SAYABOT – A SERVICE ROBOT FOR HOSPITALITY

S Gokul Narayanan (12R212)
M J Niranjan (13R439)

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: ROBOTICS AND AUTOMATION ENGINEERING

of Anna University

April 2016

DEPARTMENT OF ROBOTICS AND AUTOMATION ENGINEERING
PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)
COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

SAYABOT – A SERVICE ROBOT FOR HOSPITALITY

Bona fide record of work done by

S Gokul Narayanan (12R212)
M J Niranjan (13R439)

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: ROBOTICS AND AUTOMATION ENGINEERING

of Anna University

April 2016

...……	……..……
Dr. B. Vinod	**Dr. B. Vinod**
Faculty Guide	Head of the Department

Certified that the candidate was examined in the viva-voce examination held on …….

……..	……..
(Internal Examiner)	(External Examiner)

CONTENTS

ACKNOWLEDGEMENT

It is a pleasure to pay our allegiance to **Dr.R.Rudramoorthy,** Principal, PSG College of Technology for having provided the necessary facilities.

We are greatly indebted to **Dr.B.Vinod**, Head of the Department of Robotics and Automation Engineering and also our faculty guide, for the support and guidance that he extended towards this project work.

We wish to express our indebted gratitude and special thanks to **Mr.T.Jayakrishnan,** C.E.O, ASIMOV Robotics, Cochin who allowed us to carry out our industrial project work at their esteemed organization.

We express our deepest thanks to **Mr.Ravi Kumar Venkat**, Principal Robotics Engineer, for taking part in useful decision & giving necessary advices and guidance and arranged all facilities to make life easier. We choose this moment to acknowledge their contribution gratefully.

We thank all the faculty members, the supporting staff and the non-teaching staff for their timely help, co-operation and co-ordination during the course of our project.

Finally we thank our parents, for their continuous support and encouragement in doing the project.

Above all, we thank the Almighty for the blessings showered on us to complete our project work successfully.

SYNOPSIS

Service industries like hospitality, healthcare, financial services are beginning to dip their toes in new robotic technologies due to their increased functionalities and declining cost of deployment. As a result, the service industry may eventually find deployment of robots a competitive necessity. Our objective was to identify the right sensors, customize the existing algorithms and integrate them into a complete system in ROS (Robot Operating System) for the service robot, Sayabot*, to address the requirements of the above mentioned Service Industries.

The software on this robot was organized to have the following four core software modules: Perception, Voice Synthesis, Arm Manipulation and Navigation. Here the Perception module was designed to process the inputs as facial images of the user standing in front of the robot. Voice synthesis and Arm manipulation modules, which were triggered by the outputs of the Perception module using State Machine libraries in ROS, were designed to function simultaneously with voice and hand gestures, to give the users a feel that they are interacting with a human. The last module i.e. Navigation was designed to navigate autonomously and guide the user to the commanded pre-defined destinations.

The software modules were tested with their respective hardware and validated accordingly. Then the modules were integrated and deployed in a real-time scenario, and the results were documented. *Sayabot is a robot, which has it's hardware designed and built in at ASIMoV Robotics Pvt. Ltd, comprises a mobile base, two 3 DOF arm manipulators, a passive Hand, and head with active lip movements.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

For more than three decades, the robots have been used for automating the industrial work which was found to be dangerous and laborious for humans. But this is not the situation anymore. As the technology evolved, humans found that the potential for the robots are not only in the field of manufacturing but also in other fields too. If technology can automate the mundane tasks of human, it will free them to focus on higher-value activities. The mundane tasks generally come under the service sector. So, currently the market for the service robot is in boom. The biggest markets are expected to be hospitality, banking, and tourism. It is anticipated that by 2018, a total of over 150 000 units will be sold at a total value of around 20 billion US dollars (2015 World Robotics Survey). One of the important thing to be noted is that of all the enterprises that are involved in robotics, 15% of them are startups which shows that, investors and entrepreneurs have identified the market potential is growing in the field of robotics. Hospitality robots are becoming common everywhere since it aids humans in several ways. Our motive is to develop a software system for a service robot 'SAYABOT' using open source robot platform 'ROS'. SAYABOT has a mobile base, two 3 DOF arm manipulators, a passive hand and head with active lip movements. The banking sector was took as an use case and the software system was customized accordingly

1.1 PROJECT OBJECTIVE

The objective of the project is to compare different sensors and choose the proper one that fits the needs and requirement of a service robot in banking scenario and then to develop a software system by customizing the existing algorithm and integrating them into a complete system. The software on this robot should be organized to have the following four core software modules: Perception, Voice Synthesis, Arm Manipulation and Navigation. The goal is to have the SAYABOT working in a banking scenario with the software system

developed to help the humans in navigating through the bank. The project was divided into two phases .The first phase of the project involves sensor selection and integrating the sensors with perception module. The second phase focuses on customizing the code for the arm manipulation, voice synthesis, navigation and testing them.

1.2 PROJECT DESCRIPTION

John is a customer coming to the bank to open a saving account in the bank. He enters the bank and finds it very difficult to know about the various sections in the bank. Everyone is busy doing their work so he couldn't find the section which he is looking for. He then saw SAYABOT and goes near it, to see what it does. Fig 1.1 shows the simple requirement for the SAYABOT.

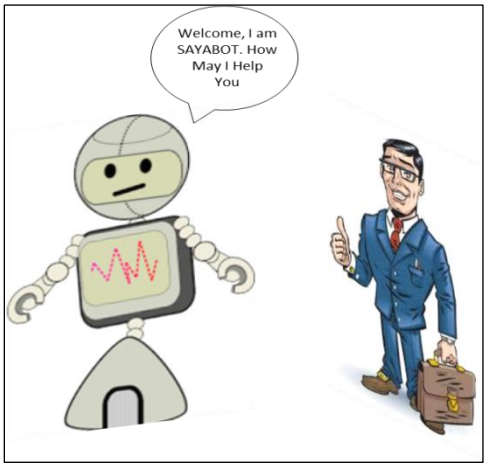

Fig 1.1 SAYABOT

As he reaches close to SAYABOT, it greets him through voice and hand gesture. Then the SAYABOT introduces itself and ask him to select the section that he is looking for through the interactive screen. Then John chooses the account section in the screen. Now SAYABOT asks whether it should direct him through voice and hand gesture or to lead him to the section. John choses the option that will to lead him to the account section. SAYABOT requests him to follow it. Then it starts to maneuver to the account section carefully by avoiding obstacles. After reaching account section SAYABOT greets him and returns to its initial position.

1.3. ORGANIZATION OF REPORT

The report is organized into 8 chapters as follows:

Chapter 2 describes the system level requirements of the robot, functional and non-functional.

Chapter 3 describes the functional architecture of various modules present in the robot.

Chapter 4 describes the various trade studies carried out to select the components.

Chapter 5 describes in detail the flow of information between the hardware and software.

Chapter 6 gives the complete description and working of each module in separate.

Chapter 7 gives an outline of the project management

Chapter 8 describes the results of the project and the future work to be done

CHAPTER 2

SYSTEM-LEVEL REQUIREMENTS

Based on the user needs, the functional and non-functional requirements for the system is formulated and grouped under either mandatory requirements or desirable requirements. They were again divided into functional and non-functional requirements. Functional requirements deal with the technical aspects of the system. A non-functional requirement deals with aesthetics of the robot.

2.1 MANDATORY REQUIREMENTS

2.1.1 Functional Requirements

Table 2.1 Mandatory Functional Requirements

S.No.	Requirements	Description
M.F.1	Detecting Human Presence	SAYABOT should be able to detect the human at a range of 1 meter from the base.
M.F.2	Greet with voice and arm gestures	SAYABOT should greets him/her saying Good Morning/Afternoon/Evening with an appropriate hand gesture
M.F.3	Accept command from customer through touch screen	The touch screen should display two options either to route to counter through voice and had gesture or to move in front of them to the counter. Once customer chooses the option then SAYABOT should act accordingly.
M.F.4	Drive Mechanism	Robot should have differential drive for the movement of the robot.
M.F.5	Navigation through the bank	SAYABOT should autonomously navigate to different counters in the bank avoiding obstacles and this should be achieve with goal tolerance of 15cm.

Table 2.1 contains the mandatory functional requirements of the SAYABOT.

2.1.2 Non-Functional Requirements

Table 2.2 Mandatory Non-Functional Requirements

S.No.	Requirements	Description
M.F.1	Optimal Height	Height of the robot should be varying 4.5ft to 6ft
M.F.2	On – board power source	On-board power source should have an backup of at least 4 hours
M.F.3	Appearance	Robot should have female face
M.F.4	User Interface	10’ inch display with touch

Table 2.2 contains the mandatory non-functional requirements of the SAYABOT.

2.2 DESIRABLE REQUIREMENTS

2.2.1 Functional Requirements

Table 2.3 Desirable Functional Requirements

S.No.	Requirements	Description
D.F.1	Safety (Manual Control And Low Battery Indicator)	There shall be manual control feature in case of any failure.
D.F.2	Face Tracking	Robot shall track the face of a customer through the pan movement of its head.
D.F.3	Manipulation	2 Arms with 3 DOF

Table 2.3 contains the desirable functional requirements of the SAYABOT.

