TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MOJTABA HEIDARYSAFA

Heuristic localization and mapping for active sensing with humanoid robot NAO

Master of Science thesis

Examiners: Prof. Risto Ritala, Prof. Jose Martinez Lastra
Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sceince
on 5.11.2014

ABSTRACT

MOJTABA HEIDARYSAFA: Heuristic localization and mapping for active sensing with humanoid robot NAO
Tampere University of Technology
Master of Science Thesis, 67 pages, 8 Appendix pages
March 2015
Master's Degree Program in Machine Automation
Major: Factory Automation
Examiner: Professor Risto Ritala, Professor Jose Martinez lastra

Keywords: AMR, Humanoid Robot, NAO, Localization, Mapping, Path planning, Active sensing

Autonomous mobile robots (AMR) have gained great attention between researches in last decades. Different platforms and algorithms has been proposed to perform such a task for different type of sensors on a large variety of robots such as aerial, underwater and ground robots.

The purpose of this thesis is to utilize vision system for autonomous navigation. The platform which has been used was NAO humanoid robot. More specifically, NAO cameras and its makers have been used to solve the two most fundamental problems of autonomous mobile robots which are localization and mapping the environment. NAO markers have been printed and positioned on virtual walls to construct an experimental environment to investigate proposed localization and mapping methods.

In algorithm side, basically NAO uses two known markers to localize itself and averages over all location predicted using each pair of known markers. At the same time NAO calculates the location of any unknown markers and add it to the Map. Moreover, A simple go-to-goal path planning algorithm has been implemented to provide a continuous localization and mapping for longer walks of NAO.

The result of this work shows that NAO can navigate in an experimental environment using only its marker and camera and reach a predefined target location successfully. Also, It has been shown that NAO can locate itself with acceptable accuracy and make a feature-based map of markers at each location.

This thesis provides a starting point for experimenting with different algorithms in path planning as well as possibility to investigate active sensing methods. Furthermore, the possibility of combining other features with NAO marker can be investigated to provide even more accurate result.

PREFACE

This work has been accomplished in the department of Automation Science and Engineering. Several individuals made this work possible through their generous guidance and contributions that I would like to thank here.

First and foremost, My special thanks goes to Prof. Risto Ritala, my thesis supervisor, which supported me by all means during the time of this work and guide me through this writing by proofreading this thesis.

Also, I would like to thank Mikko Lauri which helped me with any questions during this works and Joonas Melin for his advice and thechnical helps that made this work possible.

Last but not least, I would like to thank Prof. Jose Lastra, the head of Factory Automation program that allowed me to participate in this work and supported me with all means possible.

Tampere, 10.3.2015

Mojtaba Heidarysafa

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Autonomous Mobile Robots (AMR) have gained more attention in recent decades with the growth of technologies and they are expected to contribute increasingly to our daily life. More researchers have been interested in the field and its potentials, especially in Artificial intelligence (AI). AMRs can be used in many applications from military and space exploration to human assistant in hospitals, museum, etc. In order to reach full autonomy, AMRs must utilize and combine a variety of functions to have abilities such as navigation, exploration, etc.

Among these abilities, is the ability to navigate in an unknown, partially known or known environments. To be able to do this a robot should be able to localize itself and map the environment in the case of a partially known or unknown environments. In the past decades different approaches have been proposed to successfully provide solutions for localization of a robot and mapping of an environment based on different types of sensors. Most common sensors utilized for these solutions are sonars, lasers and cameras. The focus of this work was to provide a vision based solution for localization and mapping problem for the NAO humanoid robot.

Different humanoid robots have been developed in last decades. Two of the most world-known humanoid robots are ASIMO by Honda and NAO by Aldebaran. Despite of introducing more uncertainty as a result of biped walks, these robots gain lots of attention as autonomous robots based on their similarity to human. Applications such as human assistance, makes humanoid robots attractive platforms for research of autonomous mobile systems in future.

1.1 Objective

The main objective of this thesis work is to provide a solution for localization and mapping of NAO humanoid robot. The task can be divided into the following subtasks:

- Localize NAO using its monocular vision
- Create a feature-based map of a partially known environment
- Implement a simple path planning scenario to examine the accuracy of proposed localization and mapping approach

The implementation suggested by this work can be utilized in further research, such as navigation in an indoor environment as an assistant robot, implementing more elaborate path planning, etc.

Major contributions accomplished in this thesis can be listed as follows:

1. A localization method has been developed based on NAO's monocular vision using NAO's Markers.
2. A feature-based map has been developed using NAO markers in a partially known environment.
3. A full python code specially for NAO has been implmentend and can be utilized in further research by using this methodology as a starting point.

1.2 Thesis structure

This thesis consists of six chapters as follows:

Chapter 1 presents motivation, objective and contributions achieved during this project.

In chapter 2, the theoretical background has been presented for better grasping the subject. This chapter provides a general overview about Autonomous Mobile Robots and their differences. It describes the two main types of such robots, i.e wheeled robots and legged robots and compares them. Furthermore, the autonomous mobile navigation problem and its main component, i.e. localization and mapping, as well as active sensing has been explained.

Chapter 3 describes the NAO as the platform of this work and presents an overview of NAO's structure. The information in this chapter gives a more detailed view of NAO's hardware and software. It describes NAO's sensors and actuators and general information about the platform. Furthermore, it explains the ways of programming NAO as well as the approach selected for programing NAO in this work.

Chapter 4 reviews the methodology which has been used for this work. It describes the NAO markers and the information received by observing them. Furthermore, it presents the feature-based mapping approach and a simple go-to-goal behavior and provides a view of programming structure for this approach.

In Chapter 5, the results of experiments performed with the robot are presented. It covers the results of robot motion experiment as well as approach developed for localization. The chapter also provides the results of feature-based mapping during a go-to-goal experiment.

Finally the results of this work are concluded in Chapter 6 and future work is proposed based on the achievements of this project.

2 THEORETICAL BACKGROUND

The focus of this section is to provide an understanding about previous work, the state-of-the-art, and the approaches by other researchers in mobile robotics. The section contains three subsections. In the first part, a literature review on land-based mobile robots is presented. It is followed by a survey of solutions for localization and mapping in an environment. The last section discusses optimal sensing methodologies.

2.1 Autonomous Mobile Robots

Autonomous Robots are platforms for applications such as navigation and exploration. The ultimate goal of an autonomous robot is to navigate in an unknown unbound environment and to accomplish on its own high-level tasks assigned to it. The wide list of mobile robot tasks covers e.g. house cleaning, space exploration, rescue mission as well as military operations. Many classifications exist for mobile robots. One classification of mobile robots is based on the environment in which the robot operates: mobile robots are categorized into land-based, air-based and water-based robots.

Land-based robots are robots which traverse and operate on the ground surface. These robots are also called Unmanned Ground Vehicles (UGVs) [1]. Such robots have a large variety of applications, such as health care for elderly people, military assistance, and entertainment.

Air-based robots operate in the air without a human pilot, see Figure 2.1a. These types of robots are also called Unmanned Aerial Vehicles (UAVs) [2]. The applications such as mapping the ground and military operations are common for UAVs. The most usual types of UAVs are planes, quadcopter and blips.

Water-based robots refer to ones which traverse under water autonomously, and are called autonomous underwater vehicles (AUVs), see Figure 2.1b. Equipments such as self-contained propulsion, and sensors assisted with artificial intelligence allow AUVs to perform sampling or exploration tasks underwater with no or little human intervention [3].


Figure 2.1. a: GPS-enabled PHANTOM quadcopter (left) b: AQUA underwater robot (right)

"Localization" and "mapping" are the two main prerequisites for almost all mobile robotic actions and thus have been investigated for all types of the robots mentioned above. Commonly a task requires these functions to be implemented at the same time leading to "simultaneous localization and mapping" (SLAM) for autonomous robot's tasks.

Land-based robots are the most popular type of autonomous robots between the three types explained above. Land-based robots can further be categorized based on their motion method as wheeled robots, legged robots and snake-like robots.

A wheeled robot utilizes motor-driven wheels to travel across the ground surface. Wheeled robots are most common for navigation on smooth surfaces due to simple design and control in comparison to the legged robots. Wheels are the simplest solutions for robot locomotion. As a result of these advantages of wheeled robot, there exists a large variety of designs for wheeled robots. One way to categorize wheeled robots is based on the number of wheels they use for the locomotion.

When the robot has only two motorized wheels the main problem is to keep the balance and stay upright during its movements. This inherent instability of the robot requires an accurate control of the two wheels. A good design for two wheeled robots has a low center of gravity. One way to do so is to mount batteries under the robot frame. An example of two-wheeled robots is Nbot (Figure 2.2) which balances itself using inertia unit and encoder data.

Figure 2.2. Two wheeled robot Nbot

Three-wheeled robots move with two motor-driven and a free turning wheel. Usually the arrangement of these wheels is triangular to keep the robot balanced. Figure 2.3 shows an example of this arrangement. A good practice is to keep the center of gravity close to the center of the triangular design in order to stabilize movements. Turning can be generated with wheel rotation driven at different rates while for the straight movement both wheels rotate at the same rate.

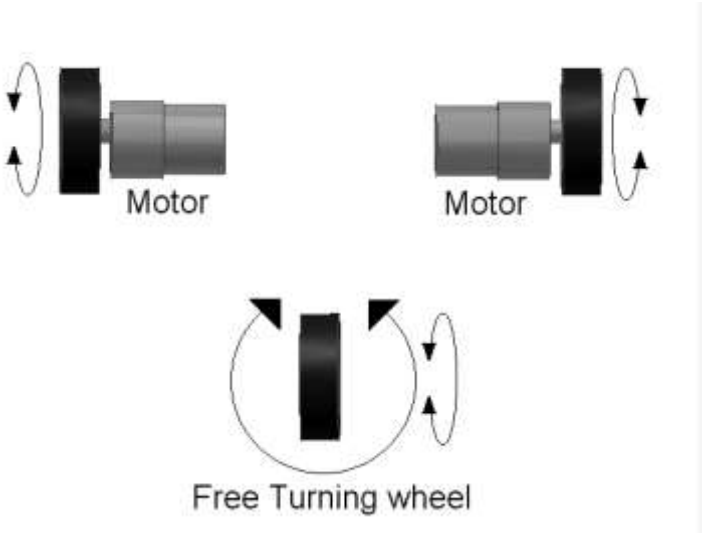

Figure 2.3. Arrangement of wheels in three wheeled robot

Furthermore, there are four wheeled robots which can be front, back or four wheel driven. The robot is steered by turning front or back wheels as pairs or both. ASE laboratory for intelligent sensing has as an experimental platform a four-wheeled robot produced by Robotnik. An example of an innovative idea is

the Mecanum wheel robot, see Figure 2.4. This type of wheel provides the ability of omnidirectional movements.

Figure 2.4. URANUS omni-directional mobile robot

Legged robots use mechanical legs or leg-shaped instruments to move. A legged robot if designed properly can give a better locomotion on rough and uneven surfaces than any wheeled robot. The number of legs in this type of robots can vary from two to eight and each leg needs at least two degrees of freedom (DOF) to allow mobility. Each degree of freedom corresponds to one joint which is usually powered by a servo.

Two-legged or bi-pedal robots use the same mechanics as human beings to walk. The similarity between these robots' locomotion and human movement has made them an interesting robot platform in the recent decades for many studies. Autonomous human-like robots with the ability to do human tasks has been a focus area of recent robotic research. Two-legged robots provide a platform to study human cognition [4]. In recent years, several companies have been involved in building humanoid robots. Honda was one of the first companies with P1-3 series of robots. Later on Honda introduced ASIMO with the ability to run. Another very good research platform is NAO robot developed by Aldebaran Company. The robot has the ability to do similar tasks to human such as picking up objects, listening and talking, and even dancing. ASE laboratory for intelligent sensing has a NAO robot.

A four-legged robot imitates the locomotion from four legged animals in nature. An example of such a robot is BigDog robot by Boston Dynamics shown in Figure 2.5. The control algorithm for this robot allows walking on snow-covered hills and even on ice surfaces [5]. Boston Dynamics introduced other robots such as Cheetah which can run at a speed of 29mph. Different methods can be used for walking of these robots, such as alternative pairs and opposite pairs.

Figure 2.5. BigDog on snow-covered hill

There are robots which use even more than four legs for their locomotion. One design for such robots is a six-legged robot. This design provides easy solution for walking as it can be controlled using static walking methods rather than dynamic methods. Similarly to four-legged robots, six-legged robots can be considered as inspired by nature. Many insects move on surfaces with six legs. Two major gaits models for such robots are waive gait and tripod gait which are presented in Figures 2.6 and 2.7 [6].

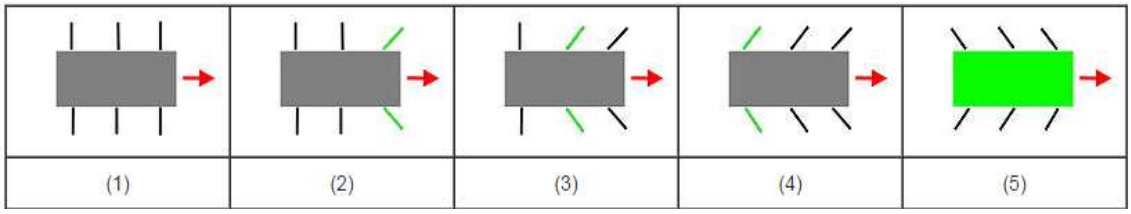

Figure 2.6. Waive gait. (1) Neutral position. (2) Front pair moves forward. (3) Second pair moves forward. (4) Third pair moves forward. (5) Body moves forward.

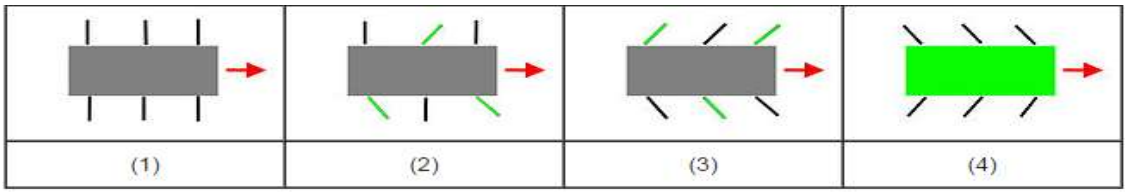

Figure 2.7. Tripod gait. (1) Neutral position. (2) Three alternating legs move forward. (3) The other set of three legs moves forward. (4) Body moves forward.

The advantages of the two main robot types can be summarized as follows:

- Wheeled robots:
    1. Simpler design and control
    2. Lower cost
    3. Possibility to customize with respect to the task
- Legged robots:
    1. More complicated designs are possible
    2. Ability to move in more rough-terrain and e.g. in stairs.

2.2 Autonomous Robot navigation problem

An autonomous robot refers to a robot which is able to navigate on its own, without any human intervention. A definition for navigation is given by R.Montello who describes it as "coordinated and goal-oriented movement of one's body through an environment" [7]. Generally speaking a robot needs to navigate either in a known environment or in a partially known environment.

The question that robot needs to answer eventually is "how can I go from where I am now to a point where I desire to be?" which deals with path planning and exploration portion of the autonomous robot problem. In order to answer to this question, the robot needs to know the answer to two other questions as well. The first question is "where am I?" which is the essence of the localization problem. The second question would be "how does the environment look like?" or more specifically "what objects exist in the environment and where are they relative to my current location". This type of questions refers to mapping of the environment by a robot.

Simultaneous localization and mapping (SLAM) is the cornerstone of autonomous navigation because maps about the environment are quite often incomplete. Combinations of localization, mapping and robot motion lead to areas of robotics as shown in Figure 2.8 [8]. This Figure emphasizes that motion/path of the robot can be chosen so that it is advantageous for localization, mapping or SLAM.

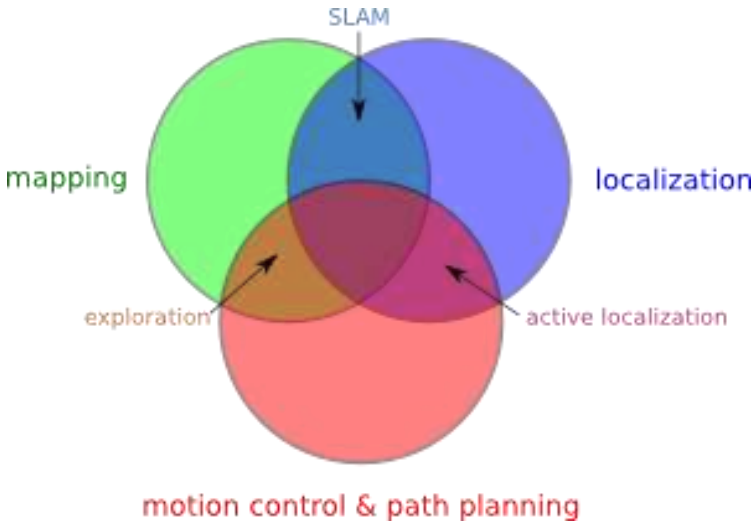

Figure 2.8. Main areas of autonomous mobile robotics and their relationships

This thesis will address localization and mapping. Furthermore, it discusses active sensing which is a union of active localization and exploration sections in Figure 2.8. Thus, a deeper look into all these areas is presented in the following sections as a literature review.

