo/Installation
[20] Installing gazebo_ros_pkgs, http://gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros.
[21] Mapping With Known Poses https://www.mathworks.com/help/robotics/examples/mapping-with-known-poses.html
[22] Localize TurtleBot Using Monte Carlo Localization https://www.mathworks.com/help/robotics/examples/localize-turtlebot-using-monte-carlo-localization.html