2.2.2 Non Functional Requirements

Table 2.4 Desirable Non-Functional Requirements

S.No.	Requirements	Description
D.N.F.1	Friendly Looking	Robot shall have an aesthetic / appealing external appearance

Table 2.4 contains the desirable non-functional requirements of the SAYABOT.

CHAPTER 3

FUNCTIONAL ARCHITECTURE

Based on the requirements of the robot, the core functions of the robot is determined and grouped under five subsystems. They are Perception, User interface, Communication, Manipulation, and Navigation. The functional architecture describes in detail about each of this subsystem and their specification. It also shows the flow of information between different subsystems.

3.1 USER INTERFACE

SAYABOT have a 7” inch touch screen display attached to the chests of it which will be used as an interface for getting inputs from the users. Once the user presence is detected, the robot greets and the screen shows various sections available in the bank and the user can select an option in it. Once the option is selected by the user, it will be published to the corresponding ros topic to carry the next function. The screen will be showing an animated map with path from the current position to destination. It also has options like repeat, escort, thank you for better interaction. The data from the touch screen are processed through ROS bridge package. These things were designed by the user interface team in ASIMoV robotics as per the requirements. A package was built to process the data from ROS bridge and trigger the corresponding module.

3.2 PERCEPTION

To perceive information about the environment, a vision sensor and LIDAR were used.

3.2.1 Vision Sensor (Camera)

The camera continuously streams video of the environment and the video is processed to detect humans in the environment.

The functional architecture of the SAYABOT is shown in the Fig 3.1.

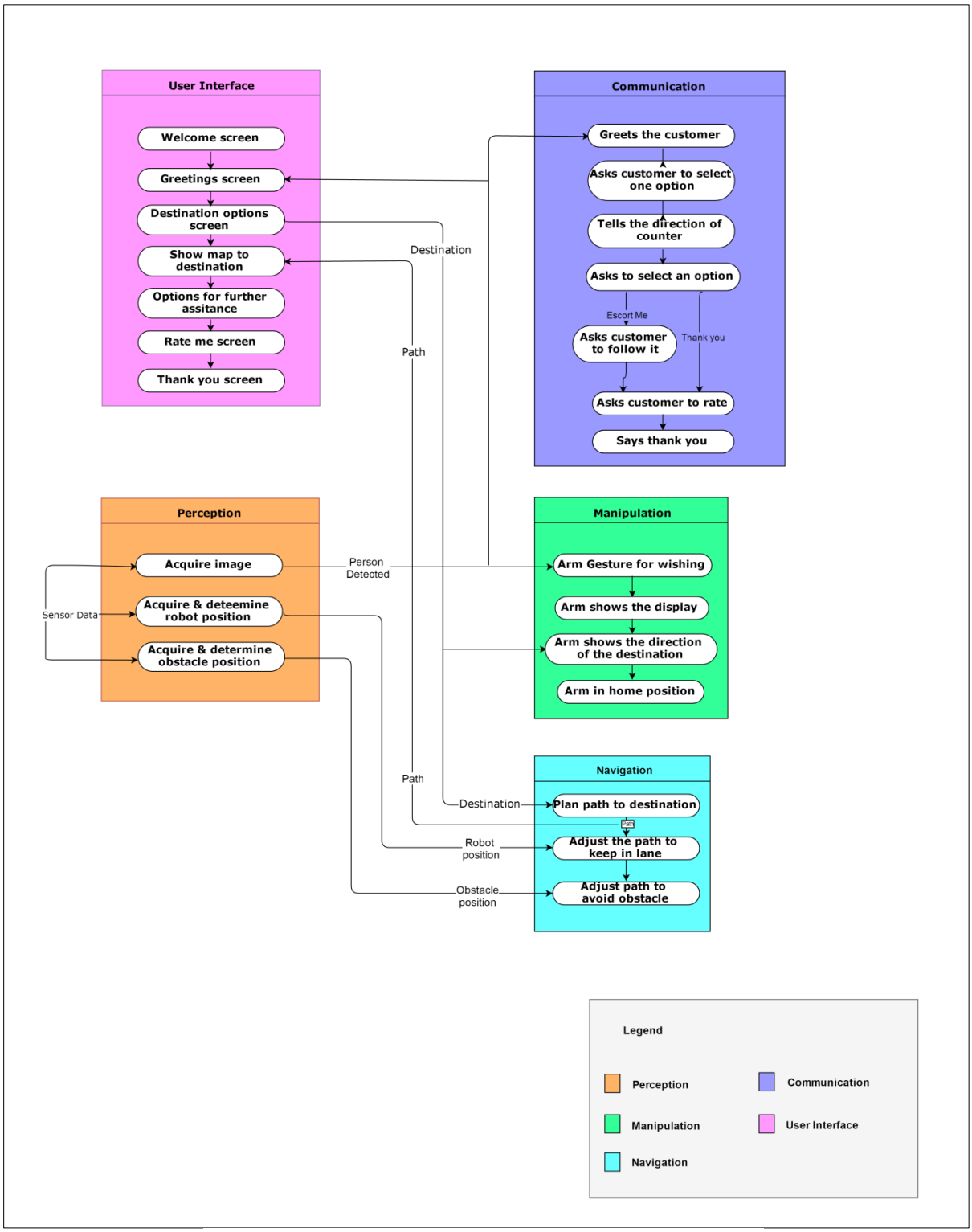

Fig 3.1 Functional Architecture of the Robot

3.2.2 LIDAR (Hokuyo)

LIDAR continuously scans the environment and updates the information about the environment in the map fed by the map server. This data is used to localize the robot in the map using adaptive monte carlo localization (amcl) technique.

3.3 MANIPULATION

SAYABOT has a 3 DOF arm. Dynamixel servo motors are used as actuators in arm. The arm is used to show gestures and directions to the users. Since the gestures of the robot arm are known beforehand, the motion of the arm is recorded using the rosbag command and the motion is repeated when the corresponding module is triggered.

3.4 NAVIGATION

Localization: Localization is done by the perception module and the position in the map is passed as co-ordinates to the navigation stack.

Path planning: The planner generates a trajectory for the robot from its current position to the goal position. This trajectory must be feasible for the robot and also the shortest from the starting position to the goal.

Obstacle avoidance: The navigation subsystem uses an occupancy grid based cost-map to localize and map obstacles in the environment. The system is capable of avoiding dynamic obstacles and can re-plan the trajectories online.

Navigation towards the Goal: Using the trajectory generated by the planner, the move base package commands the robot's base to navigate towards the goal while avoiding static and dynamic obstacles. If the path planner is not able to generate a feasible path from the starting position to the goal due to complete obstructions, the robot will stop.

3.5 COMMUNICATION

The communication module is responsible for the speech output and synchronizing it with the movement of lip accordingly. The module is programmed with sentences for greeting the user and showing directions.

CHAPTER 4

TRADE STUDY

In order to select the components required for various modules, trade study was done on the components based on the requirements .Points were weighed on a scale of 0-5 for various components, and they are tallied to get the final weight. Based on the highest weightage, the components are chosen accordingly. This trade study was carried out for on-board computer, camera, and LIDAR.

4.1 ON – BOARD COMPUTER

While choosing the on-board computer, five main factors were considered:

1. **Cost** – limited budget is allocated for on-board computer.
2. **Compatibility** - It needs to be compatible with the software that was developed, including code dependencies, drivers for Hokuyo laser and the Dynamixel servos.
3. **Performance of the processor** – SAYABOT needs at least a 5th gen. dual-core processor.
4. **Power Requirements** - Since the on-board computer will be powered from a battery pack on the robot, SAYABOT should be as power efficient as possible in order to have decent operation time.
5. **Form Factor** – SAYABOT has little room at the bottom where the on-board computer can be placed. This limited the size of the computer that can be chosen so it was an important fact to consider while selecting the computer.

Table 4.1 Trade Study on Different On-Board Computers

				
Factors	**Gigabyte Brix Pro**	**Zbox- ID90 P**	**Intel NUC**	**Asus Vivo PC**
Cost	4	4	4.2	4.4
Compatibility	3	4	4.5	3.5
Processor Performance	4	4.2	4.5	3.5
Power Requirement	2	3	4	3.8
Form Factor	4	4.5	4.2	4.7
Total	**3.4**	**4**	**4.3**	**3.9**

Table 4.1 shows the trade study different on on-board computers. Each has been scaled to a factor of 5 based on specifications.

The four products that were shortlisted to carry out a trade study. It was decided to go with **Intel NUC** because it offered the best processing power and has a decent power consumption rate, and it also had a dimension that fits inside the space we dedicated for the computer.

4.2 VISION SENSOR

4.2.1 Camera

The task is to select appropriate vision sensor for the robot. The sensor to be chosen will be fitted on the robot face. So, the sensor shouldn't affect the aesthetic of the robot face. The image from the sensor will be processed for facial detection and tracking of humans. So, a research on available vision sensors in the market was done and tabulated the same.