2.2.1 Localization

The task of localization can be described as finding an estimate of the position and orientation of a robot, the robot pose, with respect to a pre-defined global coordinate system. In localization it is assumed that there exists a map of the world with sufficiently many objects of exactly known locations. The basic form of localization can be considered dead reckoning where the position of the robot can be calculated from the previous position and the amount of robot movement from that position. Therefore a motion model in combination with a measurement system of motions such as inertia measurement unit (IMU) are two important aspects of a localization solution.

This approach has its own flaws as the accumulated error during time will increase the uncertainty of position estimation and even might lead to failure. In order to prevent such failure robot can observe the known objects of the environment to get extra information about its location. Such a task is another important part of a localization solution for the robot.

Figure 2.9 illustrates the localization process. A robot can find its pose according to information received from the environment. The information can be gathered from camera, laser sensor, sonar, etc. The assumption will be that the localization made from robot observation will be precise while in reality that is not the case. Observations have their own uncertainty depending on the precision of the sensors. One way to overcome this problem is to introduce a motion model and a motion measurement to reduce the uncertainty of observation.

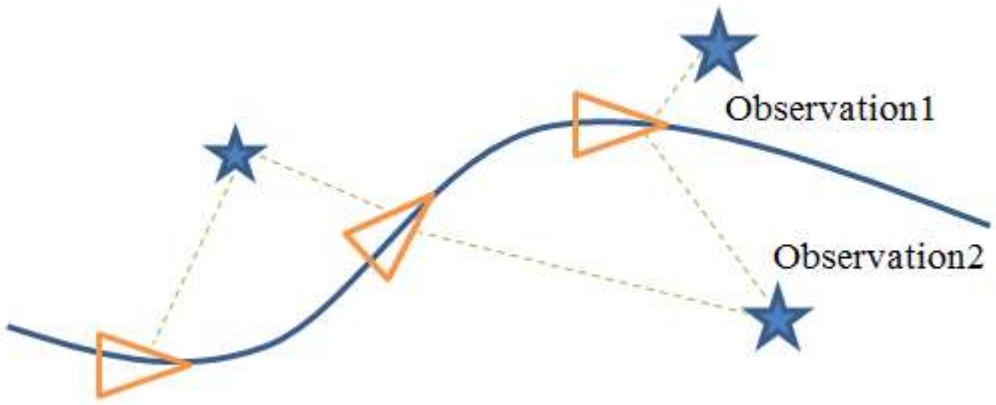

Figure 2.9. Localization schema

Accurate localization is a function of two variables. First, one should have the prior map which is accurate enough and second the observation should be as accurate as possible. In practical applications a precise prior map is not always available. Therefore, there is a need to build an environment map in many of robotic applications to do localization.

2.2.2 Environment mapping

Despite the assumption that in most robotic application the map can be provided, there are many cases that either the map is incomplete or the object locations in it are not accurate. In such scenarios the robot should be able to do the mapping autonomously. Mapping can be defined as the ability to use gathered data from robot sensors to create a description of the locations of objects in the robot's environment. In pure mapping the robot is assumed to know its pose at all times. Figure 2.10 shows a graphical model of map building [9].

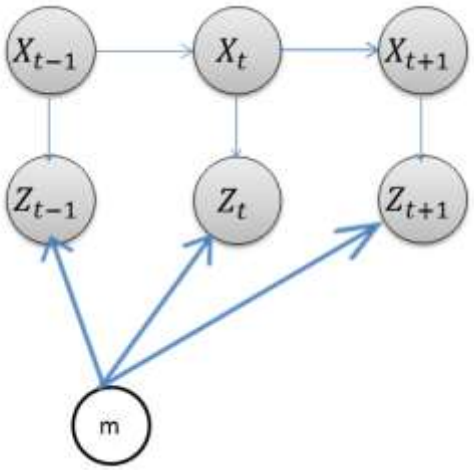

Figure 2.10. The map m is built based on distance/coordinate observations of the mapped objects and the exact information about robot pose

The graphical model represents the known pose of the robot as X. Variable Z denotes the uncertain measurement data at the each time step. Based on this model the information about the **m** is the posterior probability of **m** based on the measurement history and the corresponding known poses.

The quality of a map built in this way is very dependent on the uncertainty of observations. Even when the uncertainty of the pose of the robot is negligible – as the pure mapping assumes – the resulting map is different from reality due to measurement errors. Figure 2.11 illustrates such a situation.

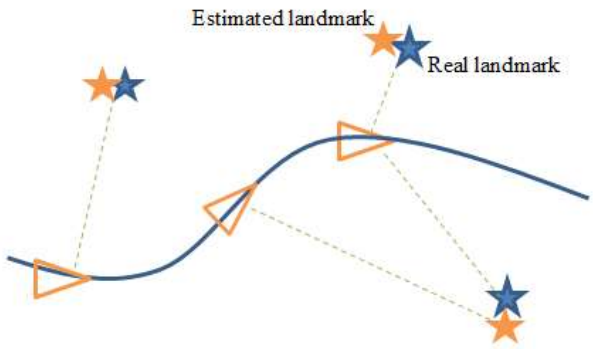

Figure 2.10. Illustration of the mapping problem with known robot pose

Landmarks can be anything from specific markers to features extracted from the images of the robot's environment. As the location of the robot is assumed known in pure mapping, the map can be expressed in an absolute form, meaning all landmarks coordinates are given in a global frame.

Representations of maps can be classified into three types: occupancy grids, feature-based maps and topological maps. Figure 2.12 shows an example of these methods.

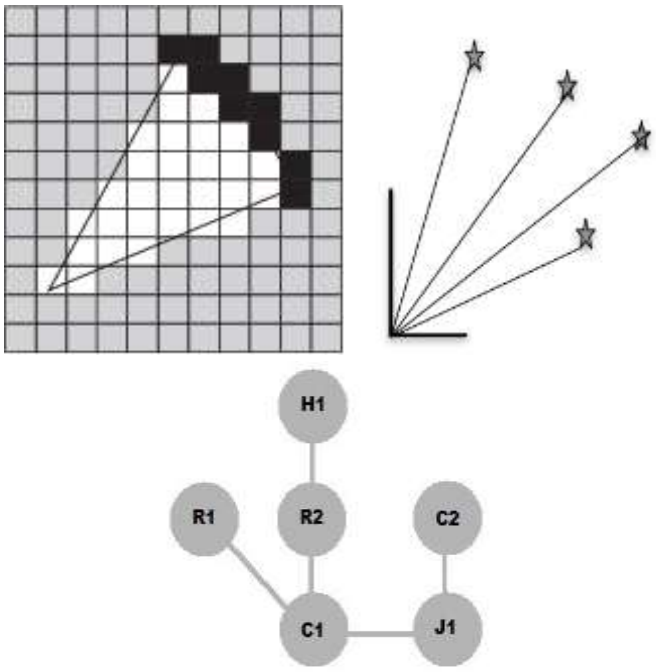

Figure 2.11. Map types. Left: occupancy grid. Right: feature-based map. Bottom: topological map.

An occupancy grid describes the robot environment as a 2D grid of binary-valued squares. Each square can be occupied by an object or free. An occupied square will be presented in black and a free square is colored in white. Squares with no information about them are presented in gray. This approach was introduced by Moravec and Elfes for sonar sensor information [10]. One advantage of occupancy grid is the possibility of combining data from different sensor scans. A disadvantage of this approach is its weak performance in large environments due to the significant increase in calculations as well as difficulties in adding new maps to the old corresponding map.

Feature-based maps represent the environment as a set of features located in a global coordinate system. These features can be things such as a feature point or a corner, a line, etc. As a result this type of map is only a sparse representation of landmarks. This approach was introduced by Smith while addressing the simultaneous localization and mapping problem [11]. The method uses robot sensor data to get landmark distance and angle and thus its position in robot frame. The cost of computation and data association problem – how to recognize which landmark is which – are the two main disadvantages of this method. Both occupancy grids and feature-based maps are considered metric maps as they use Cartesian coordinates to represent environments.

Unlike the occupancy grids and feature maps, a topological map does not represent environment as a metric map. This method uses the graphical concept by presenting environment as a set of nodes connected by links to each other. Brook [12] and Mataric [13] are considered to be the first to implement topological maps. The idea for such presentation comes from the fact that humans and animals do not create a metric map of their

environment but rather think of relationships between places. Although such maps are suited for reliable navigation, they may fail when the complexity of the environment increases.

Building an autonomous precise map without accurate localization is a considerably more complex task. Simultaneous localization and mapping (SLAM) has been studied intensively for real robotic problems. In this thesis proper SLAM is not tackled but localization and mapping are connected in a heuristic manner suited for environments about which there is good initial map information. Thus, SLAM is not reviewed here.

2.2.3 Exploration and active sensing

A good definition of exploration in mobile robots is given by Thrun as

> "It is the problem of controlling a robot so as to maximize its knowledge
>
> about the external world [9]"

Many studies consider the balance between exploration and exploitation as the core problem of autonomous robots. Exploration consists of the localization and mapping, and motion planning. In other words, the main idea of optimal exploration is that the robot path is chosen to provide the best (in prior expectation sense) localization/mapping based on the landmarks it is going to observe with its sensors.

Active sensing is closely related to the exploration tasks in robotics. Generally speaking, active sensing can be considered related to two questions of:

1. Where to focus the sensors that have operational degree of freedom, including sensing opportunities generated by robot movement?
2. How to quantify before making a particular sensing action how good is – what is the objective function when optimizing sensing actions?

While dealing with mobile robots, the second question is the more profound one. Put differently, how should the robot act so that useful information is more likely to be gathered in the future steps. Another closely related concept is known as "active localization". The difference is that active localization refers to robot motion decisions to determine its pose while active sensing is more closely related to sensing decisions during motion. However, there are cases where researchers do not make this distinction and refer to both as "active sensing" [14].

Active sensing usually involves tasks with significant uncertainties that influence the performance in the execution of tasks. Active sensing policies can be solved by model-

based methods or with reinforcement learning. An example of a model–based approach for navigation of a mobile robot can be such that a robot should move from an uncertain initial pose to a goal pose within a specific time. In reinforcement learning the robot uses sequential experiences with states and outcomes to find the best possible action.

An important taxonomy for active sensing classifies it as myopic and non-myopic adaptive sensing. Myopic sensor management is optimizing the performance on short term, one decision ahead, to keep the problem simple. Non-myopic sensor management on the other hand can be considered as a scheme that trades off costs for long-term performance [15], but this approach leads to computationally complex problems.

Long-term plan can be described as a set of actions performed in a sequence. The most general representation of optimal action sequencing is Markov Decision processes (MDP) for fully observable states and Partially Observable Markov Decision Processes (POMDP) for cases that states are not completely observable. MDP is pure planning problem wheras POMDP in mobile robotics can be considered as planning and SLAM combined.

A POMDP consists of these elements:

- A set of states
- A set of actions
- A state-transition law which describes the next state distribution after performing an action in the current state
- A reward function
- A set of possible observation
- One or several observation laws which describe the distribution of data values when the observation is made at a given state

If the actions affect which of the observations can be made, POMDP is an active sensing problem. Depending on the reward function, it is an exploitation, exploration or combined task. POMDP begins with some initial state information, a probability distribution. In this state an action is performed and a reward is received based on the action and state. After the action observation data is received based on the state and the action performed. As a result of the action the state information changes to a distribution as described by state-transition law. After receiving the data the state information is updated. The process repeats the same way and as a result, the state information evolves as probability distributions.

POMDP as a description has been proposed for many problems in autonomous robotics. The recent work at the TUT group can be considered as an example of solving active sensing for exploration with POMDP approach. The problem which is addressed in this case is “which direction the robot should focus its attention to gain maximum information”, i.e. which of the sensing actions to make or which of the observation laws to

apply. The effects of a greedy strategy and non-myopic strategy on the result have been investigated [16]. Figure 2.13 shows a simulation of an environment for this problem.

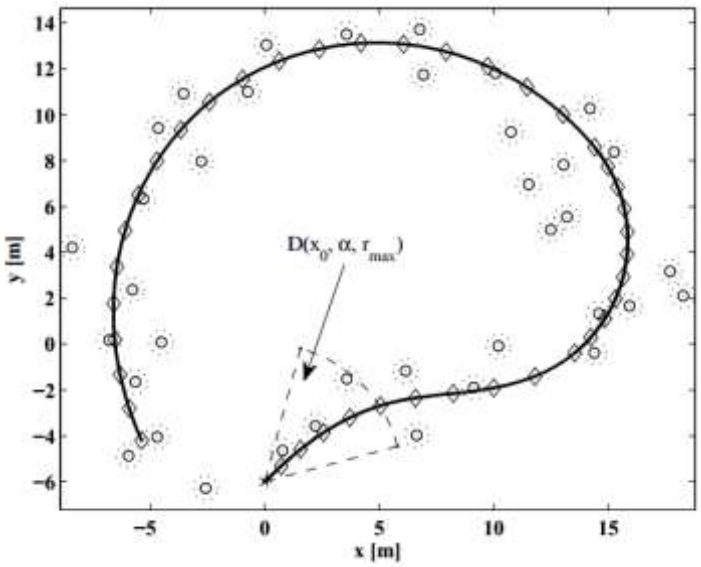

Figure 2.12. Illustration of active sensing for robot attention focus ,photo courtesy of M. Lauri

In Figure 2.13 the robot is expected to traverse along the solid line trajectory and it should make motion decisions on specific points of the trajectory. In this case, Selecting the focus of machine vision system of the robot in order to gather better information was the main area of research.

This thesis does not address directly active sensing algorithms and their implementation. While dealing with localization and mapping in this thesis, simplifying assumptions have been made.

First of all, during the localization phase, the robot only relies on the observation and does not include any information from motion model and measurements. This is because the robot walk is rather uncertain and no good walk models exist. Furthermore, the assumption is that the observation is accurate enough to localize the robot based on pure observation of the environment.

With respect to the mapping phase of this thesis, the idea is that in each measurement the same amount of information about the feature location has been gathered. Therefore, estimates are simply averaged over the previous locations of features at each step. Detailed information about this implementation will be discussed in the following sections of this thesis.

3 ROBOTIC PLATFORM NAO

The platform used for all implementations of this thesis is NAO robot from Aldebaran Robotics Company, a French company. Being reasonably priced, NAO is a good candidate for research on humanoid robots. It has been used in research and competitions since 2008 and has gained a huge attention in education and research. Aldebaran Robotics has a verity of models of NAO such as H21, H25, T14 (torso only) and T2 (torso without hands). There are different versions of products starting from V3+ to V4 [17]. The version of NAO used in this work is NAO H25 V4.0 (Full body robot), see Figure 3.1.

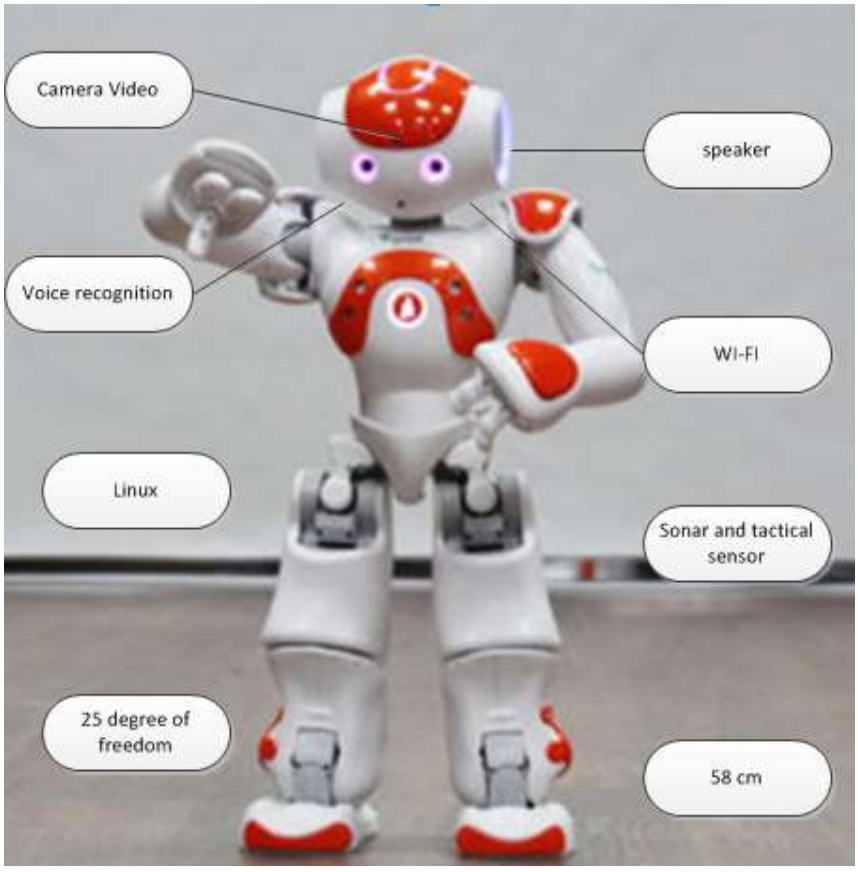

Figure 3.1. Main characteristics of NAO H-25 V4

There is an option for H25 which comes with a laser scanner on the head. However, this project relies on NAO's camera for the perception of the world. This chapter describes NAO's abilities and its structure. It includes an overview of NAO hardware and software.