The selection of sensors for our robot is mainly based on Field of View, Form factor, compatibility with Ros and price. Based on these criteria four sensors were finalized which will be appropriate for our robot. They are kinect, Intel Realsense F200, prime sense & iBall Web Camera C 12.0.

Kinect & primesensor are the most commonly used sensors for robotic vision because of their low cost and depth image capabilities which enables to capture the details of environment more precisely. But the form factor of these sensor is high that makes the face of the robot less aesthetic. This made them as inappropriate for the robot.

Intel Realsense F200 is the new vision sensor available in the market. This sensor is cheaper than kinect and primesense and its form factor is also less. Even though the form factor is less, this sensor when used in the robot doesn't give the human face like appearance.

Table 4.2 Trade Study on Different Vision Sensors

S.no	Camera	Price (Rupees)	Size	Pros	Cons
1	Kinect	11 000	305×76×64mm Wt(high) 3lbs	Quality device drivers Error <0.01m for distance less than 3m	Not suitable for long range
2	Primesense	18 000	178×51×38mm	Doesnt require external powersupply	Doesnt work with usb3.0
3	Tof sensors Eg:camcube	80 000	Varies	Suitable for long range	Gray scale images only
4	Bumblebee2	1 20 000	153×46×36mm	Drivers in ros. Compact	High cost
5	Structure sensor	26 000	Compact	Supports OpenNI 2	Need separate Hacker cable to use. Cost: Rs.2,000
6	Intel realsense camera (F200)	6 700	150×30×58mm	Ros Integration is available	Shorter range (0.2 meters - 1.2 meters, indoors only)
7	Orbbec astra	10 000	Product dimensions: 165×30×40 mm weight: 0.8 pounds	Optimized range: 0.4 - 8.0m (optimized range: 0.6 - 6.0m)	No Ros Package
8	LI-USB30-V024STEREO WVGA	27 000	80×15×17 mm		RAW data output without compression
9	iBall Web Camera C 12.0	1 215	Small size	Low cost	Only RGB Image

Table 4.2 shows the trade study on different vision sensors with their pros and cons.

The USB camera was selected to use as vision sensor. They are very cheap and they can be used in the place of the eyes which doesn't affect the form factor of the face. It also makes the robot face more human like.

The pixel resolution of the camera should be medium so that processing time and memory storage can be reduced. So, the 640 × 480 resolution was finalized and the camera was chosen accordingly.

4.2.2 LIDAR

There are different laser available in the market. The laser must be able cover a wide area in front of it and so it must have wide range. It must consume less current and should be working through USB power. Here are the main characteristics that was looked for

1. **Range:** 270º
2. **Power consumption:** Less than 0.5A
3. **Power source:** USB
4. **Range:** 0.02m to 7m
5. **Accuracy:** less than 0.02 m
6. **Budget:** less than 60000

Table 4.3 Trade Study on Different LIDAR

					
Factors	**Weight**	**Hokuyo UTM-30LX-EW**	**Hokuyo UST-10LX**	**Hokuyo URG-04LX-UG01**	**Sick TiM3xx**
Range	30	3	3	5	3
Accuracy	30	4	3.2	4	3.2
Power consumption	20	2.5	3	4.5	4
Frequency	10	4.5	4	4	4.2
Cost	10	2	3	3.5	4
Total	**100**	**3.25**	**3.16**	**4.35**	**3.48**

The four products that were shortlisted to carry out a trade study are shown in Table 4.3. From the tabulation, Hokuyo URG-04LX-UG01 was selected to be used in the robot for perception.

CHAPTER 5

CYBER-PHYSICAL ARCHITECTURE

Cyber –physical architecture describes the interactions between the various functional components and the software developed for it. They are divided into hardware and software. The hardware section describes the physical elements used and the flow of information through them. The software section describes the method through which the integration of software modules was done using SMACH.

5.1 HARDWARE

The hardware cyber-physical architecture of SAYABOT is shown in the Fig 5.1.

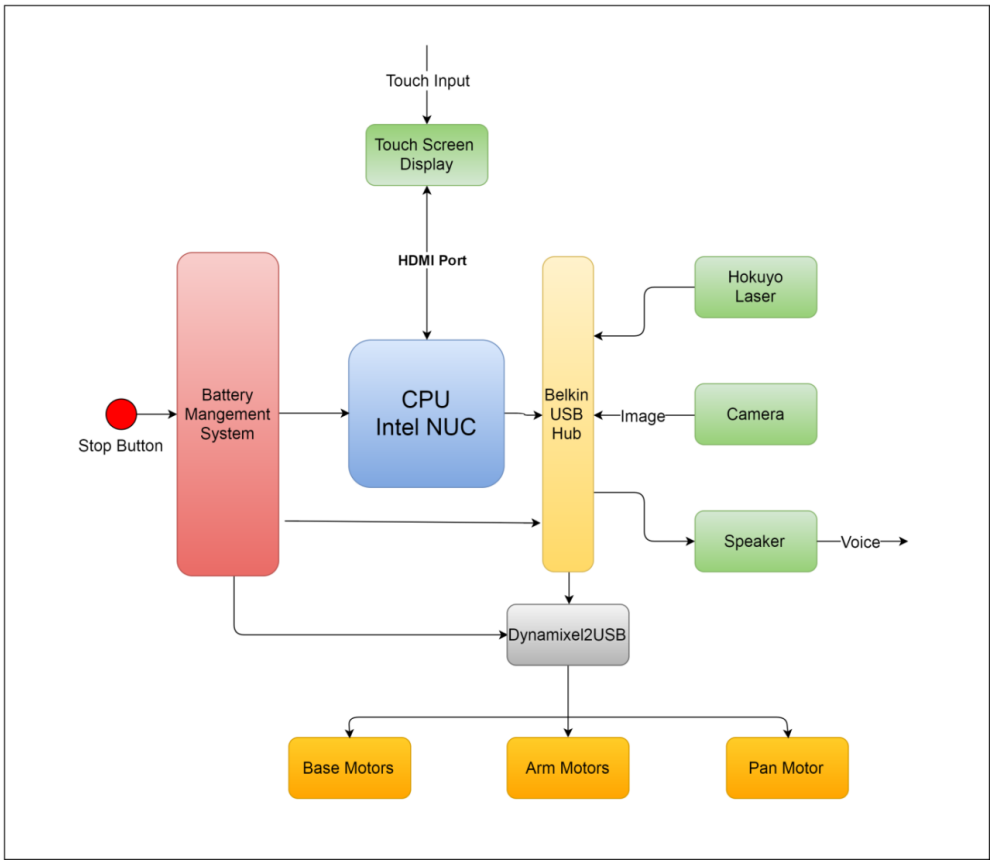

Fig 5.1 Cyber physical architecture of the SAYABOT

Cyber physical architecture describes the physical elements which are used in the system to realize the functional architecture. It also gives an overview about the physical structure which is built and the order in which they are connected.

5.2 SOFTWARE

The software cyber-physical architecture of SAYABOT is shown in the Fig 5.2.

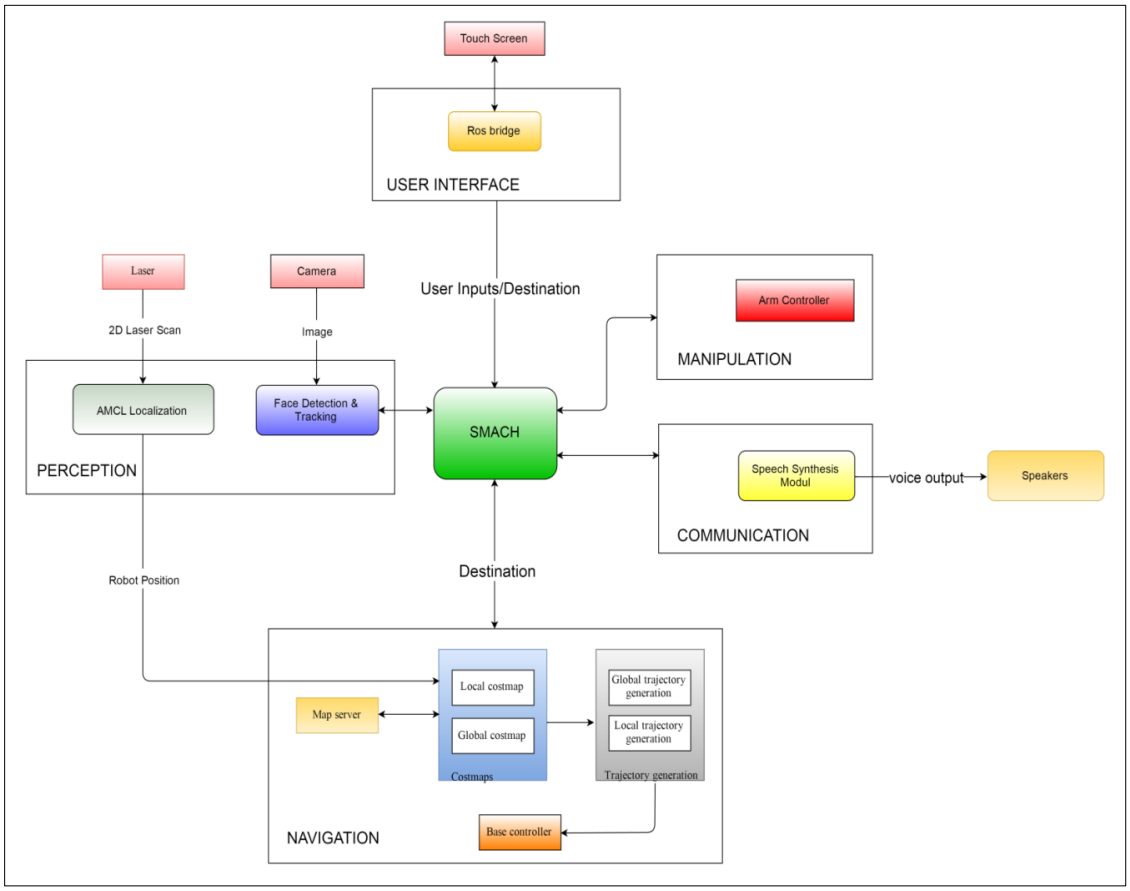

Fig 5.2 Cyber Physical Architecture of the SAYABOT Software

SMACH was implemented to integrate the modules present in system and to create task-level architecture. It also helps to define states and schedule the tasks accordingly.

Action state class in SMACH has been used to implement this architecture. The SMACH states are defined beforehand and they are executed based on each state outcome. Using the state outcomes from each task, the SMACH interacts with the task level state flow.

Initially face detection state will be active and when a user comes in front of the robot, the node will detect the face and provides an outcome ‘Detected’ to the SMACH. Then the SMACH triggers the gesture function in arm manipulation module, welcome screen in user interface module, and greeting function in communication module concurrently. Once all the states get executed and finishes, it sends ‘Succeeded’ outcome to SMACH.

Then the SMACH checks the input received from touch screen. If the input is ‘Escort me’ ,then it triggers the corresponding function in navigation module and home position function in arm manipulation module concurrently or else it triggers the show direction function in arm manipulation module. After the completion of this state, it sends an ‘Over’ outcome.

The next state triggers home position function in navigation and arm manipulation module. After this the outcome will be ‘Task completed’. This will restart the SMACH states again.

CHAPTER 6

SYSTEM DESCRIPTION AND EVALUATION

The system of the robot is divided into several subsystems. They are Mechanical, Perception, Arm Manipulation, Speech Synthesis and Navigation. Each subsystem was tested separately in different testing conditions and the performance was observed and tabulated. Then the results were critically examined and evaluated to find the actual performance of the system.

6.1 MECHANICAL

To test the software modules, the hardware prototype which was already available in the Sayabot were used and modification was done on top of it. The mechanical prototypes which we used for testing are the head, the lips, the arm, and the base.

6.1.1 Head

Fig 6.1 shows the prototype of the head with camera and motors fixed on it.

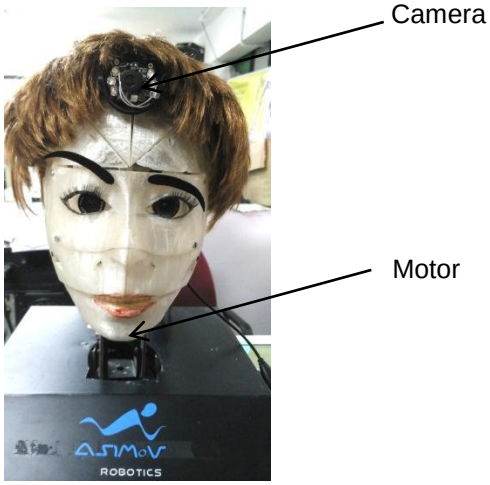

Fig 6.1 Head Prototype for Testing with Camera

Dynamixel servo motors were fixed on the bottom of the head to get the pan movement of the head. Dynamixel MX-64 series was used. The camera was fixed on the top of the face for image processing.

6.1.2 Lip Movement

The lip prototype for the testing is shown in the Fig 6.2.

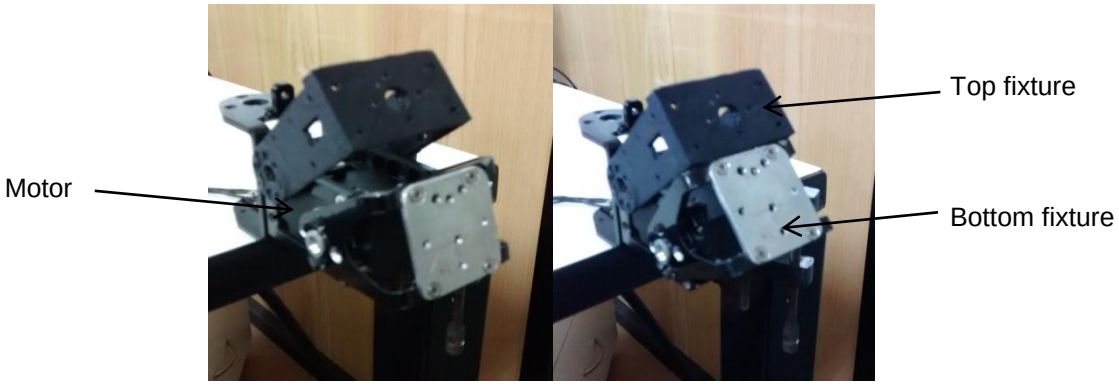

Fig 6.2 Lip Prototype for Testing

For testing the synchronization of lip movement with speech, we built a mechanical module using Dynamixel MX -24 servo motors which is shown in Fig 6.2. Two fixtures were fixed on the motor to act like jaws. The top fixture is fixed over the motor and it's static while the bottom fixture is attached to the motor shaft and it's movable.

6.1.3 Arm

The Cool400 Arm manipulator for the testing is shown in the Fig 6.3.

Fig 6.3 Cool400 Arm

For testing the record and playback function of arm, the cool400 arm built by the ASIMoV robotics which has 6DOF.

6.1.4 Base

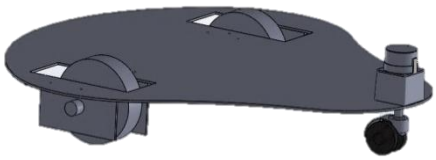

Fig 6.4 CAD Model of Base

Fig 6.5 Real Base

Fig 6.4 shows the image of the CAD model developed for the base .The model was developed as per the future requirement of Sayabot. Fig 6.5 shows the base which was built by the hardware team from the ASIMoV robotics .It contains two Dynamixel servos with wheels attached to it and a castor wheel fixed in front .Laser was fixed on the front part of base. Navigation modules were tested with this base.

6.2 PERCEPTION

Sayabot uses a simple USB web camera and laser to perceive its environment. The images from camera are processed and used for the two main functions.

a. **Face Detection** – Detects the number of human faces in image and publishes the same in a Rostopic.
b. **Face Tracking** – Once a human face is detected, robot's head starts to track it still face remains in the view of the robot.

6.2.1 Face Detection

In order to serve a customer, a major requirement of SAYABOT's perception is to detect a customer whenever he /she present in its view. SAYABOT thus detects human face using an algorithm based on Hog Haar Cascade. The algorithm detects the human face whenever a customer comes into the robot's view i.e. customer comes and stands in front of the robot for the service.

The algorithm first converts the raw RGB images published by a camera into the opencv image format i.e. MAT. Then it applies all the features of trained hog haar cascade classifiers in the image in order to find the human in the image. The algorithm then gives result as an image with faces highlighted and as a Boolean message, thus publishes 1 if the human face is detected in the image else 0 if face is not found.

The result of this algorithm on a test image is shown in Fig 6.6.

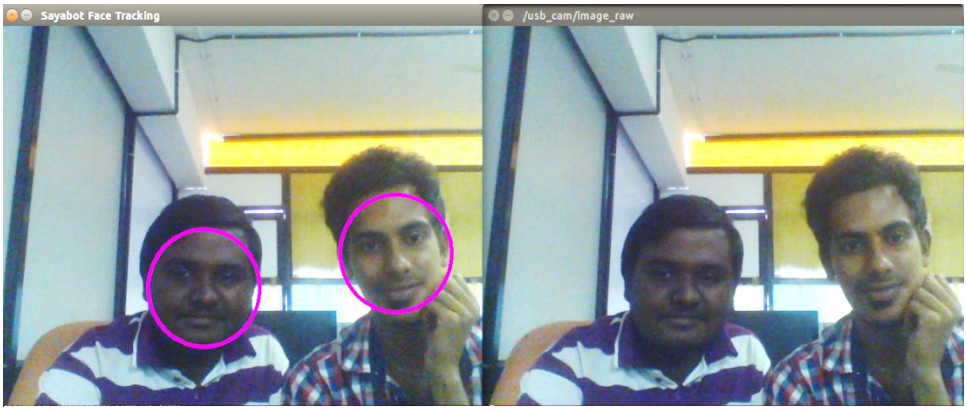

Fig 6.6 Face Detection Images (Right) Raw Image (LEFT) Output

The algorithm was implemented as a roscpp node which subscribes the raw image from the usb_cam node and publishes the message as Ros topic.