3.1 NAO Robot

NAO is a 58 cm tall humanoid robot which is programmable using different languages, such as C++, JAVA, Python, .NET, MATLAB, and Urbi. NAO has been the first project of Aldebaran Robotics established in 2004 and its first product in 2007. In the same year NAO replaced the AIBO dog as the Robocop standard platform. NAO's abilities such as biped walking, sensing close objects, talking, and performing complicated tasks with the help of an on-board processor provides a platform not only to implement usual mobile robotic algorithms but also to offer a platform for research on futuristic robot ideas. Specifications of the robot are given in Table 3.1.

Table 3.1: NAO robot main characteristics

NAO V4 General Specifications	
Height	58 centimeters
Weight	4.3 kilograms
Built-in OS	Linux
Compatible OS	Windows, Mac OS, Linux
CPU	Intel Atom @ 1.6 GHz
Vision	Two HD 1280x960 cameras
Connectivity	Ethernet, Wi-Fi, infra-red

Since 2008 many universities and laboratories around the world have started to use NAO for their research. Aldebaran improved NAO from V3.2 to V4 gradually till 2011. The latter version is equipped with a better processor, HD cameras and it is more reliable in comparison to its predecessors. NAO's hardware and software structure is described in the following sections.

3.2 NAO hardware and equipment

In order to use NAO as a robotic platform, one should have good knowledge of its hardware and software. This section describes NAO H25 equipment in more details. It first presents the computer hardware of the robot, then describes its sensors, and finally the mechanical structure will be explained.

3.2.1 Hardware

NAO V4 has Intel Atom @1.6 GHz processor which is improved from the previous versions that used AMD geode 550Mhz. The built-in processor is located in NAO's head. It is possible to add external flash memory at NAO's head in the back. In order to communicate with NAO, one can either connect to an Ethernet port placed in the back of NAO's head or to connect through Wi-Fi. NAO is compatible with the IEEE 802.11g Wi-Fi standard and can use both WEP and WPA networks. Through the transceivers in NAO's eyes an infrared connection is possible to communicate with other robots or devices that support infrared. It is possible to give NAO commands through infrared emitters such as remote controls [18]. NAO is powered by a lithium ion 27.6 Wh battery located at the back of its torso. Aldebaran claims that the battery provides autonomy for 60 minutes in active usage and 90 minutes in normal mode, and it takes 5 hours to charge fully.

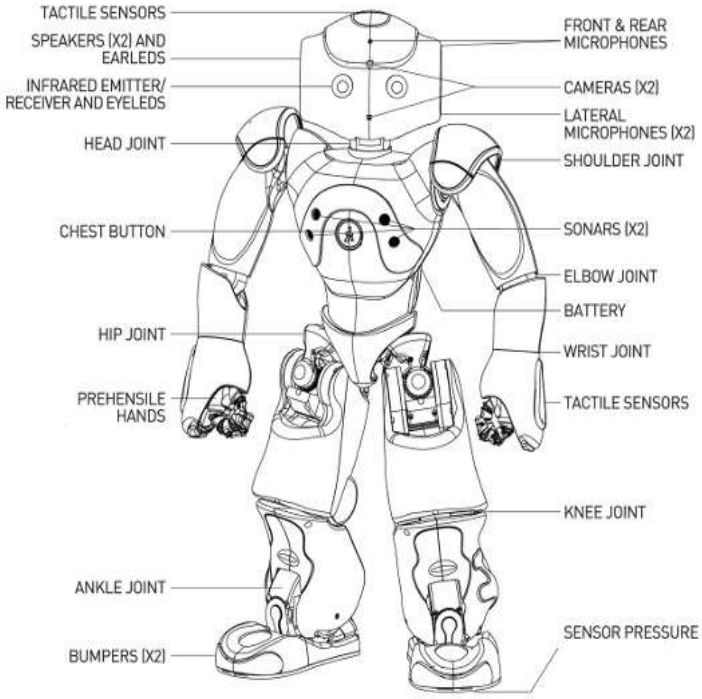

Figure 3.2. NAO sensors and joints

3.2.2 NAO sensors

NAO has a variety of sensors that helps it to gather information about itself and its environment. Figure 3.2 shows roughly where each of the sensors is located. Figure 3.3 categorizes the NAO sensors into proprioceptive and exteroceptive ones. Proprioceptive sensors provide data about the robot itself. NAO uses as proprioceptive sensors an Inertia Measurement Unit (IMU), force sensitive resistors (FSR) and magnetic rotary encoders (MRE). Exteroceptive sensors provide a way for the robot to perceive its environment. Exteroceptive sensors that are used with NAO are: contact and tactical sen-

sors, sonar sensors, cameras and an infrared sensor, and an optional laser scanner which was not available in this work.

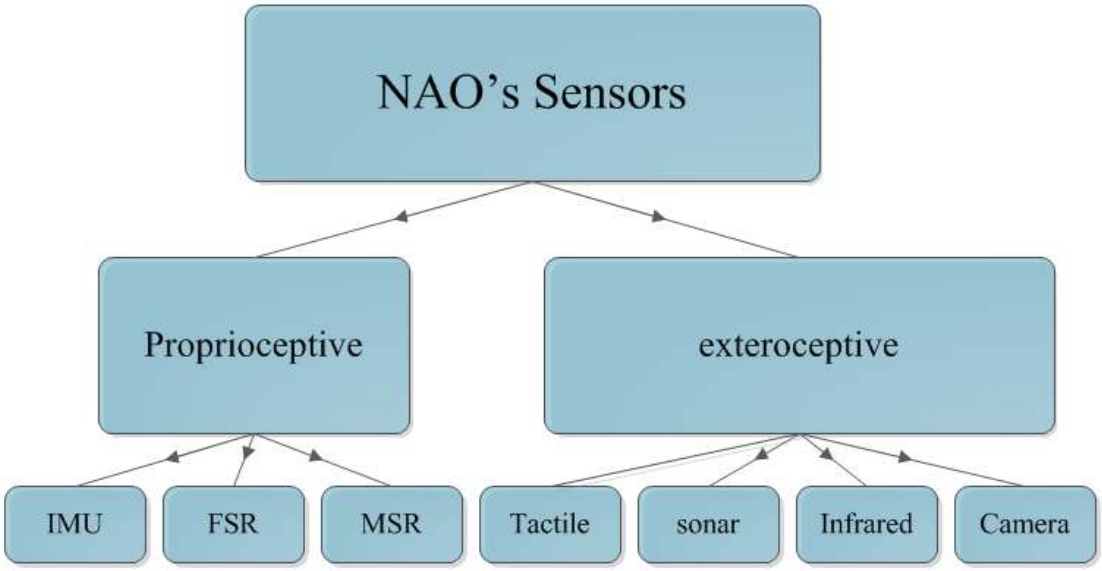

Figure 3.3. Types of sensors in NAO

IMUs use a combination of gyroscope and accelerometer data to estimate the motion of mobile robots. In NAO the IMU is also used to get the robot posture and stability of the robot during its movements. NAO IMU is equipped with a 3-axis accelerometer and two 1-axis gyros located in the torso. The problem of using only IMU based motion estimation is that it is subject to uncertainty and after awhile its error increases dramatically.

Force Sensitive Resistors (FSR) change their resistance according to the force applied to them. There are 8 FSRs in NAO. Each foot of NAO has 4 FSRs located under its sole. The analysis of robot stability applies data from these sensors. Figure 3.4 shows the position of the FSRs.

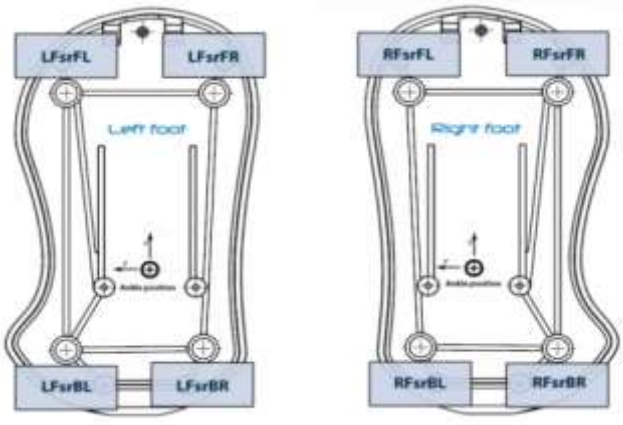

Figure 3.4. Locations of force sensitive resistors

Magnetic Rotary Encoder (MRE) uses the change of magnetic field to determine the rotational motion of a shaft or a motor. NAO is armed with 36 MREs, which provide information on all the joints of the robot

Tactile sensors provide information about the physical interaction of the robot with the environment. They correspond to the haptic sense of humans. NAO has three tactile sensors on its head and three on both wrists. There is also a push button on its chest and a bumper in front of each foot. The purpose of these sensors is to detect object collision while walking or to trigger commands to the robot.

Sonar sensors are regularly used in mobile robots. Sonar sensors send a signal and receive the reflection of that signal from objects in robot environment. Using time of flight (TOF), the sensor computes the distance of an object from the robot. Although sonar sensors are quite popular in robotics applications due to their low cost, they have some limitations, such as not receiving reflection from some surfaces, and receiving multiple reflection and interference of the reflections. Furthermore, these sensors are rather uncertain compared to laser scanners. NAO robot has 2 sonar sensors which able it to measure distances to obstacle approximately in the range of 0.25 to 2.5 meters in a 60 degree conic field.

Vision systems play a very important role in robotic sensing. Many algorithms have been developed to utilize cameras as distance sensors and/or object detectors. NAO robot is equipped with 2 video cameras in its head, one in the robot's forehead to view straight in front of the robot and another one located at its mouth in order to view the ground in front of the robot. In NAO H25, which is used for our project, video cameras provide up to 1280x960 resolution at 30 frames per second (fps). Experiments with NAO in this work prove that the field of vision is as specified by Aldebaran Company. Figure 3.5 shows the field of view characteristics of the cameras.

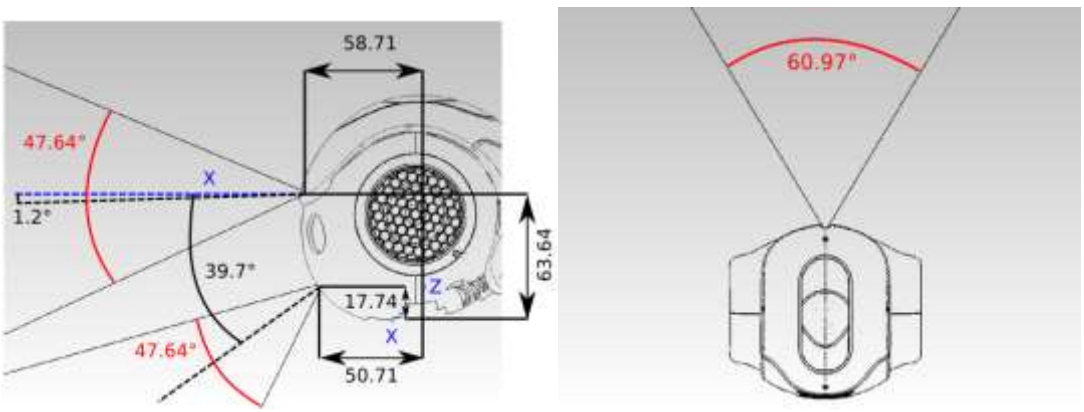

Figure 3.5. NAO's camera's field of view

As can be seen from this Figure, there is no considerable overlap between the two video cameras' fields of view. Therefore the cameras do not provide stereo vision. This project utilized NAO straight-in-front camera as a monocular vision system.

3.2.3 Mechanical structure

NAO has 25 degrees of freedom (DOF). Thus, there are 25 electric motors in its joints to generate the movements of the robot. 11 degrees of freedom belongs to lower part of the robot and the rest belongs to upper part. The table 3.2 shows the distribution of DOFs in NAO robot.

Table 3.2: NAO degrees of freedom (DOF)

Total degree of freedom	
Head	2 DOF
Arm	5 DOF each
Leg	5 DOF each
Hand	1 DOF
Pelvis	1 DOF

In order to do any physical action with NAO robot, one must turn on the stiffness of the corresponding joints. This will able the joint motor to move parts of NAO's body. One should be careful not to keep joints locked for a too long period of time. This will increase the temperature at joints and may even damage the joints.

3.3 NAO's Software

This Section describes the software architecture and the tools available to program NAO. The Section presents the main software components of NAO, different ways to program NAO robot, and other software, which comes with the robot. Software developed for NAO provides both for novice and experts the possibility of programming NAO. Visual interfaces such as Choregraphe can help anyone to program NAO while NAO SDK packages provided for experts enable them to program NAO through different computer languages.

The software for NAO can be divided into two categories: embedded software and desktop software, which allows a remote computer control of NAO. The main software which is running on NAO robot is called NAOqi. NAOqi runs on a robot operating system called openNAO. Any desktop software should eventually connect to NAOqi to execute a program. Figure 3.6 shows how remote software and NAOqi are connected.

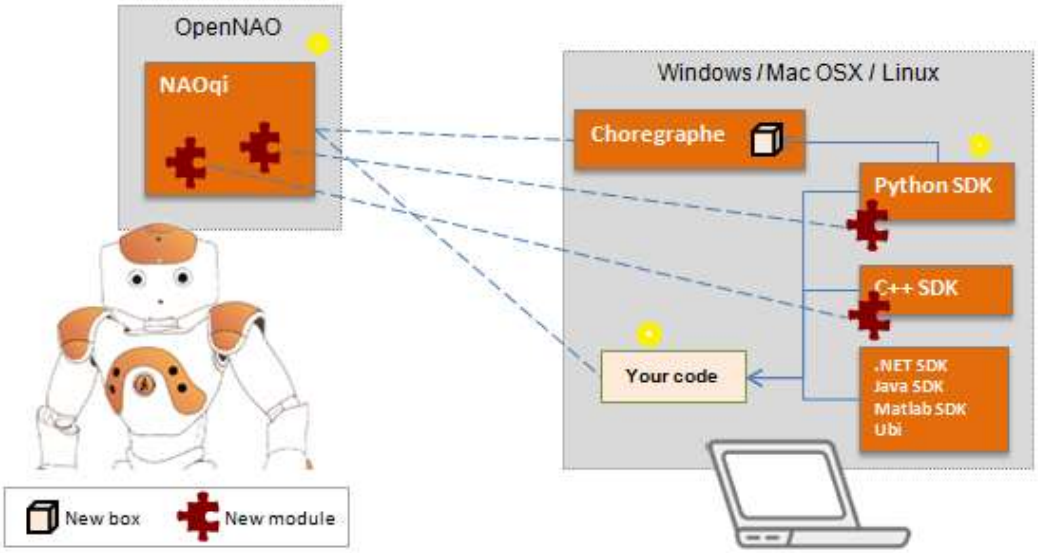

Figure 3.6. NAO's software interaction

Boxes marked with yellow dots are those which have been used in this thesis. In the following, these components will be discussed in more details.

3.3.1 NAOqi Framework

NAOqi can be considered as the brain of NAO as it is responsible for executing actions of any sort. NAOqi framework is cross-platform, cross-language and introspection. Cross-platform means that NAOqi is independent of the platform it is running on. Therefore, it can be run on any operating system such as Linux, Windows and Mac. The ability to develop modules in Python or C++ and use them anywhere needed is called the cross-language property. Introspection means that NAOqi knows where to look for the functions that are needed.

NAOqi is a collection of modules that encapsulates methods for motion, vision, audio to control the robot, and to acquire important data from NAO robot.

When NAOqi executes a program, it loads libraries which encapsulate all modules and methods. NAOqi works as a broker so that any method can be accessed from other modules or across the network. Figure 3.7 shows the tree structure of modules in NAOqi.

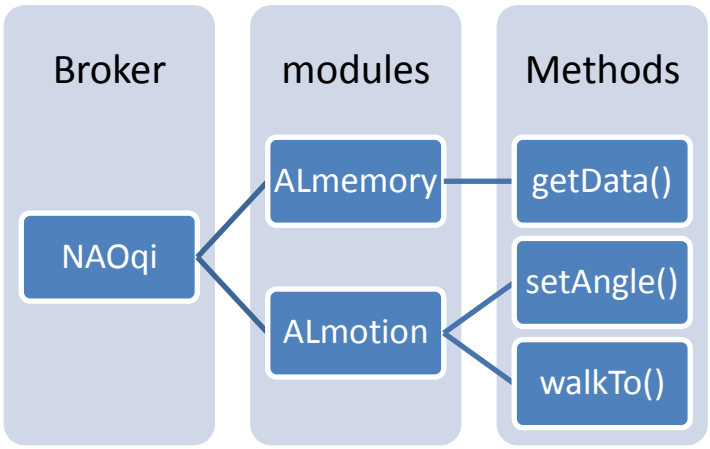

Figure 3.7. NAOqi structure illustration

The broker structure of NAOqi and the modules not only allow the access to the methods but also provides the service to look up for the modules and methods from outside the process.

3.3.2 Choregraphe

Aldebaran provides a desktop software for beginners to program NAO robot. Choregraphe is a graphical user-friendly environment that allows many methods to be used through a simple drag and drop of function boxes. In order to use Choregraphe it is not necessary to have a robot. One can install NAOqi on a desktop computer and Choregraphe can be used with the NAOqi running on the same computer. Figure 3.8 shows the Choregraphe environment.

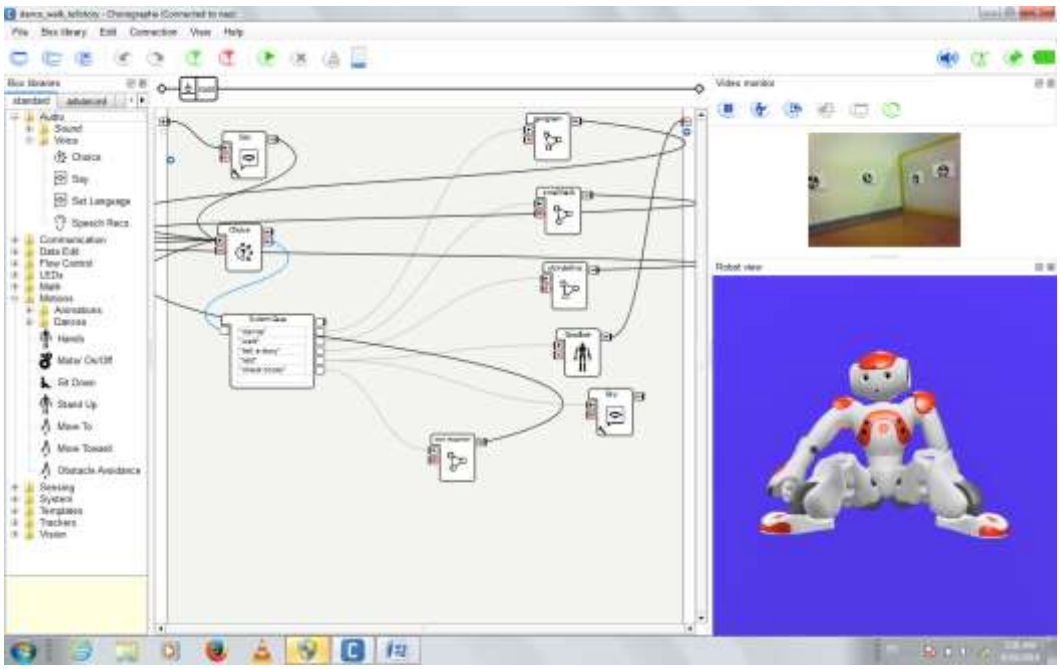

Figure 3.8. Choregraphe environment

Choregraphe interface has three main sections, which are the box libraries, the flow diagram and the robot view. Box libraries allow the use of predefined methods such as sit down, stand up, and dance, which are fairly complicated to develop from scratch. It also provides easy access to methods in vision, audio and motion. These blocks are written in Python. Further boxes can be implemented with Python. Diagram space is the main window of Choregraphe and allows connection of boxes to flow digrams and execution of behaviors. Robot view window shows a 3D model of the robot using the information coming from robot sensors. Further windows can be added to interface as needed.

3.3.3 Monitor and Webots

Monitor and Webots are another desktop software which can be used by NAO robot. Monitor software is installed during Choregraphe installation and it provides access to data acquired from NAO sensors and vision system. It can be connected to robot memory to query the data values of the sensors. It can be also connected to NAO camera to investigate vision information. Figure 3.9 shows the Monitor interface.

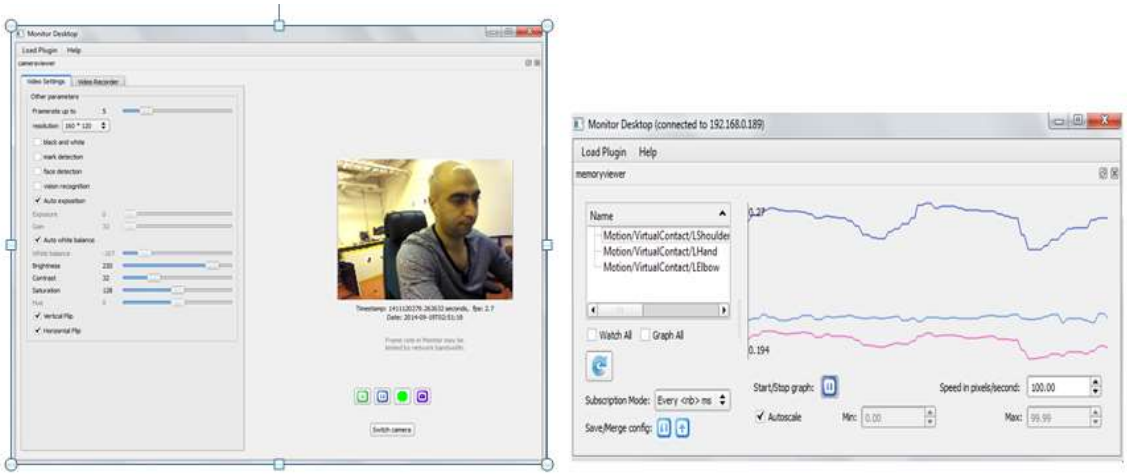

Figure 3.9. Monitor software interface

Webots is a simulator for robots, developed by Cyberbotics. Cyberbotics provided in cooperation with Aldebaran Company a version specific to simulate NAO in a virtual environment. Webots can be linked easily to Choregraphe and Monitor which makes it an interesting environment to experiment before applying to a real robot. Figure 3.10 shows Webots interface.

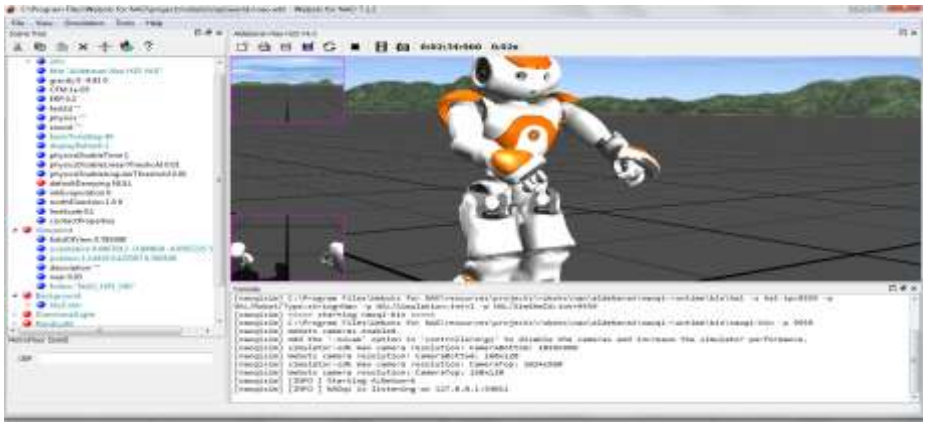

Figure 3.10. Webots interface

3.3.4 NAO programing

NAO robot is a fully programmable platform. NAO supports C++ and Python as languages which can be used directly on the robot. However, other languages such as MATLAB, Java, .NET, Urbi can be used to program NAO through the SDK packages provided.

In this project, Python has been selected as the programming language. Python is well supported by Aldebaran and the available API in Python makes it relatively easy to work with. In this section a brief overview of NAO programing in Python will be presented to provide an idea of how NAO can be programmed in practice.

In order to be able to program NAO, one should have a comprehensive understanding of NAOqi. As it was explained the basic structure of NAOqi consists of brokers. In order to apply a broker, an object of represented module is created with a proxy, so that the methods of a specific module will be available. Figure 3.11 shows a simple case for text to speech module which allows NAO to talk.

```
1. from naoqi import ALProxy
2. speaker    = ALProxy("ALTextToSpeech", "nao.local", 9559)
3. speaker.say('Hello Mojtaba, I am  NAO')
```

Figure 3.11. A simple python code using NAOqi package

This is an example of remote programing of NAO. To create a proxy one needs to define an IP address and a port to connect to NAOqi broker and to use a module (in this case “ALTextToSpeech” module). Furthermore, to be able to use NAOqi remotely, one must import it as a library as given in the first line in Figure 3.11. Aldebaran provided an online documentation for Modules and Python sample codes for some of these Modules. The most important modules and methods for this project are described in more details as follows.

NAO robot motion: ALMotion module includes most of the high level and low level commands to control the robot motion. An important aspect to consider is that one should set the joint stiffness “ON” otherwise the motion command does not have any effect. ALMotion provides high level methods such as Moveinit() which makes the robot to stand with initial standing position and MoveTo() which moves the robot a specific distance. Low level methods for joint control are for example setAngle() and changeAngle(), which set a joint value, respectively changes it.

NAO robot Vision: there is a set of modules for different aspects of the vision of NAO, such as ALFaceDetection, ALPhotoCapture, ALMovementDetection, and ALLandmarkDetection. ALLandmarkDetection is the module which has been used most intensively throughout this work. It covers the area of vision which uses specific markers known by NAO robot.

NAO robot memory: ALMemory modules are a collection of methods related to memory of NAO. It provides the access to the state and values of NAO actuators and sensors. Main function which allows to access to memory data is called GetData ().

4 METHODOLOGY AND IMPLEMENTATION

This Chapter includes the methodology used for this project. It describes the methods for inquiring data from robot vision and for transforming the data into meaningful information for the global localization of the robot. This Chapter covers also a simple planning algorithm for finding the path with which the robot reaches the target.

4.1 NAO vision markers

NAO can use a variety of vision packages through the robot operation systems (ROS) [19] as well as vision packages provided by Aldebaran, such as redBallDetection and faceDetection. In this project the focus was to utilize the landmark module for vision-based localization. This Section describes the main idea of this approach.

4.1.1 NAO markers

Aldebaran provides the NAO robot with specific markers which can be detected in the surrounding environment by NAO vision. A package of 29 different markers, black circles with a white pattern on them, is provided. NAO can get information related to the detection of these markers by using ALLandmarkDetection module. Figure 4.1 shows examples of NAO markers.

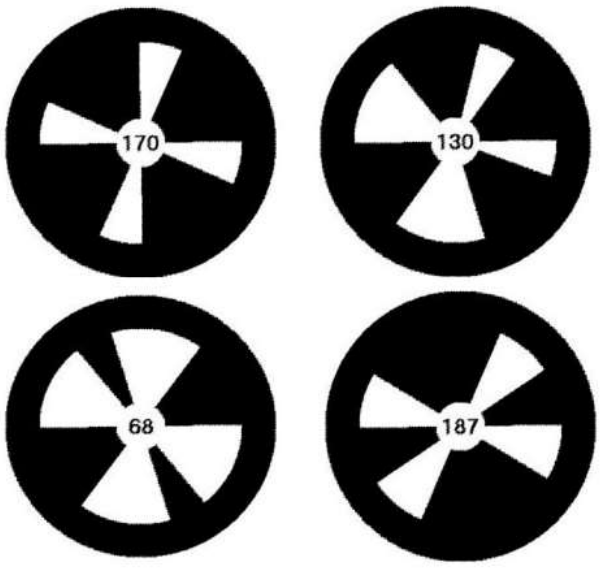

Figure 4.1. Examples of NAO markers

In practice NAO can detect the difference between these markers and associate them to the id number shown on the marker. The numbers on markers are only for human understanding. Using Monitor software with a vision plugin one can investigate the results

of marker detection. The software denotes detections with a circle around the markers. Figure 4.2 shows an example of using Monitor software.

Figure 4.2. Marker detection with monitor software

4.1.2 Landmark Limitations

Although the built-in module for landmark detection provides a promising approach for visual localization, the method suffers from some practical limitations. The first requirement is for illumination. The documentation suggests that the detection is possible under the office lighting, i.e. between 100 Lux and 500 Lux. Experiments show that lighting conditions above or under these thresholds may indeed result in either misclassification of the markers or no detection at all.

Another limitation is related to the size range of the markers in the image. The documentation specifies that the size of a marker detectable to NAO is between 14 and 160 pixels in the QVGA image. This poses a real problem in implementation as the experiments show that with usual size printed markers, NAO can not detect markers at distances larger than 200 cm due to that the size of the marker in pixels is too small.

The third limitation is the tilt between the marker plane and camera plane. In practice NAO is not able to detect markers which are tilted by more than 60 degrees with respect to the robot line of sight.

4.1.3 Landmark data structure

Despite all the limitations the markers are well suited for small indoor environments. To utilize these markers, one should have a clear understanding about the data that built-in methods derive from these markers. Methods, such as GetData (), provide the data about the marker observation. The data derived from NAO memory is a list of lists and its structure is as follows:

[[TimeStampField] [Mark_info_0 , Mark_info_1, . . . , Mark_info_N-1]]

TimeStampField consists of 2 elements which show the time when the marker is detected at Unix time milliseconds and microseconds respectively. The second list is a list of all markers that are detected at the time of imaging. Marker_info element consists of the information about each marker and has the following structure:

[0, alpha, beta, sizeX, sizeY, heading] [MarkID]

Where alpha and beta are the angular vertical and horizontal location of center of marker with respect to image center in radian respectively, and sizeX, sizeY are the angular size of the marker in radian. Heading shows how the marker is orientated with respect to the vertical axis of NAO camera, and the MarkerID gives the id number of the marker. However, experiments show that in reality the values of sizeX and sizeY are always identical and heading value alone is not accurate enough to get the orientation of the marker with respect to NAO's head.

4.1.4 Marker coordinate

In order to get the coordinates of a landmark in the robot frame some further calculations are necessary. This analysis uses alpha, beta and sizeX values gathered by ALLandmarkDetection module. Figure 4.3 shows the geometry of the imaging. Given the real size of markers, one can calculate the distance of the marker to NAO camera using Equation (1) when the marker is close to the line of sight of robot camera:

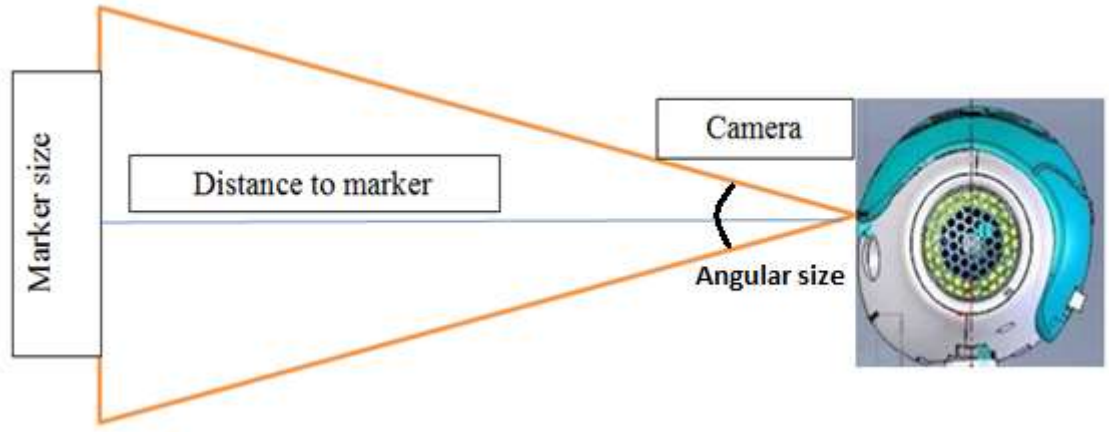

Figure 4.3. Visualization of the triangle created by the marker and the camera

$$D = \frac{(mS)/2}{\tan(a/2)} \tag{1}$$

Where **D**, **a** and **mS** stands for distance to marker, angular size and marker size respectively. Angular size is the size of the marker in radian, which is the sizeX value and

marker size is the known size of printed marker. The alpha and beta value can be used to get the rotational transformation between camera frame and the landmark using a built-in function of NAO ALmath module. To get the coordinate of the marker, one more step is needed, which is converting the coordinate in camera frame to robot frame. This conversion can be done using built-in functions which use the transformation relationship of the currently used camera to the robot frame. Thus, the landmark coordinates in robot frame can be calculated by Equation (2) with built-in functions:

$$\begin{aligned} \text{landmarkToRobot} \\ &= \text{cameraToRobot} \\ &* \text{landmarkToCameraRotationTransform} \\ &* \text{landmarkToCameraTranslationTransform} \end{aligned} \tag{2}$$

The resulting transformation includes the X and Y coordinate of the marker in robot frame. It is shown in the next section how to use this coordinate to localize the robot in the global frame. This equation also provides the Z coordinate of marker but in this work we considered markers as features in a two dimension plane.