6.2.2 Face Tracking

After detecting the human face, SAYABOT will track the face using a pan movement of its head, until customer is present in its view. Co-ordinates of the detected faces are attained from the algorithm which is used for the face detection. Using the X co-ordinated of detected face, movement of the face is suspected and signal to the servo of pan movement of SAYABOT head is given accordingly. Fig 6.7 shows the message passing between the ROS nodes of face tracking.

Fig 6.7 ROS Tropic Graph of Face Tracking

6.2.3 Testing and Validation

Real Time Testing

This experiment was conducted to test the efficiency of face detection algorithm in the real time. A robotic face and fixed a camera at a height of five feet was taken for this experiment. Then dynamixel motor at the bottom of the face to enable pan movement was fixed. Then the robot was kept the robot in an environment similar to the banking environment and a notice requesting the people to take part in this experiment was pasted. Then markings on the floor for minimal distance (0.5m) and maximum distance (1m) was made for the people reference to stand in front of robot. The volunteers were asked to stand within this marking at different angles. So, that the robot can detect and track their face accordingly. First it was tested with the single person in front of robot and then tested with multiple persons, also the video feed from the camera to find the response of different persons on looking at the robot was recorded. Fig 6.8 shows the testing model for this experiment.

Fig 6.8 Testing Model

People's Response

Through the video feed from the camera, it was found that some volunteers moved their face very fast, in order to test the response of the robot's head movement. The algorithm was able to detect the face and track them when they moved in nominal speed. But the pan movement of the robot's head became erratic when they moved very faster.

Some volunteer's face wasn't completely covered because they stood outside the camera's lateral field of view and so their faces weren't detected nor tracked.

There were few volunteers whose height was more than 6.2 feet. The camera was unable to capture their faces and they needed to be bent to come into the view of camera. They expected that the robot should be able to capture their face without them bending.

Some people read the notice about the experiment but they showed no interest to be a part of it because they felt uncomfortable to stand in front of the robot. Some expected facial expression from the robot to make it more realistic.

Results and Validation:

Table 6.1 Results of real time testing

Number Of Person	Detection	Tracking	Remark
1	Detected	Perfect	-
2	Detected	Good	When two persons are very close
3	Detected	Not Perfect	When three person were in frame tracking failed because of overlapping in features of different faces
>4	Partial Detection	Not Perfect	Only part of the faces are covered so detection failed

Through this experiment, the performance of the algorithm in the real time was evaluated and tabulated the same and shown in the table 6.1.

The algorithm detects the face and finds the center of it and adjusts the face such that the human face is at the center of camera frame. There is a processing time involved for this adjustment to take place completely. When the test case was a single person moving with nominal speed, the algorithm detected the face and tracked it perfectly. Even the algorithm worked perfect for two people standing with the distance apart. When two persons were

standing close to each other, the algorithm failed because while tracking instead of following a single persons face, it continuously switches between the faces of both This is because during the movement of robots head, the relative position of the faces from the center of frame are approximately equal .So it becomes difficult for the algorithm choose between the faces. This resulted in jerkier movement of robots head.

When the test case had people three or more, features of faces were hidden for some faces and partially visible for some cases, this decrease the efficiency of the algorithm to detect faces and fail most of the time. During testing it was found that the light and background of the environment also plays an important role in accuracy of the detection. When the light present in then environment is very low or high, some of the features of the faces are being hidden and so sometimes false negative detection happens. When the background of environment is too crowded sometimes false positive detection happens.

6.3 MANIPULATION

Sayabot has two arms manipulators for gesturing like showing NAMASTE sign and showing the direction of destination to the customer. The gesture that is going to be done by sayabot are limited and predefined ones, so the methodology that has been used to do is to record a required motion of the arm by moving it manually and play back those recorded motion of arm to make that gesture.

The arm manipulator in sayabot is a 6DOF arm. It comprises of 6 Dynamixel servos for each joint.

6.5.1 Recording the Motion

Recording the motion of arm can be done by the recording the current positions of the each motor for each joint while moving it manually. When the Controller nodes for servos are brought up in Ros, then it was poosible to subscribe the state of the each motor through the /arm_controller/state Ros topic. Now the state message that is publishing is of dynamixel_msgs/JointState message; it comprised of a header, motor_id, etc. The dynamixel_msgs format is shown in the Fig 6.9.

Message: dynamixel_msgs/JointState.msg

```
Header header
string name          # joint name
int32[] motor_ids    # motor ids controlling this joint
int32[] motor_temps # motor temperatures, same order as motor_ids

float64 goal_pos     # commanded position (in radians)
float64 current_pos # current joint position (in radians)
float64 error        # error between commanded and current positions (in radians)
float64 velocity     # current joint speed (in radians per second)
float64 load         # current load
bool is_moving       # is joint currently in motion
```

Fig 6.9 dynamixel_msgs/JointState.msg

For recording, only the current positions of the each motor are required to subscribe. So a rospy node was created for subscribing the JointState message of each motor and publishes only the current position of each motor to the different ROS topic. Fig 6.10 shows the ROS topic graph of current position publisher.

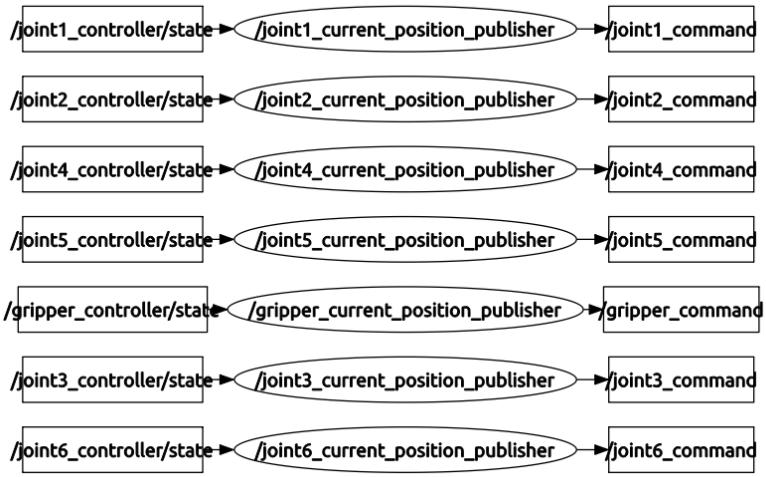

Fig 6.10 ROS Topic Graph of Current Position Publisher

Now the current position of each motor which is getting published to a topic has to be recorded while moving the arm which was by using rosbag command. Rosbag is a ROS package that provides a command-line tool for recording and playing back the messages that are publishing in rostopic. Using rosbag the current position of each motor while moving the arm manually was recorded, this recorded data will get stored in a bag file (file format in ROS for storing ROS message data).

Fig 6.11 shows the ROS topic graph of recorder.

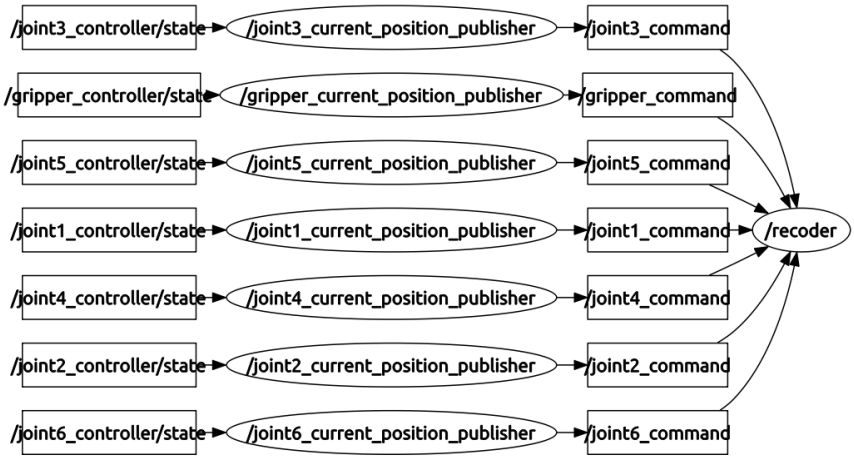

Fig 6.11 ROS Topic Graph of Recorder

6.5.2 Playback of Recorded Motion

A bag file that contains the positions of the each motor recorded while moving the arm manually, now the data that is been stored is like a coordinate point of a trajectory in which the arm is moved manually. To playback the same motion that is recorded, all the motor should move as per the position of the recorded motion. To-do so, the values are published as the goal position commands to the motors. This is done by playing back the bag file and remapping the topic to the commands topic of the each dynamixel motor. As the messages are published to commands topic of motors, motors will move accordingly to the command which will bring up the recorded motion of the arm. Fig 6.12 shows the ROS topic graph of player.

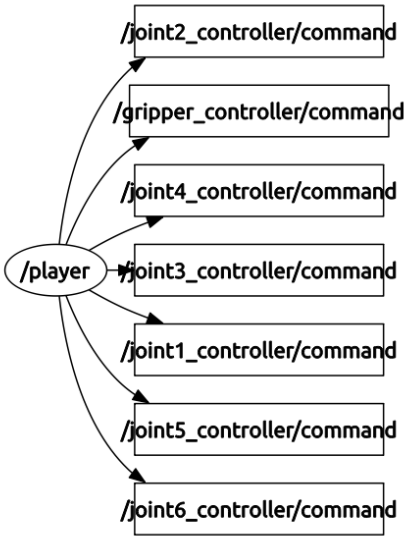

Fig 6.12 ROS Topic Graph of Player

6.5.3 Testing Arm Manipulation

The experiment is conducted using Cool Arm 6D 400. The arm was moved manually in different speed and motion was recorded. And recorded motion was played back in different speed and found out the accuracy of this module.