4.2 NAO Localization

As was shown in the previous section, it is possible to get for any marker its relative 2D coordinate with respect to robot frame. Localization is based on that the marker coordinates are known in the global frame. However, knowing one marker coordinates in robot frame and in global frame is not sufficient for determining the location of robot in real world. This section describes the approach developed to solve this issue.

Let us first tackle the transformation between global frame of the world and local frames of the robot. NAO's frame has its X axis from NAO forward and Y axis direction is to the left of NAO. Figure 4.4 shows NAO's frame and the global frame together.

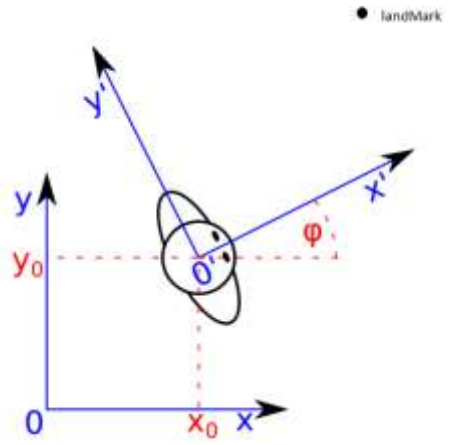

Figure 4.4. NAO frame and global frame

In general a 2D transformation between two frames is a combination of rotation and translation, written as:

$$\begin{pmatrix} X_{global} \\ Y_{global} \end{pmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \begin{pmatrix} X_{robot} \\ Y_{robot} \end{pmatrix} + \begin{pmatrix} X_0 \\ Y_0 \end{pmatrix} \quad (3)$$

Where X_0 and Y_0 are the coordinates of the robot in global frame. The subscript "robot" denotes the coordinates of the marker in the robot frame and the subscript "global" denotes the global coordinates of the marker. The angle φ is the orientation of the robot in the global frame. The same equations can be written in a more compact way as

$$\begin{bmatrix} X_{global} \\ Y_{global} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi & X_0 \\ \sin\varphi & \cos\varphi & Y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{robot} \\ Y_{robot} \\ 1 \end{bmatrix} \quad (4)$$

As can be seen from Equation (4) knowing only the global coordinates and local coordinates of one marker is not enough for solving the location of the robot and its orientation. However, if there is at least two markers with known global locations, the corresponding set of two equations, Eq. (4), can be solved assuming that the pose of the robot has not changed between imaging the markers, or it has changed by a known amount. Figure 4.5 illustrates a case with two markers.

It should be noted that with two markers there are 4 equations and 3 variables to solve. However, defining cosine and sine of orientation as independent variables leads to four equations and four variables. This obviously leads to a constraint for the solution as the squares of cosine and sine must add up to one.

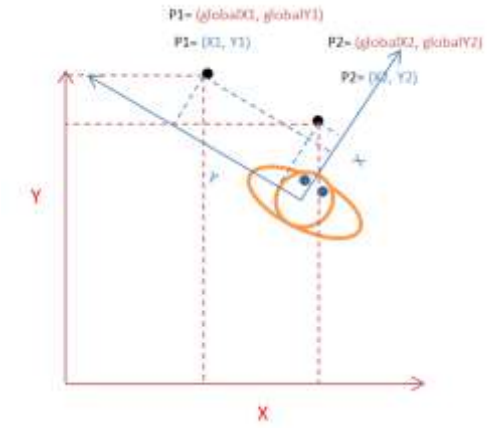

Figure 4. 5. Representation of two marker locations in global and robot frame.

$$\begin{bmatrix} X1_{\text{Global}} \\ Y1_{\text{Global}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{a} & -\mathbf{b} & \mathrm{X}_0 \\ \mathbf{b} & \mathbf{a} & \mathrm{Y}_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{X}_1 \\ \mathrm{Y}_1 \\ 1 \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} X2_{\text{Global}} \\ Y2_{\text{Global}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{a} & -\mathbf{b} & \mathrm{X}_0 \\ \mathbf{b} & \mathbf{a} & \mathrm{Y}_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{X}_2 \\ \mathrm{Y}_2 \\ 1 \end{bmatrix} \tag{6}$$

Solving these four equations, one can find X_0 and Y_0 and therefore localize the robot. The orientation of the robot can be calculated from each of the parameters **a** or **b**. In practice, in order to be able to solve Equations (5-6), the two landmarks selected should have significant distance from each other, otherwise there will be huge errors in robot pose estimation. In our set up, this can be achieved selecting a pair of markers on two different walls of robot environment. The pseudocode 4.1 shows the localization algorithm that also deals with practical problems.

```
1   Begin
2   Get a list of known markers with their relative coordinate in robot frame.
3     For any pair of markers:
4       If they have different X and Y with each other (not same wall markers)
    then
5         Solve coupled equations and get the pose of the robot.
6       Endif
7       If pose is acceptable then
8         Add it to accepted robot poses list
9       Endif
10    Endfor
11  Average over robot pose list
12  End
```

Pseudo code 4.1: Localization algorithm

As it can be seen in line 4 the algorithm checks that the selected markers do not have similar X or Y coordinates. In other word, the markers on the same walls in our environment will be discarded. Furthermore the algorithm checks in line 8 for the possibility of the pose given the condition of robot environment. This prevents of averaging over incorrect poses produced as a result of wrong marker detection.

The implementation must deal with how the dissimilarity of the marker positions is analyzed (line 4), how the nonlinear equations Eqs (5-6) are solved (line 5) and how the constraint between variables a and b is handled (line 8).

A very important part of the localization is in solving the Equations (5-6) for two markers of known global coordinates. Unfortunately, this equation cannot be solved directly with core python libraries. The approach used here is to solve it with a package external to python, called **sympy** that was downloaded and installed and then imported into the code.

Sympy is a library for symbolic mathematics and written entirely in python and does not need any further external python library. Sympy makes it possible to solve equations in Matlab fashion. This means that one can define variables as symbols and solve equations with respect to these symbols. Figure 4.6 shows an example of sympy code.

```
>>> from sympy.solvers import solve
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> solve(x**2 - 1, x)
[-1, 1]
```

Figure 4.6. A simple python code using sympy library

After solving these equations, it remains to extract the orientation of the robot. Obviously, two options exist. One can either use the cosine or the sine solved.

$$\varphi = \sin^{-1} b \tag{7}$$

$$\text{or } \varphi = \cos^{-1} a \tag{8}$$

Experiments show that using cosine usually provides better estimation of the orientation. Another point should be considered is that the sine-based orientation is in the range of $-\pi/2 \leq \varphi \leq \pi/2$ and the cosine-based in $0 \leq \varphi \leq \pi$. A 2D planar robot can take any orientation in the range $0 \leq \varphi \leq 2\pi$, Thus straightforward solutions (7-8) are not sufficient to cover all possible orientations. However, using the sign of cosine and sine, one can develop a simple algorithm to assign the right value to the orientation. The algorithm to do so is described in pseudocode 4.2.

```
1   Begin
2    if cosine part is between -0.9 and 0.9 or sine part is between -0.9 and 0.9:
3       if sine part and cosine part are both positive (first quarter):
4         calculated orientation is not changed
5       elseif sine>0 and cosine <0(second quarter) and sine part is used for calculation
6         replace orientation with 180- orientation
7       elseif sine and cosine >0(third quarter) and cosine part is used for calculation
8         replace orientation with negative orientation
9       elseif sine part negative and cosine part positive(fourth quarter)
10        if cosine part used to get orientation
11          change it to negative orientation
12        else
13          replace orientation with 180-orientation
14  end
```

Pseudo code 4.2: Algorithm to get orientation for all quarters of a 2D plane

Another issue in implementation is the uncertainty in measurements. The uncertainty may result in a drastic failure if not dealt with properly. In this work, we corrected cases

which may lead to such failure. This correction is very crucial in particular when calculating the orientation. Two important cases have been considered in this work.

The first issue arises when the robot orientation is close to 0 or π. Then a and b obtained by solving Equation (5-6) may have absolute value larger than 1. Then $\varphi = \cos^{-1} a$ is undefined. To solve this problem we can use the $\varphi = \sin^{-1} b$ equation to get an estimate of orientation. The algorithm is described in pseudocode 4.3.

```
1  Begin
2  if cosine between -0.9 and 0.9:
3        Calculate orientation from arccosine
4  else
5       Calculate orientation from arcsine
6  endif
7  end
```

Pseudo code 4.3: Orientation using sine and cosine

The second issue is related to averaging over the orientations in specific areas of planar orientation. As the planar orientation is described as $0 \leq \varphi \leq \pi$ or $-\pi \leq \varphi \leq 0$, for orientation near π there might be cases where it can be calculated with negative value while most values are positive and averaging over all values will cause for negative value to cancel out some positive value and produce incorrect orientation. Therefore, these types of estimated orientations must be preprocessed. The preprocessing procedure is shown in pseudo code 4.4.

```
1  Begin
2  if the average of all heading>0
3    add 360 degree to all orientation less than -170 degree
4  endif
5  if the average of all heading<0
6    subtract 360 from all orientation more than 170 degree
7  endif
8  end
```

Pseudo code 4.4: Correct orientation values with the wrong sign

4.3 Mapping environment features with Nao

Mapping is another major area of interest in most robotic applications. In this project, NAO robot has been used to create a feature-based map of the environment. A feature can be any distinct property of the environment and for our purpose we decided to use NAO landmarks as features.

In this project the problem of mapping has been solved after the robot has localized itself using known marker. On the next step, any marker of unknown location with its known relative coordinates in robot frame can be mapped to global frame using Equation (4).

In this case X_0,Y_0,φ are the known pose of the robot and X_{robot} and Y_{robot} are the measured relative coordinates of the marker to be mapped in the robot frame. Thus, global marker coordinates are obtained.

4.4 Planning

This section deals with robot movement to reach a target. This is usually called go-to-goal behavior in planning. The idea for a go-to-goal planner is to plan the path to the target from robot's present location and to control the robot movement along the path. As obstacles were neglected in this thesis, the path is a straight line with a direction and length. Figure 4.7 shows a robot and a target in 2D plane.

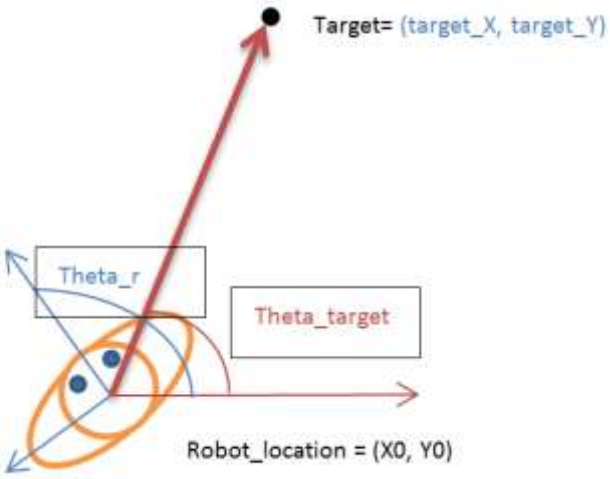

Figure 4.6. Illustration of a robot with a defined target

The Euclidean distance between the target and the robot is:

$$D = \sqrt{(X_{target} - X_0)^2 + (Y_{target} - Y_0)^2} \tag{9}$$

The direction to the target is:

$$\theta = \tan\left(\frac{target_y - Y_0}{target_x - X_0}\right) \tag{10}$$

Based on the orientation of the robot and the direction to the target with respect to global frame, one can find the equivalent turning angle for the robot to align the robot in the target direction. The shortest turning angle of the robot can be calculated with pseudo code 4.5.

```
1  Begin
2  assign target angle - pose angle to turning angle
3  if turning angle > 180
4    reduce turning angle by 360 degree
5  endif
6  if turning angle <-180
7    increase turning angle by 360 degree
8  endif
9  End
```

Pseudo code 4.5: Shortest turning angle for go-to-goal behavior

Before the robot moves in a direction, it checks with its sonar sensors that there is enough space in front of the robot to execute movement toward the target. Thus collisions to walls are avoided.

The ultimate goal of this project was to make a feature-based map of environment while reaching a specified target. Such a task is a combination of mapping, localization and planning. The separate approaches for all these areas has been explained above. A heuristic combination of them to perform the task is shown as a flow chart presented in Figure 4.8.

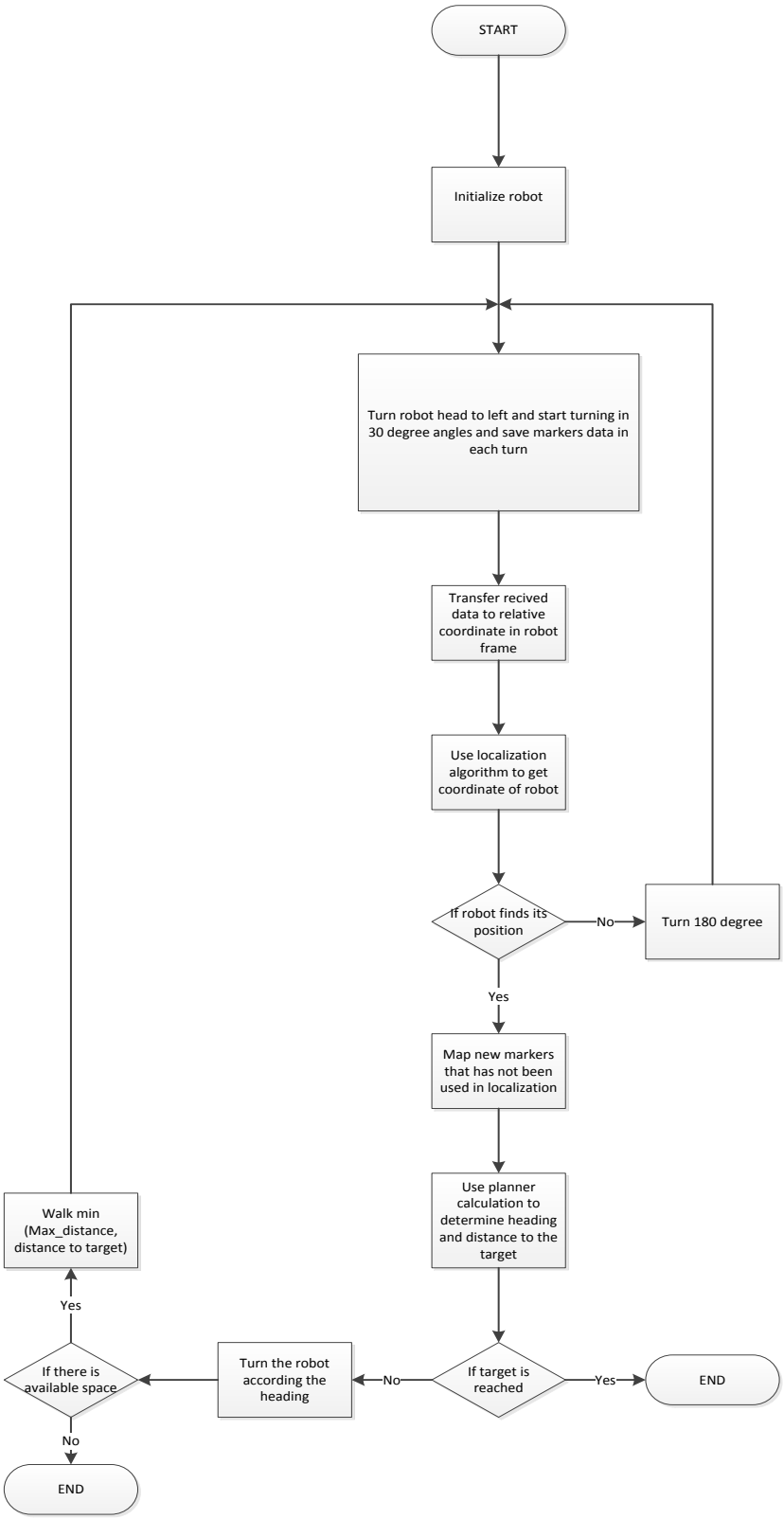

Figure 4.7. Flow chart of localization and mapping with go-to-goal behavior in the absence of obstacles

Thus, the robot starts by turning its head to angles between x and y with respect to torso in 30 degree steps and gets the marker data at each step. After finishing the observations, it calculates the relative coordinates of markers in the robot frame. Using the localization algorithm, it tries to find its location based on markers with known locations.