Table 6.2 Results of Arm Manipulation Testing

Recording Speed	Play backing Speed	Results
Slow	-1X	✘
	1X	✓
	2X	✓ ✓ ✓
Medium	-1X	✓
	1X	✓ ✓
	2X	✓
Fast	-1X	✓ ✓
	1X	✓
	2X	✘

The results are tabulated and shown in the table 6.2.

From the results it was clear that recording the motion of the arm in slow speed and while playing in fast mode gives the accurate play back motion.

6.4 SPEECH SYNTHESIS MODULE

Speech synthesis module is responsible for generating the speech to communicate with customer

The main processes involved in this module are summarized below:

1. Read the text based input for which a speech signal has to be generated.
2. Normalize input text.
3. Parse this text into words.
4. Parse these words into phonemes (text to speech).
5. For each word, process as follows:
   I. Get length of word.
   II. Generate signal for the servo of lip movement

III. Get index of next word
IV. Calculate the wait time (depends on length of the word).
V. Apply the wait time to synchronize the servo motor movement for each word.

The fourth step involves the conversion of the text into speech. For this, the text is sent to the sound_play, it provides a ROS node that translates commands on a ROS topic (robotsound) into sounds.

Then the module also generates the control signal for the servo motor to have a lip movement while SAYABOT speaks. This movement of servo motor is synchronized with the each word of the sentence that robot speaks.

The Ros topic message passing in this module is shown in the Fig 6.13.

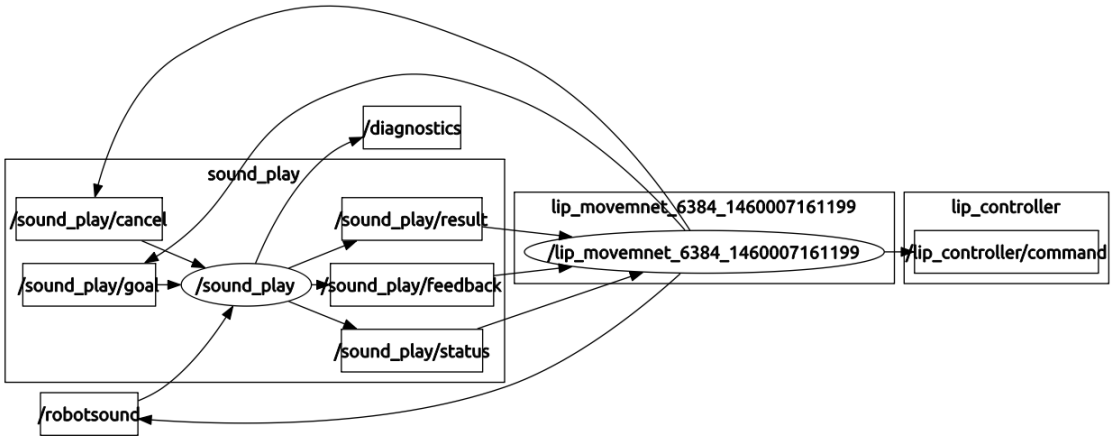

Fig 6.13 ROS Topic Graph of Speech Synthesis Module

6.4.1 Testing Speech Synthesis Module

For testing the speech synthesis module, the sentences with different amount of words are given as the text inputs to the module then the speech output and the lip movement were observed and tabulated.

Table 6.3 Results of Speech Synthesis Module Testing

Sentence Length	Speech Out Delay (seconds)	Delay in Servo Motor Sync (seconds)
<5 words	-	<2
<10 words	1 - 2	2 - 3
<15 words	2 - 5	3 - 5
>15 words	>5	4 - 5

The table 6.3 shows the observation of the delays in speech output and in servo motor sync for sentences with different number of words.

With the observed data from the testing of speech synthesis module, it was decided to use less than 10 words in each sentence to have elegant speech and lip movement which will give a feel of robot speaking.

6.5 Testing of Module Integration

The above three modules were integrated and were tested to find the efficiency of the system. To control each modules function and have it as a system, we used State machine libraries in ROS i.e. SMACH. Using SMACH, an action client state for each module with appropriate outcomes was created, so that it can be mapped with other modules to integrate.

Below Fig 6.14 shows the Integration of all components required for three modules for testing to make test model for the testing.

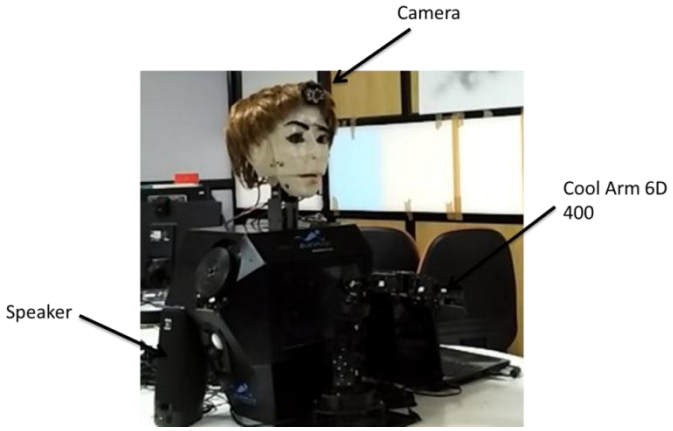

Fig 6.14 Test Model

Face detection of the robot will be working, once it detects the customer face then state is transferred to Arm Manipulation State in which both arm manipulation module and speech synthesis modules are concatenated for the user interface. After finishing the Arm Manipulation State the control again passes to the Face Detection State. Fig 6.15 shows the state diagram of integrated modules.

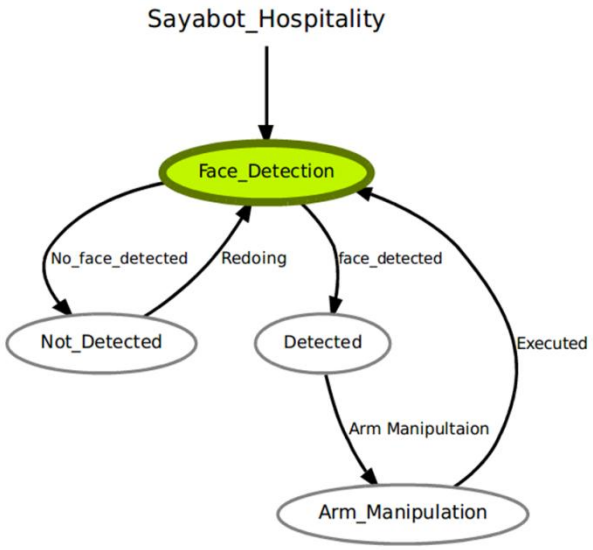

Fig 6.15 State Diagram of Module Integration

Results:

Through this testing the performance of the system which includes Perception, Arm Manipulation and Speech Synthesis Module was evaluated. During the test, each person was sent to stand before the robot and wait for its response. As soon as a person stood before the robot, robot detected the face and started to track it. Then the Arm Manipulation State started where robot greeted the person through voice and simultaneously moved the arm to show the greeting gesture to the person.

Once Arm Manipulation State was over, robot again goes to the Face Detection State very soon because of this, it detected a same person who stood before the robot and continued the sequence one more time. This was the drawback we found on the robot from this test. It was rectified by adding some delay time after the Arm Manipulation State, so that there will be time for the person to leave before robot goes back to its Face Detection State.

6.6 NAVIGATION

The navigation subsystem is responsible for system's high level motion planning. It coordinates with other subsystem in order to achieve the overall system objective. Once the user gives the destination in the user interface subsystem, the system starts to plan its path to reach the destination. The flow of information inside the subsystem is described in Fig 6.16.