In the case the robot fails to locate itself, it will turn backwards to get a better field of view and thus enough data. Markers whose locations are unknown are mapped based on the estimated location of the robot so that in the next step the distance between these targets and the robot is available for localization. As long as the robot is not close enough to the target it turns towards the target and moves either a given maximum distance or the estimated distance to the target, depending which one is smaller. The maximum of a single walk is set because Nao's walk is uncertain and localization is done only between the walks. A practical value for maximum walk is 50 cm. A check for available space is also implemented using sonar sensors to prevent collision with the walls.

The implementation in Python differs slightly from this flow chart. The idea in this implementation was to separate actions from the perception of the robot. Therefore, four classes have been developed. These four classes are named as main class, robot class, world class and planner class. There is a module for reducing computational complexity of the main class. The main class is responsible for initiation of the robot by creating objects of the robot and the world. Robot class deals with all the motion actions of the robot, such as turn and move. The world class covers perceptive processes such as mapping new markers, and localization of the robot. Planner class provides the go to goal behavior results, such as desired heading and distance to the target. The diagram 4.9 shows the Python structure used to implement the task.

Such structure will allow the implementation of motion control, path planning and perception algorithms separately for future developments in each of these areas.

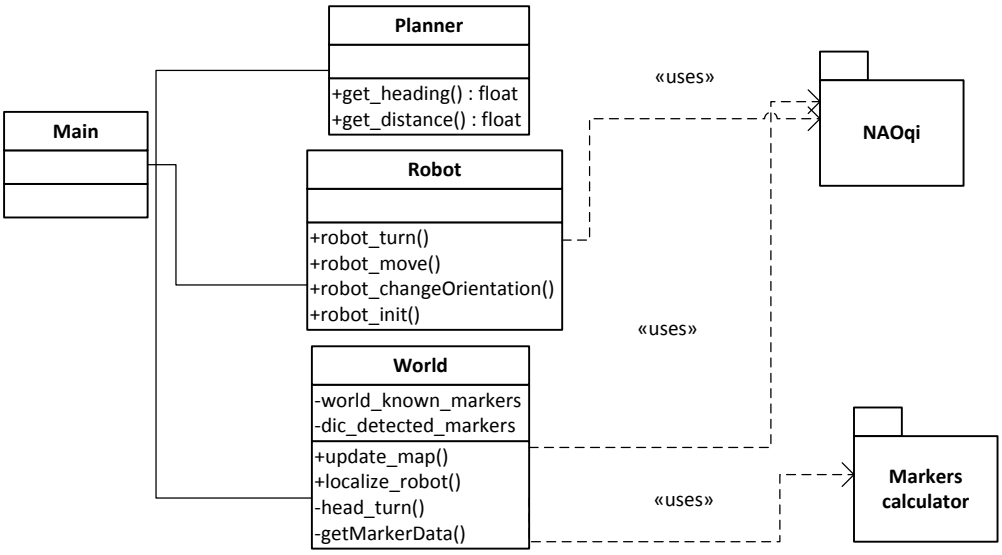

Figure 4.8. Python structure of whole program

Not all the functions of each class are shown in this Figure. World class applies a package of methods called markers calculator in its implementation. Most of the algorithms explained in localization and mapping have been defined as methods of this package. Both Robot class and World class use the NAOqi package which allows access to built-in methods for movement of the robot and detection of markers. The Python code used for implementation of these classes is provided in Appendix 1.

5 RESULTS AND EXPERIMENTS

This section describes the results of various NAO experiments and results related to localization and mapping. Furthermore, the result of such tasks in a go to goal behavior will be discussed.

5.1 NAO movements

Before going into details in NAO experiments, the robot motion has been discussed first. NAO, as any other robots, suffers from uncertainty in its motion. It can be seen from the experiments that although Aldebaran has tried to improve NAO motion and speed, there are still significant flaws. Inaccuracy in following a straight line or turning by a given angle affects the go-to-goal task significantly. Table 5.1 shows the result of an experiment in which NAO was supposed to follow a straight line. This table shows the deviation from X direction walk for 1 meter and 3 meters walk.

Table 5.1: Straight walk experiments

Deviation in 3 meters walk		Deviation in 1 meter walk	
error in direction of walk in centimeters	error in direction perpendicular to walk in centimeters	error in direction of walk in centimeters	error in direction perpendicular to walk in centimeters
-10	+90	2	+2
-6	+92	0	+5
-12	+86	0	+16
-14	+82	0	+14
----	Robot turned left +infinitiy	0	+3

As it can be seen from the table, NAO has a tendency to shift to its left side. This will be a problem in long distances. Due to such a drift, we selected smaller maximal distances in designing a go-to-goal behavior. Turning in place also is not accurate. Table 5.2 shows the difference between the desired angle and result of the experiment.

Table 5.2: turning experiment results

Desired angle in degree	Measured result in degree
45	47
90	108.5
135	157
180	209

As the table demonstrates, NAO has a tendency to turn more than desired. The error introduced to the result is the larger, the larger the requested angle.

5.2 Environment set up of the experiment

In order to perform the experiments a specific environment has been set up. Using vertical walls, the environment was limited to a rectangle with approximately 180cm width and 230cm length. Figure 5.1 shows this environment.

Figure 5.1. Experiment environment

As it can be seen from figure 5.1, NAO markers are located on the walls of the environment. In total there are 13 unique NAO markers on the walls of which 7 markers have a known location and the rest are subject to feature-base mapping. Localization and mapping are not sensitive to the height of these markers. The reason for keeping these markers in approximately the height of NAO is to be sure that it can observe them while scanning with its head.

5.3 Localization and mapping experiments

As the first set of experiments, we study localizing the robot using the algorithm described above. First, the robot is standing at a point trying to get an approximation of its location. Next, the robot tries to map unknown markers to the environment. Based on the experiments the localization accuracy has been estimated. In the first experiment robot faced the right-side of the environment. Figure 5.2 shows the result of this experiment.

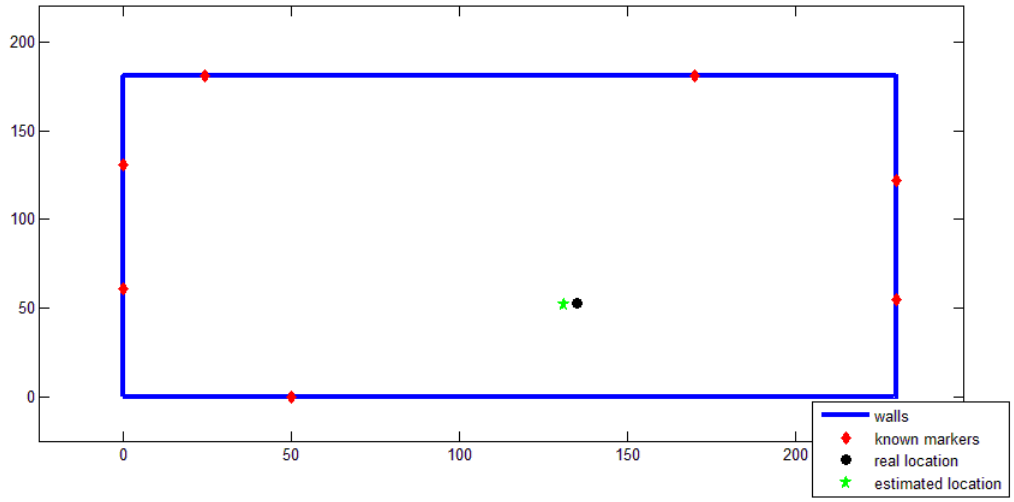

Figure 5.2. Localization with focus on right-side markers (pose 1)

As it can be seen from the experiment there is some error in the estimated location of the robot. The real location of the robot is at (135, 55, <60) while the estimated location is calculated as (131, 52, <55). The amount of error introduced in this experiment is within the acceptable range of the intended application. However, it introduces more uncertainty in the mapping part which is performed afterward. Figure 5.3 shows the new markers mapped to the environment.

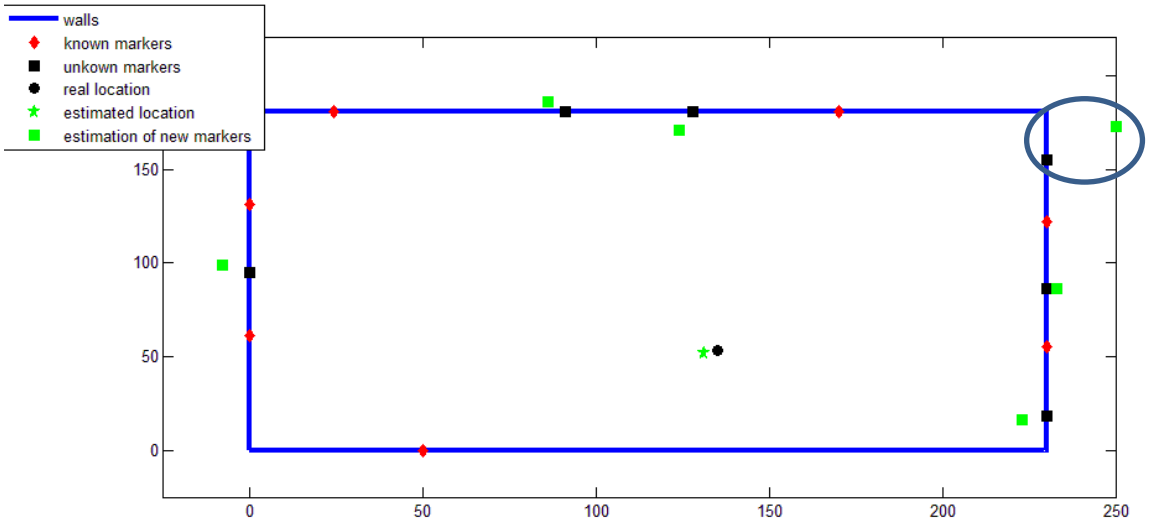

Figure 5.3. Mapping with the focus on right-side markers (pose 1)

The figure shows the deviation in coordinates between real marker and estimated one. Except one, the markers have been mapped with less that 15 centimeter error from the real markers. The wildly deviating marker location is in fact a result of marker detection error. The robot vision detects marker number 107 while the real marker is number 146. This can be the result of the marker plane tilt, which is one of the limitations of the marker detection method.

Similar experiment has been performed with another pose of robot. In this experiment the real location is at (92, 90, <6). Figure 5.4 shows the result of this experiment.

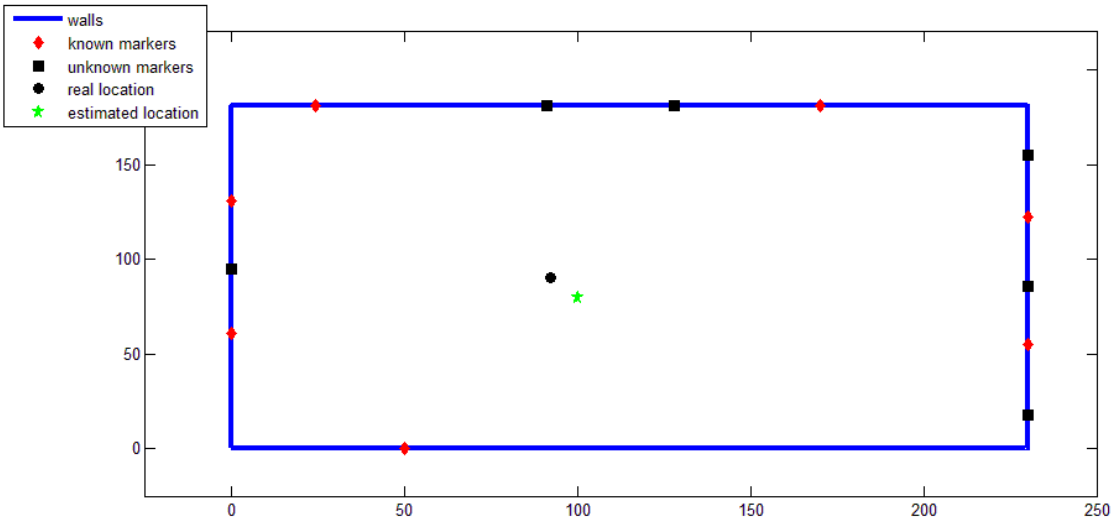

Figure 5.4. Localization with focus on right-side marker (pose 2)

The result shows calculated robot pose at (100, 80, <12). Again, the deviation seems to be acceptably low. One can see the effect of mapping using these values in figure 5.5.

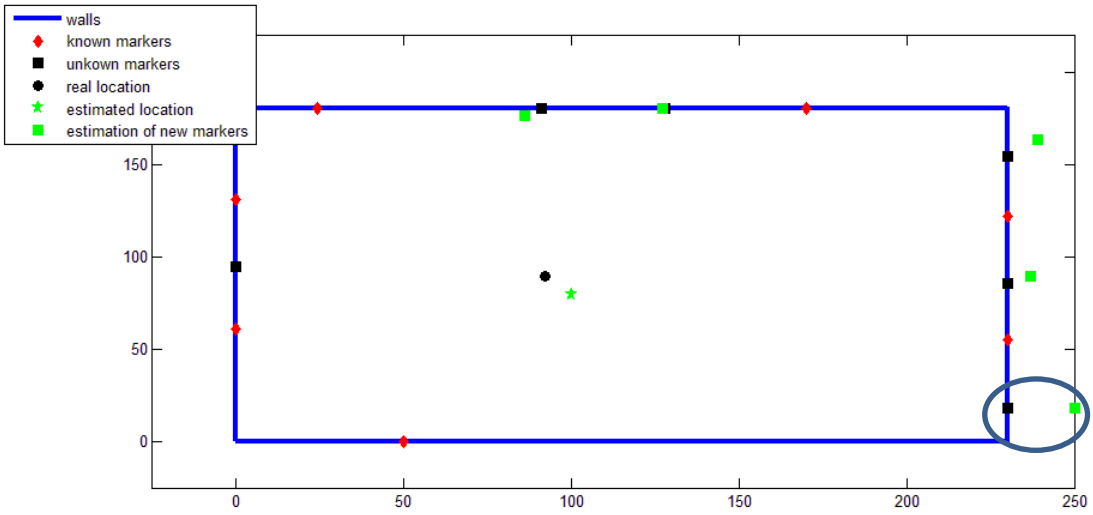

Figure 5.5. Mapping with focus on right-side markers (pose 2)

The figure shows that except one marker the rest of markers seem to be in an acceptable range for such an application. In the other two experiments shown in here the robot is facing the left-side of the environment.

In this experiment, the real pose of the robot is at (92, 120, <133). Figure 5.6 shows the result for localization.

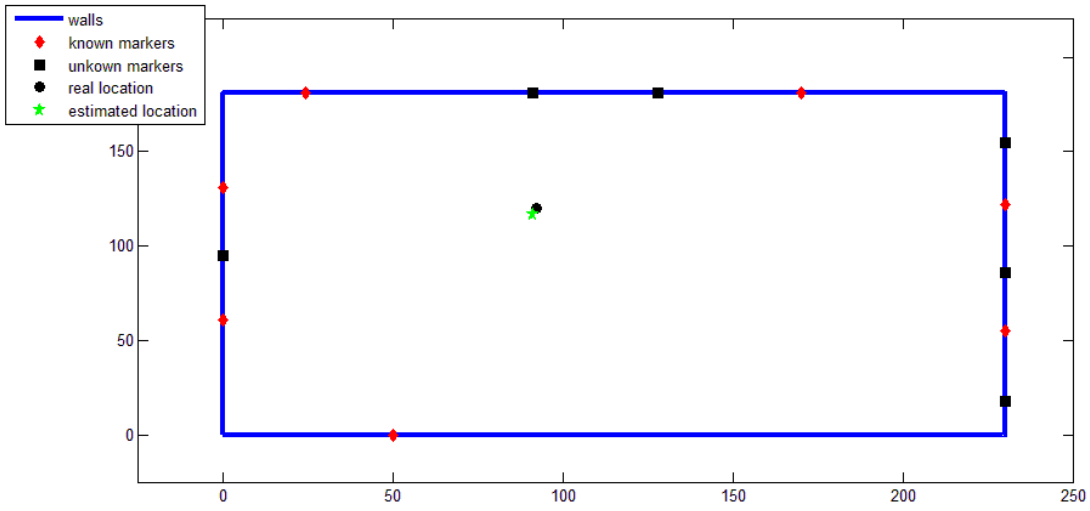

Figure 5.6. Localization with focus on left-side markers (pose 1)

The result shows calculated robot pose at (91, 117, <135). This is a much better estimation of robot location than the two previous experiments. Figure 5.7 shows the mapped markers for this location.

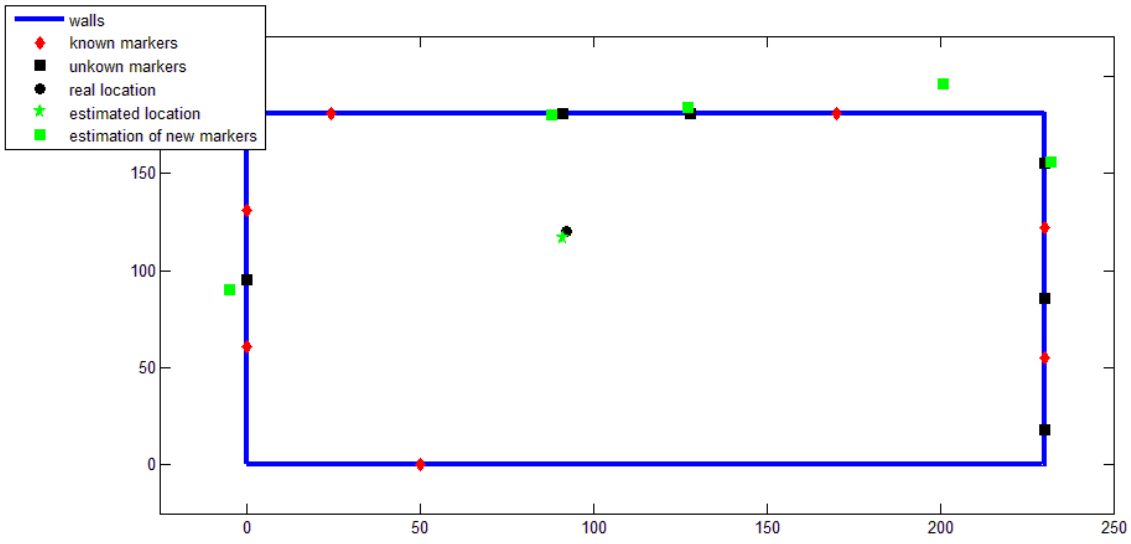

Figure 5.7. Mapping with focus on left-side markers (pose1)

Due to less uncertainty introduced in localization phase significant better mapping can be seen in the figure. . However, there is an outlier point while mapping which is the result of misidentification of the known marker (number 108) as a new unknown marker (number 110).

In the last experiment the robot was facing the left-side but with a different pose. The real location of the robot is at (62, 60, <-133). Figure 5.8 shows the estimated location for such a set-up.

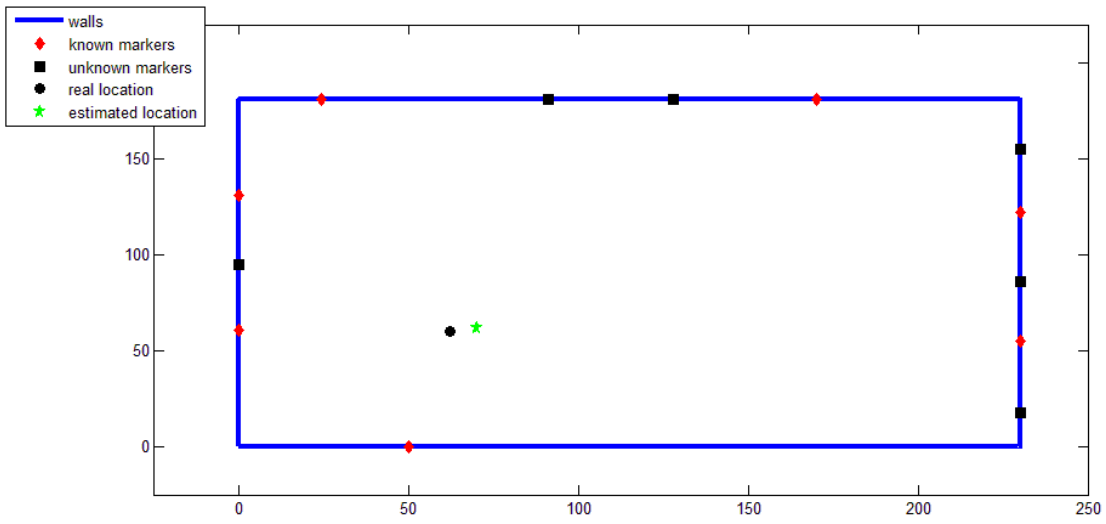

Figure 5.8. Localization with focus on left-side markers (pose 2)

Location estimated in this set-up is at (70, 62, <-128). The effect of such deviation can be seen in the mapping phase. Figure 5.9 shows the resulting map.

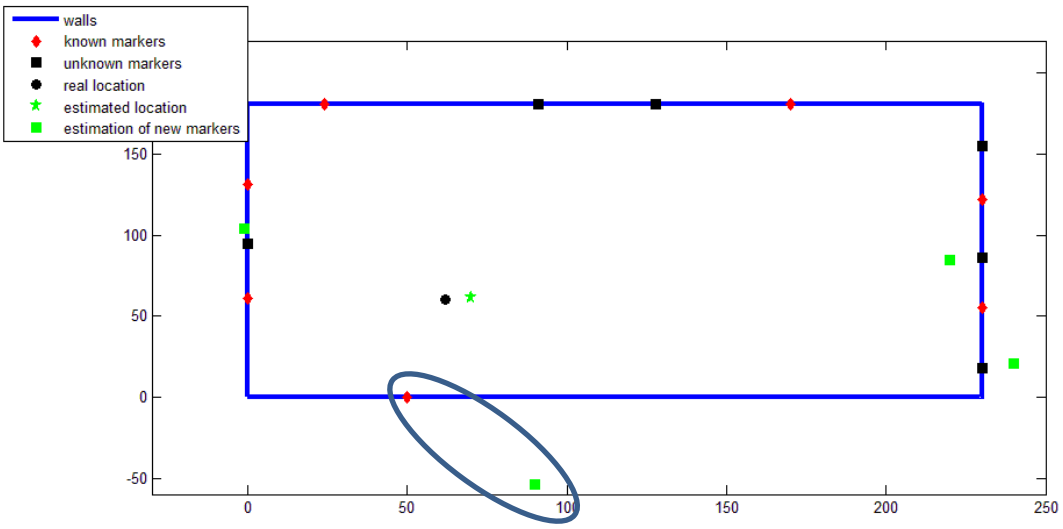

Figure 5.9. Mapping with focus on left-side markers (pose 2)

New mapped markers still have some deviation from the original place. However, there is another outlier marker mapped to the world. This can be a result of misidentification of a known marker as an unknown marker and thus the wrong coordinates introduced for such a new marker.

5.4 Map building while robot moves

While a stationary mapping might be enough in some applications, the reality of robotics problems frequently deals with robot's motions in the environment. A set of experiments were performed to see the effect of motion and how it can affect mapping of the environment.

A target is defined for robot and the task for the robot is that to move toward the target and map new markers while localizing in the process. The target point is given as (170 100) and initial location of the robot is at (62 60).

The first step is a stationary localization. New mapped markers are as follows:

$$[112: (233{,}91), 170: (232\ 28), 175: (-4{,}101)]$$

The result of this mapping is shown in Figure 5.10.

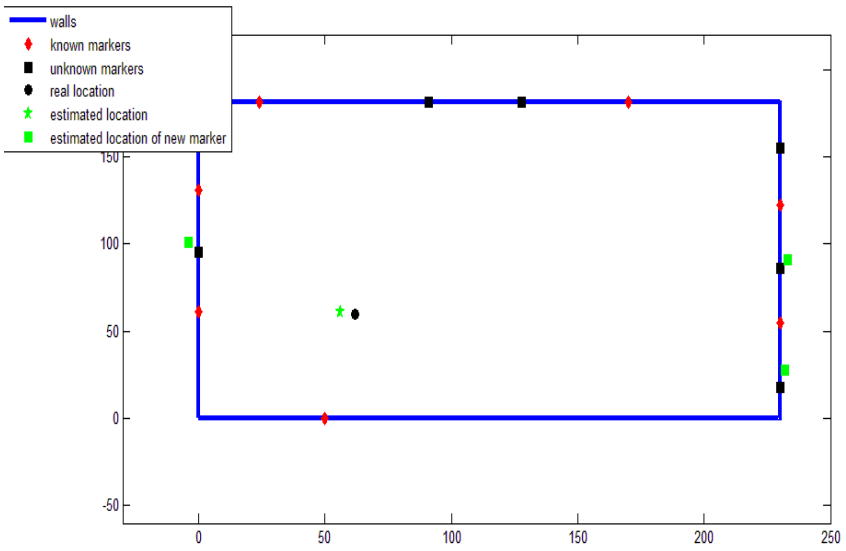

Figure 5.10. Result of localization and mapping after first step

The robot now turns towards the target, walks 50 cms, localizes itself with the known markers and remaps the unknown markers. This resulted in new marker locations as follows:

$$[112: (226{,}80), 170: (233\ 10), 175: (2{,}86), 109: (88{,}174),$$

$$146: (228{,}146), 114: (128{,}178)]$$

The locations of the three markers present on both maps are simply averaged:

$$[112: (229.5{,}85.5), 170: (232.5\ 19), 175: (-1{,}88.5), 109: (88{,}174)$$

$$, 146: (228{,}146), 114: (128{,}178)]$$

Figure 5.11 shows the change in mapping using the average of the two measurements.

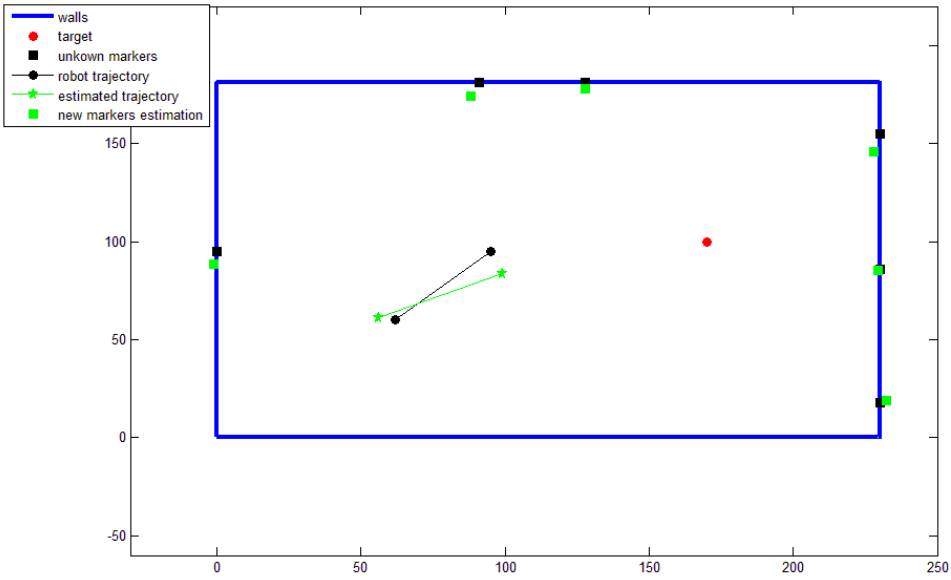

Figure 5.11. Localization and mapping result after second step

It can be concluded that in this case a simple averaging strategy improved all the new marker locations. On the next step robot moves again and reaches to the defined target vicinity. The new mapped marker locations are the following:

$$[112:(231{,}91), 170:(246, 10), 114:(108{,}178), 109:(66{,}176), 146:(221{,}161)]$$

As a result of averaging over the previous step the final mapped markers are located at following:

$$[112:(230{,}87.3), 170:(237{,}16\ ), 114:(118{,}178), 109:(77{,}175),$$

$$146:(224.5{,}153.5), 175:(-1{,}88.5)]$$

Visualization of final result is shown as Figure 5.12.

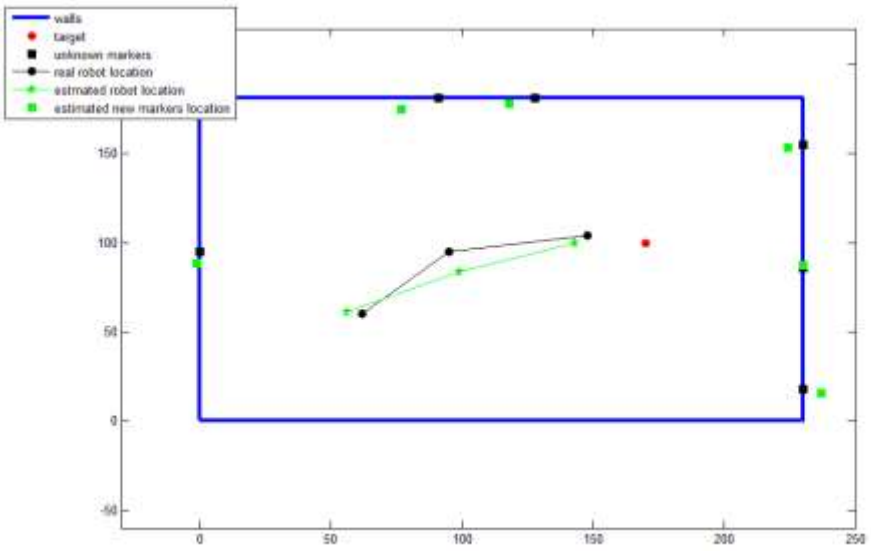

Figure 5.12. Localization and mapping after the last step

Clearly the estimated location in the last step added more deviation from reality. This can be easily explained as the result of accumulated error in estimation of each step since we simply average over all previous locations of new markers. However, after three mappings the estimated locations have a small deviation compared to the size of the environment.

6 CONCLUSION AND FUTURE WORKS

This thesis has focused on developing a localization method using the NAO vision system and easily detectable features, the NAO markers, and to create a feature-based map using this information. Localization and mapping the environment have been the focus of much research in past decades. The current work has focused on simple models that do not require any motion models.

The results of the experiments show that localization and mapping techniques in this work are accurate enough to be used as a starting point for further studies. A promising continuation of this work could be developing active sensing which was discussed in the theoretical background of this thesis. Providing some information about the available map, NAO can turn its attention to the areas with more information to localize itself. At the moment NAO scans the entire field of vision each time to localize itself. This may not be the most efficient way since some areas do not add information useful in localization. In active sensing NAO would try to view the most promising directions and save time at each localization step.

Another interesting approach could be to use the result of this work to find a minimum set of actions in order to reach a target. At the moment, NAO is walking at most 50 centimeters at each step. This value was selected as an example of walk length with an acceptable error. However, one can try to minimize time to reach a target using different walk lengths considering the error introduced in the system. The shortest-time-to-target problem can be seen as a tradeoff between two factors: covering distances quickly and maintaining accurate localization. Obviously, such error will increase the time needed to reach target.

Last but not least, the result of this work can provide a platform for implementing more elaborate path planning algorithms. As an example, the environment could be modified in such a way that it includes obstacles for the robot to avoid. Different path planning methods, such as **A* star** algorithm or equivalent approaches can be investigated and compared. Furthermore, other feature detection methods can be integrated to marker detection to reach more acurrate localization results for more complicated environments.

In conclusion, using NAO humanoid robot, this thesis offers great opportunities to work on autonomous mobile robots algorithms and problems such as active sensing, path planning problem, etc. It also delivers a short description of NAO robot's abilities in completing autonomous tasks and its equipments.

REFERENCES

[1] M. H. Hebert, Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon, Kluwer Academic Publishers, 1997.

[2] W.Hall, and T.Arlington, unmanned aerial vehicles, 2010.

[3] A. Caiti, *Underwater Robots: Past, Present and Future*.

[4] R. A. Brooks, C. Breazeal, M. Marjanović, B. Scassellati, M. M. Williamson, "The Cog Project: Building a Humanoid Robot," in *Lecture Notes in Computer Science*, 1999, pp. 52-87.

[5] M. Raibert, K. Blankespoor, G. Nelson, R. Playter, "BigDog, the Rough-Terrain Quadruped Robot," in *The International Federation of Automatic Control*, 2008.

[6] "more than four," [Online]. Available: http://en.wikibooks.org/wiki/Robotics/Types_of_Robots/Walkers. [Accessed 1 10 2014].

[7] D.R.Montello, and Sas. Corina, "Human Factors of Wayfinding in Navigation".

[8] "Simultaneous Localization and Mapping (SLAM)," [Online]. Available: https://www.tu-chemnitz.de/etit/proaut/forschung/SLAM.html. [Accessed 2 10 2014].

[9] S. Thrun and W.Bugard, probabilistic robotics, MIT Press, 2005.