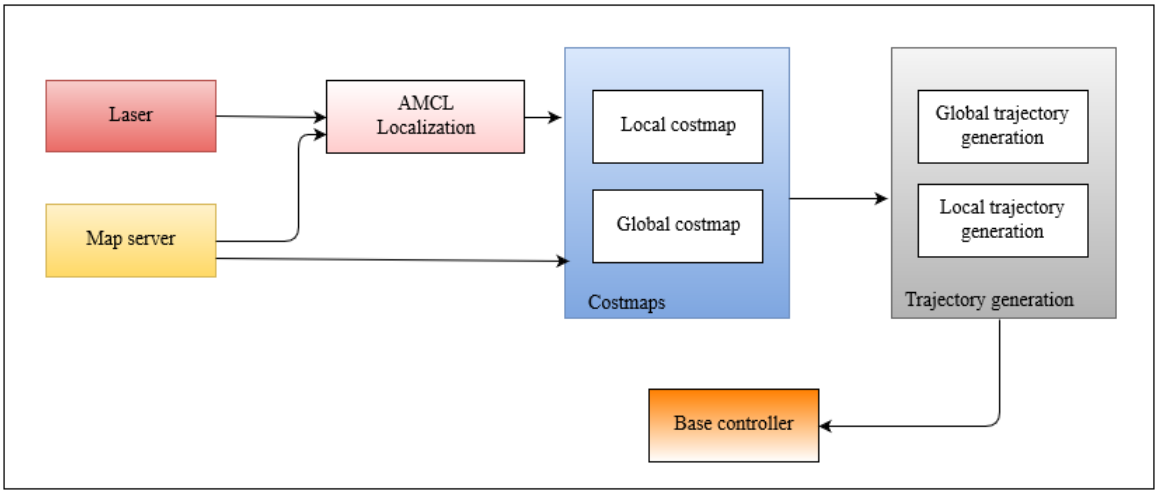

Fig 6.16 Flow of data

Main components of navigation stack:

1. Mapping and Localization
2. Path planning
   - Global path planning
   - Local path planning
3. Obstacle avoidance
   -Static obstacle avoidance
   -Dynamic obstacle avoidance

6.6.1 Mapping and Localization:

Localization tells the robot where it is in relation to the environment. Mobile robot localization is the problem of determining the pose of the robot with respect to the map of environment. It is also called as pose estimation. For proper localization, data are generally fused from different sensors like laser, Inertial Measuring Unit (IMU), wheel encoders.

Generally maps are described in global coordinate frame, which are independent of robot pose. Localization establishes a correspondence between map coordinate frame and robot local coordinate frame.

The data from laser was used for localization. The Adaptive Monte Carlo Localization (AMCL) package in ROS was implemented for localization. In order to use this package, the position of laser should be fixed with respect to base. It uses a particle filter to track the pose of a robot against a known map. So, the map of the environment should be passed as a parameter to the AMCL node.

In order to test the robot, an arena was built .The dimensions of the arena is 3.7m x 1.7m. The map of the arena was generated by recording the scan data and transfer functions from laser and encoders respectively. The flow of information inside the Localization and Mapping subsystem is described in Fig 6.17.

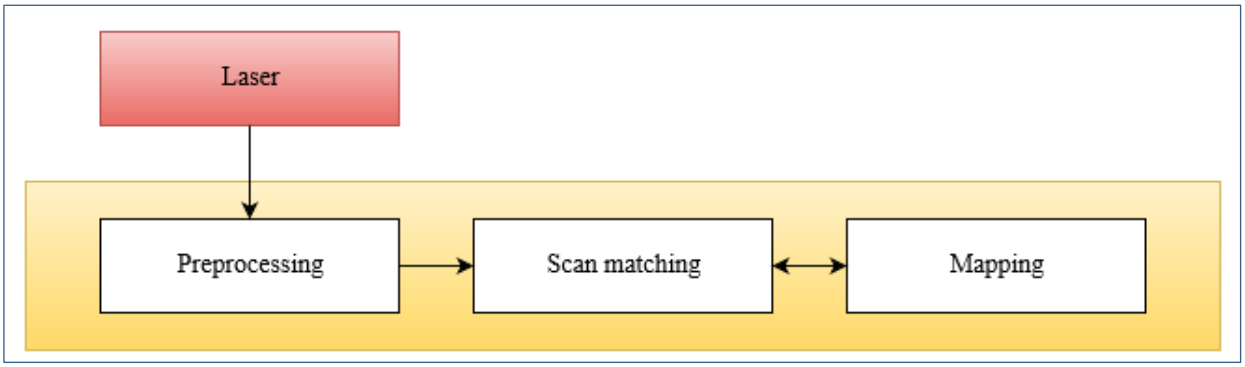

Fig 6.17 Localization and Mapping subsystem

6.6.2 Path planning:

Path planning generates the trajectory for the motion of the robot by taking account of map borders, dynamic and static obstacles. It contains costmaps and planners for generating the trajectory of the robot

Costmaps:

The costmaps stores the information about the environment in an occupancy grid. Costmaps are of two types: Local and global cost map. Global costmaps are used to create long term plans over the environment whereas the local costmaps are used to clear obstacles in map and for local planning. They are initialized through map server and uses sensor data and information from the static map to store and update information about obstacles in the world. Each bit of functionality exists in a layer. For instance, the static map is one layer, and the obstacles are another layer.

Each cell in the map can either be free or occupied but the cost of each cell varies. Based on the sensor data the cells are either marked (insert obstacle information into costmap) or cleared (remove obstacle from costmap). Robots closeness to the obstacle is controlled inflation radius parameter.

Inflation is the process of propagating cost values out from occupied cells that decrease with distance. The values are classified into 5 types based on the cost of the cell. The Fig 6.18 shows the various costs occur through inflation radius graph.

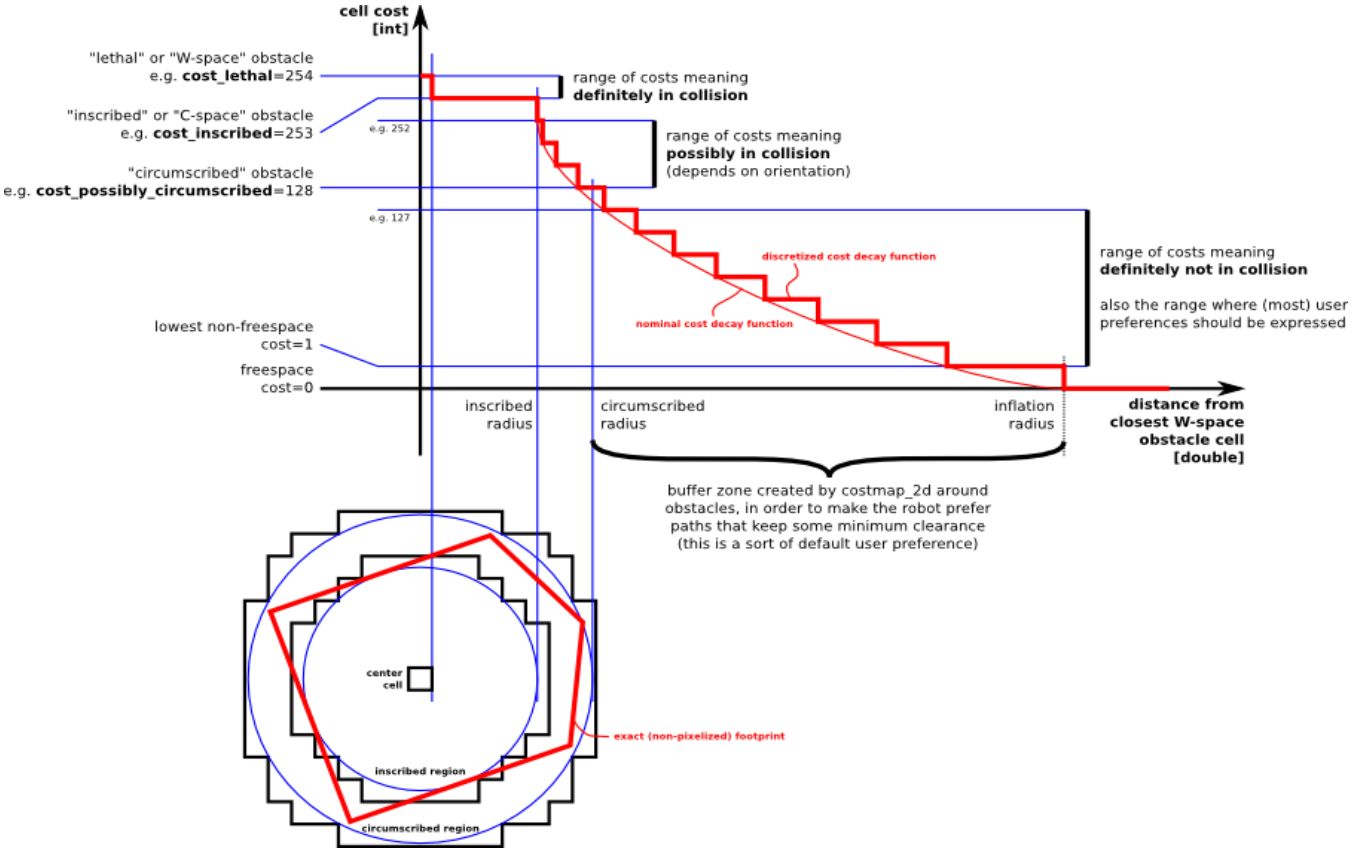

Fig 6.18 Inflation Radius Graph

The different costs related to the inflation are explained below:

Lethal cost	-	No free space. Collision is obvious.
Inscribed cost	-	Partial free space .So, the robot possibly in collision.
Circumscribed cost	-	Collision might occur based on orientation of the robot.
Free space cost	-	No collision and is free to move.
Unknown cost	-	No information about the cell.

Global Path Planner:

The global path planner generates a high-level plan for the navigation stack to follow. The global planner is fed with the obstacle and cost information contained in the global costmap, information from the robot's localization system, and a goal in the world .The goal position is given through the user interface. From this information the global planner creates a plan for the robot to follow to reach the goal location. It creates a series of waypoints for the local planner to achieve.