[10] H. Moravec and A. E. Elfes, "High resolution maps from wide angle sonar," in *Robotics and Automation*, 1985.

[11] R. Smith, M. Self, and P. Cheeseman, "A Stochastic Map For Uncertain Spatial Relationships," in *International Symposium on Robotics Research*, 1987.

[12] R. A. Brooks, "Visual map making for a mobile robot," in *Robotics and Automation*, 1985.

[13] M. J. Mataric, "Environment learning using a distributed representation," in *Robotics and Automation*, 1990.

[14] L. Mihaylova , T. Lefebvre , H. Bruyninckx , K. Gadeyne , and J. De Schutter, "active sensing for robotic - A Survey," 2002.

[15] Edwin K. P. Chong, Christopher M. Kreucher, and Alfred O. Hero III , "Partially Observable Markov Decision Process Approximations for Adaptive Sensing,"

Discrete Event Dyn Syst, pp. 377-422, 2009.

[16] R. Ritala.and Mikko Lauri, "Stochastic control for maximizing mutual information in active sensing," in *ICRA 2014 Workshop on Robots in Homes and Industry*, 2014.

[17] "NAO software documentaion," [Online]. Available: http://doc.aldebaran.com/1-14/family/index.html. [Accessed 2 10 2014].

[18] "NAO humanoid robot connectivity," robotic center, [Online]. Available: http://www.robotcenter.co.uk/pages/nao-humanoid-robot-connectivity. [Accessed 5 10 2014].

[19] P.Newman, "On The Structure and Solution of the Simultaneous Localisation and Map Building Problem," sydney, 1999.

APPENDIX 1

Implementation python codes

Main class

```
from robot import *
from planner import *
from world import *
speaker    = ALProxy("ALTextToSpeech", "nao.local", 9559)
my_nao = robot()
print 'hello before world'
enviournment = world()
print 'hello after world'
my_nao.robot_init()
print 'hello after init robot'
counter = 0
while True:
  nao_x,nao_y,nao_theta = enviournment.localize_robot()
  if nao_x == 0 and nao_y == 0:
    speaker.say('can not localize. I need more feature')
    my_nao.robot_changeOrientation()
    nao_x,nao_y,nao_theta = enviournment.localize_robot()
  print 'hello after after run for localize'
  mapped_marker = enviournment.update_map(nao_x,nao_y,nao_theta)
  my_nao.head_forward()
  new_planner =planner([nao_x,nao_y],[170,100])
  target_heading = new_planner.get_heading()

  print 'heading'+str(target_heading)
  target_distance = new_planner.get_distance()
  print 'distance'+str(target_distance)
  if target_distance<=30:
    speaker.say('target reached')
    break
  my_nao.robot_turn(target_heading,nao_theta)
  my_nao.robot_move(target_distance)
```

Robot Class

```
import time
import math
from naoqi import ALProxy
import almath
motion= ALProxy("ALMotion", 'nao.local', 9559)
class robot(object):


  def __init__(self):
   # self.motion= ALProxy("ALMotion", 'nao.local', 9559)
    self.memory = ALProxy("ALMemory", 'nao.local', 9559)
    self.speaker    = ALProxy("ALTextToSpeech", "nao.local", 9559)
    self.sonar = ALProxy("ALSonar", 'nao.local', 9559)



  def robot_turn(self,target_heading, pose_heading):
```

```
    motion.setStiffnesses("Body", 1.0)
    turning_angel = target_heading - pose_heading
    if turning_angel>180:
      turning_angel-=360
    if turning_angel<-180:
      turning_angel+=360
    turning_angel_rad = turning_angel* almath.TO_RAD
    turn = round(turning_angel_rad,4)
    # id can be given using post attribute of any proxy for any event
    print "I am going to turn"+str(turning_angel)
    print "its radian"+str(turning_angel_rad)
    motion.moveTo(0, 0,turn )

    #wait is a method for any ALProxy which cause them to wait.
    # it can use an id to wait for another thing to happen before this
ALProxy
    while motion.moveIsActive():
      time.sleep(0.2)
      print "still turnng"



    self.speaker.say("let's go ")
  def robot_move(self,distance):
    available_distance = self.check_distance_step()
    print 'available distance is:'+str(available_distance)
    move_command = min (int(distance),50)
    move_command = math.floor(move_command)/100.0
    move = round(move_command,1)
    print 'the distance is :'+str(distance)
    print 'robot should move :'+str(move)
    if available_distance>move_command:
      motion.moveTo(move,0,0)
      while motion.moveIsActive():
     time.sleep(0.2)



  def robot_init(self):
    motion.setStiffnesses("Body", 1.0)
    motion.moveInit()

  def head_forward(self):
    motion.setStiffnesses("Head", 1.0)
    print 'inside head forward'
    motion.setAngles("HeadYaw", 0.0, 0.3)
    time.sleep(1)


  def check_distance_step(self):
    self.sonar.subscribe("myApplication")
  # Get sonar left first echo (distance in meters to the first obsta-
cle).
    Rval-
ue=self.memory.getData("Device/SubDeviceList/US/Left/Sensor/Value")

  # Same thing for right.

Lvalue=self.memory.getData("Device/SubDeviceList/US/Right/Sensor/Value
")
```

```
    return min(Lvalue,Rvalue)
  def robot_changeOrientation(self):
    turning = 100*almath.TO_RAD
    motion.moveTo(0, 0, turning)

    #wait is a method for any ALProxy which cause them to wait.
    # it can use an id to wait for another thing to happen before this
ALProxy
    while motion.moveIsActive():
      time.sleep(0.2)
```

Planner Class

```
import math
import almath
import numpy as np

class planner(object):
  '''plans the direction of move for the next movement
  also the distance approximation till target
  '''

  def __init__(self, robot_pose,target_pose):
    # initials with a pose and a target location
    self.robot_pose = robot_pose
    self.target_pose = target_pose
    self.y_distance =  float(self.target_pose[1]-self.robot_pose[1])
    self.x_distance =  float(self.target_pose[0]-self.robot_pose[0])

  def get_distance(self):
    #distance in meter to target


    distance = math.sqrt (self.x_distance**2+self.y_distance**2)
    return distance
  def get_heading(self):
    #heading angle to target with respect to the X axis
    print 'the y_distance'+str(self.y_distance)
    theta= math.atan2(self.y_distance,self.x_distance)*180/math.pi


    return theta
```

World Class

```
import time
from naoqi import ALProxy
import almath
from getMArkerCord import *
ip = 'nao.local'
memoryProxy = ALProxy("ALMemory", ip, 9559)

landmarkProxy = ALProxy("ALLandMarkDetection", ip, 9559)
speaker    = ALProxy("ALTextToSpeech", "nao.local", 9559)
head_yaw =  memor-
yProxy.getData("Device/SubDeviceList/HeadYaw/Position/Actuator/Value")
```

```
world_known_markers = {124:[24,181],80:[230, 122],85:[0,131], 171:
[0,61],64:[50,0],187:[230,55],108:[170,181]}
dic_detected_markers ={}
class world (object):
    ''' the intention of the class is to grab all information about
the world
    that the robot is in it '''

    def __init__(self):
      pass


    def localize_robot(self):
      speaker.say("localization starts")
      global dic_detected_markers
      dic_detected_markers = get_MarkerData()
      print 'the detected marker dictionary is :
'+str(dic_detected_markers)

      exactX, exactY, exactHead = getExactLoca-
tion(world_known_markers, dic_detected_markers)
      print 'world localization give the location at
'+str([exactX,exactY,exactHead])
      return [exactX,exactY,exactHead]

    def update_map(self,X,Y,theta):

      new_mapped= mapMarkers(X,Y,theta, world_known_markers,
dic_detected_markers)
      print 'the new markers are located'+ str(new_mapped)

def get_MarkerData():
  #import rpdb2; rpdb2.start_embedded_debugger('robo')
  landmarkProxy.subscribe("landmarkTest")
  markerList = []
  markerListIndex = []
  dic_detected_markers ={}
  counter=0
  indexCounter = 0

  angels =[115,90,60,30,0,-30,-60,-90,-115]

  for angle in angels:
    head_turn(angle)
    newMarkerList = getMarkerData()
    print 'it\'s in first for loop'
    for item in newMarkerList:
      print 'it\'s in second for loop'
      if item[1] not in markerListIndex:
     print 'it\'s in if statement'
        markerList.insert(counter,item)
        x, y=  calculateRelCord(item[0][1], item[0][2],item[0][3])
        dic_detected_markers[item[1][0]]= [x,y]
        markerListIndex.insert(indexCounter,item[1])
        counter+=1
        indexCounter+=1
        print 'length of marker indexes'+str(len(markerListIndex))
```

```
  landmarkProxy.unsubscribe("landmarkTest")
  return dic_detected_markers


def getMarkerData():

  markers_detected =[]
  markData = memoryProxy.getData("LandmarkDetected")



  if len(markData)!= 0:
    print("Found markers!")
    markers_detected = markData[1]
  else:
    print("Nothing found :(")

  return markers_detected



def head_turn(angle):
  try:
        motion = ALProxy("ALMotion", ip, 9559)
  except Exception,e:
        print "Could not create proxy to ALMotion"

motion.setStiffnesses("Head", 1.0)

    # Example showing a slow, relative move of "HeadYaw".
    # Calling this multiple times will move the head further.
  names            = ["HeadYaw", "HeadPitch"]
  set_angle          = [angle*almath.TO_RAD,0]
  fractionMaxSpeed = 0.4

  motion.setAngles(names, set_angle, fractionMaxSpeed)

  time.sleep(1.5)

  print 'head turned '+str(angle)
```

GetMArkerCord package (Markers Calculator)

```
import math
from sympy import *
import almath
from naoqi import ALProxy
import time
landmarkTheoreticalSize = 0.108 #in meters
currentCamera = "CameraTop"
motionProxy = ALProxy("ALMotion", 'nao.local', 9559)
memoryProxy = ALProxy("ALMemory", 'nao.local', 9559)
def calculateRelCord(zImage,yImage, angularSize):
  distanceFromCameraToLandmark= landmarkTheoreticalSize / ( 2 *
math.tan( angularSize / 2))
  # Get current camera position in NAO space.
  transform = motionProxy.getTransform(currentCamera, 2, True)
  transformList = almath.vectorFloat(transform)
  robotToCamera = almath.Transform(transformList)
  # Compute the rotation to point towards the landmark.
```

```
  cameraToLandmarkRotationTransform = al-
math.Transform_from3DRotation(0, yImage, zImage)
  # Compute the translation to reach the landmark.
  cameraToLandmarkTranslationTransform = al-
math.Transform(distanceFromCameraToLandmark, 0, 0)

  # Combine all transformations to get the landmark position in NAO
space.
  robotToLandmark = robotToCamera * cameraToLandmarkRotationTransform
*cameraToLandmarkTranslationTransform
  # final relative y and x cordinate of a marker in robot frame
  x = robotToLandmark.r1_c4
  y = robotToLandmark.r2_c4

  return x , y

def calculateRobotPose(x1, y1 ,x2 ,y2 , robot-
ToLandX1,robotToLandY1,robotToLandX2,robotToLandY2):

  a,b,c,d =symbols('a b c d')
  S=solve([a*x1-b*y1+c-100*robotToLandX1,a*y1+b*x1+d-
100*robotToLandY1,a*x2-b*y2+c-100*robotToLandX2,a*y2+b*x2+d-
100*robotToLandY2],[a,b,c,d])

  robotx, roboty=symbols('robotx roboty')
  R=solve([S[a]*robotx-
S[b]*roboty+S[c],S[b]*robotx+S[a]*roboty+S[d]],[robotx,roboty])
  print 'S[a] is equal:' + str(S[a])
  print 'S[b] is equal:' + str(S[b])
  if (-S[b]<=0.9 and -S[b]>=-0.9) or (S[a]<=0.9 and S[a]>=-0.9):
    if (-S[b]>0) & (S[a]>0):
      angle,CosUsed = (-S[b],S[a])
      angle,CosUsed = getAnglefromSinCosine(-S[b],S[a])
    elif (-S[b]>0)& (S[a]<0):
      angle,CosUsed = getAnglefromSinCosine(-S[b],S[a])
      if CosUsed == False:
        angle= 180 - angle
    elif (-S[b]<0) & (S[a]>0):
      angle,CosUsed = getAnglefromSinCosine(-S[b],S[a])
      if CosUsed == True:
        angle = -angle
    else:
      angle,CosUsed = getAnglefromSinCosine(-S[b],S[a])
      if CosUsed == True:
        angle = -angle
      else:
        angle =-180-angle
  else:
    angle =-1000



  return R[robotx], R[roboty], angle

def getMarkerData():
  markData = memoryProxy.getData("LandmarkDetected")
  while (len(markData) == 0):
    markData = memoryProxy.getData("LandmarkDetected")

  markerList = markData[1]
  #print "this is the return for marker known"+ str(markerList)
```

```
  return markerList
def MarkerListextend(Item, List):
  if Item in List:
    ItemNumber=List.index(Item)
  else:
    List.append(Item)
    ItemNumber=List.index(Item)
def headTurn():
   motionProxy.setStiffnesses("Head", 1.0)

    # Example showing a slow, relative move of "HeadYaw".
    # Calling this multiple times will move the head further.
   names            = "HeadYaw"
   changes          = 30.0*almath.TO_RAD
   fractionMaxSpeed = 0.05
   motionProxy.changeAngles(names, changes, fractionMaxSpeed)


   time.sleep(2.0)

   motionProxy.setStiffnesses("Head", 0.0)
# get the exact location of robot and it's heading by adding the value
of  of each pair of known marker on differnet walls
def getExactLocation(worldCord , dicMarkerlist):
  localized=0
  exactHead=0
  exactlocationX = 0
  exactlocationY = 0
  meanXlocation = 0
  meanYlocation = 0
  meanHeading = 0
  finalMeanHead = 0
  headingsList= []
  for key1 in worldCord:
    for key2 in worldCord:

      if (key1  in dicMarkerlist) & (key2 in dicMarkerlist) & (world-
Cord[key1][1] != worldCord[key2][1] ) & (worldCord[key1][0] != world-
Cord[key2][0]):
      print 'markers used for localization
are'+str(key1)+'and'+str(key2)

      robotPoseX, robotposeY, heading = calculateRobot-
Pose(worldCord[key1][0],worldCord[key1][1], worldCord[key2][0], world-
Cord[key2][1], dicMarker-
list[key1][0],dicMarkerlist[key1][1],dicMarkerlist[key2][0],dicMarkerl
ist[key2][1])
      if robotPoseX>0 and robotPoseX<230 and robotposeY>0 and robotpos-
eY<180 and heading!=-1000:
        localized+=1
        meanXlocation += robotPoseX
        meanYlocation += robotposeY
        meanHeading += heading
        headingsList.append(heading)


      print "new location is : "+str([robotPoseX,robotposeY,heading])
  if localized !=0:
    if (meanHeading/localized>0 ):
      for item in headingsList:
        if item<-170:
```

```
            headingsList[headingsList.index(item)] = 360+ item
            print 'nothing is wrong'
        elif (meanHeading/localized<0):
          for item in headingsList:
            if item> 170:
              headingsList[headingsList.index(item)] = -360+ item

        for item in headingsList:
          finalMeanHead= item+ finalMeanHead


        exactlocationX =meanXlocation/localized
        exactlocationY =meanYlocation/localized
        exactHead = finalMeanHead/ localized
      print "heading list is:"+ str(headingsList)
      return exactlocationX, exactlocationY,exactHead
# this function gets the value of x and y of an non located marker in
robot frame and mao them to real world map!
def mapMarkers(X,Y,heading, worldmarkers, markerList):
      mappedMarker={}
      headCos = math.cos(math.pi*heading/180)
      headSin = math.sin(math.pi*heading/180)

      print 'cosine value'+ str(headCos)
      print 'sine value ' + str(headSin)
      for key in markerList:
       if key not in worldmarkers:

          print str(key)+'markerList value '+ str(markerList[key])

          testX = headCos*markerList[key][0]*100-
headSin*markerList[key][1]*100+X
          testY =
headSin*markerList[key][0]*100+headCos*markerList[key][1]*100+Y
          mappedMarker[key] = [testX,testY]
      print "mapped markers are:"+ str(mappedMarker)
      return mappedMarker
# this function gets the cosine or sine depending of the value of co-
sine (if it is between 0.9,-0.9 cosine is used)
# otherwise we use sine
def getAnglefromSinCosine(sinPart, cosPart):
      CosUsed= False
      if (cosPart>=-0.9) & (cosPart<=0.9):
        head1 = math.acos(cosPart)*180/math.pi
        print 'head1 of cosine : '+str(head1)
        CosUsed = True
      else :
        head1 = math.asin(sinPart)*180/math.pi
        print 'head1 of sine : '+str(head1)


      return head1,CosUsed
```