Local Path Planner:

The local planner is responsible for generating velocity commands for the mobile base that will safely move the robot towards a goal. The global planner passes the plan to local planner and the local planner attempts to follow it as closely as possible while taking into account the kinematics and dynamics of the robot as well as the obstacle information stored in the local costmap. In addition to the global costmap, a local costmap is also used because it takes account of the local obstacles which were not present in the global map (static) when it was created.

Move Base:

The move base node links together a global and local planner to accomplish its global navigation task. The move base supports any planner which adheres the requirements of navigation core packages. It provides the ROS interface for configuring, running, and interfacing with the navigation stack.

The move base provides certain nodes which are optional, depending on our algorithm and some nodes are platform specific which can be chosen based on the sensor and actuators which are used.

Fig 6.19 shows an overview of this move base configuration with navigation stack.

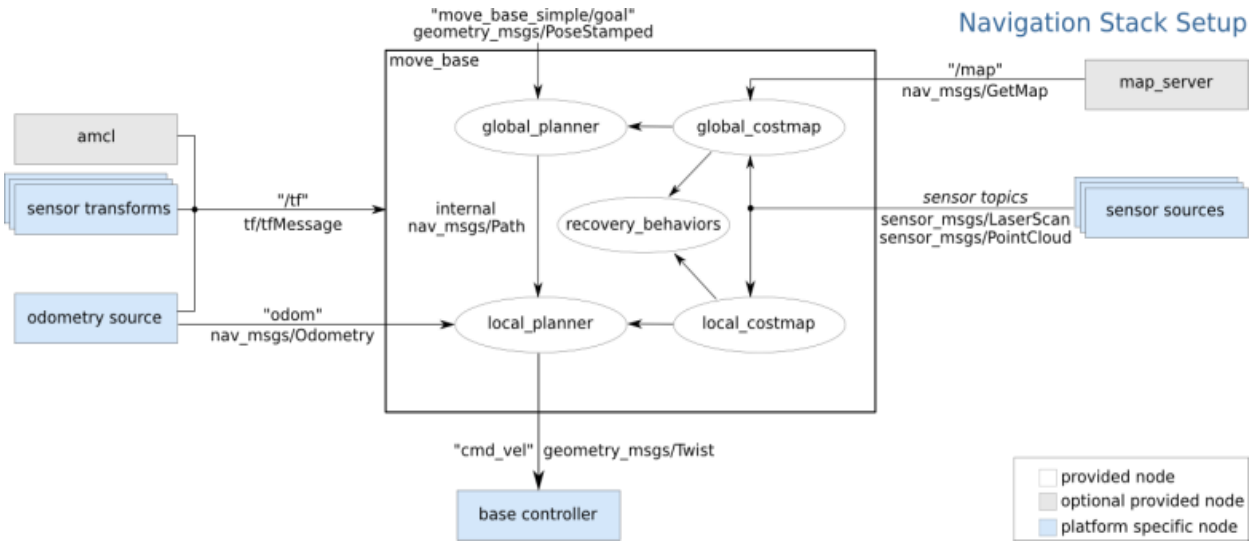

Fig 6.19 Move Base Configuration

Fig 6.20 shows the complete model of the system which was developed.

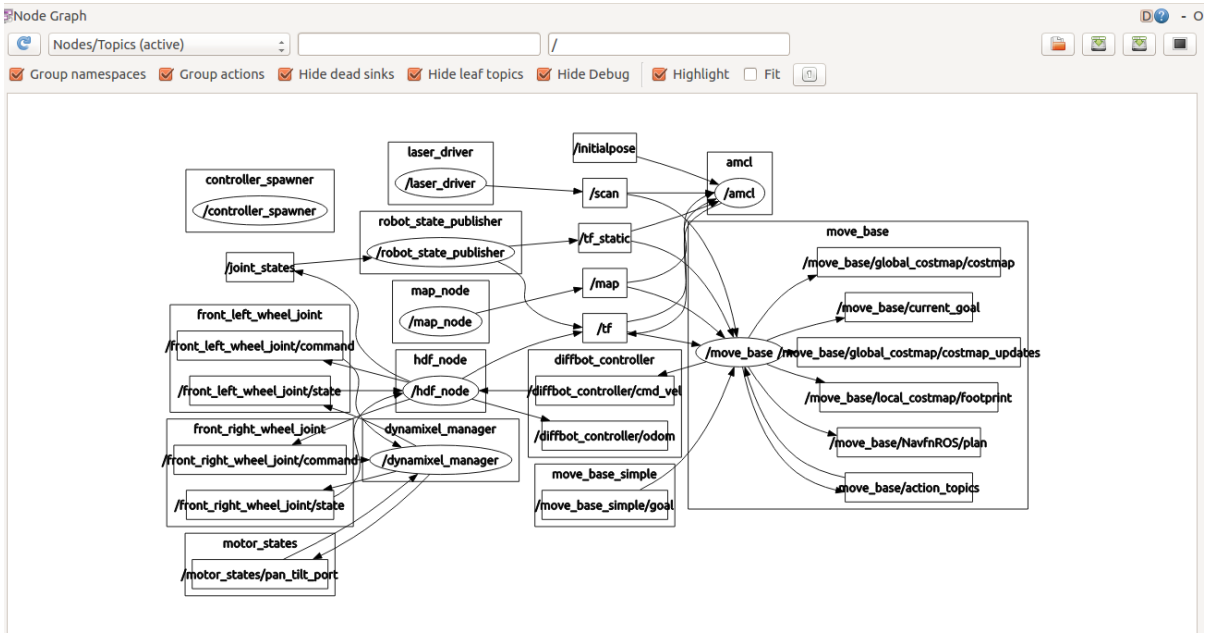

Fig 6.20 Graph of Nodes

6.6.3 Modeling, Analysis and Testing:

The first version of sayabot's navigation was successfully completed and tested using temporary base and test arena. The following components were integrated and tested:

1. Odometry
2. 2D laser scan based mapping
3. Trajectory planner
4. Obstacle avoidance

Test for Odometry:

In order to estimate the longitudinal and latitude accuracy in motion, the robot was commanded to go straight at a velocity of .02m/s for a distance of 3m. The path traced by the robot and the actual path are shown in the Fig 6.21.

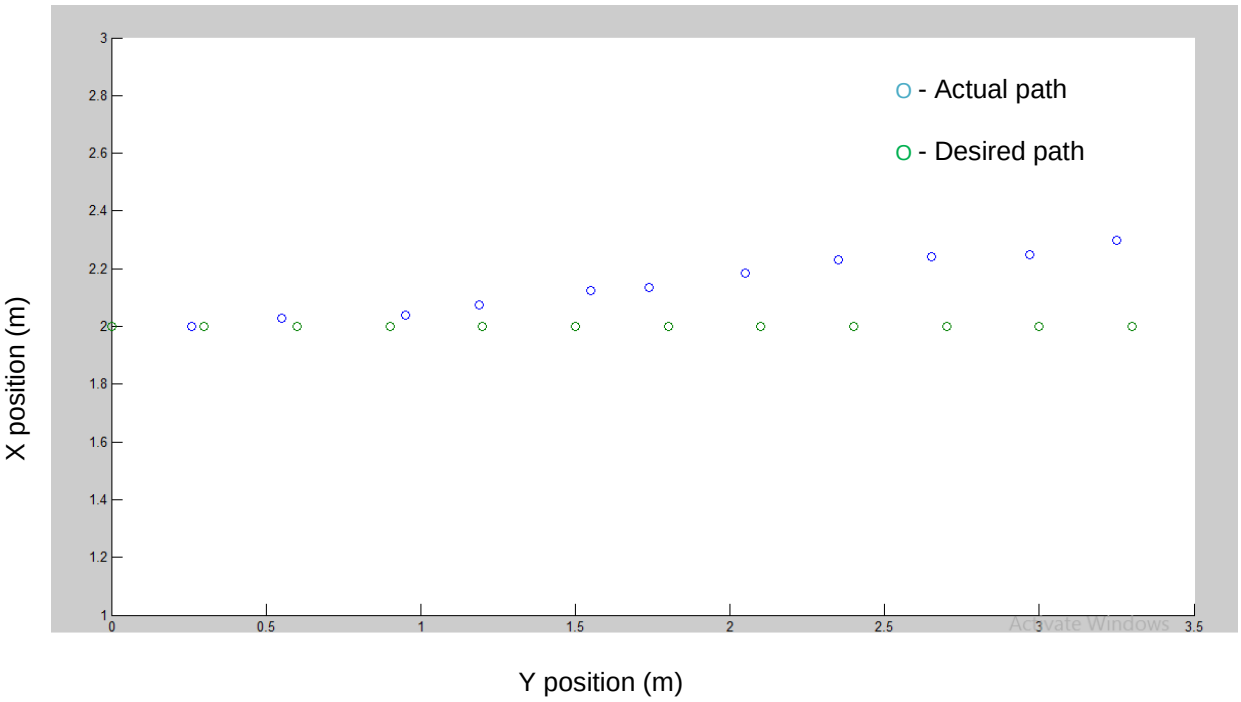

Fig 6.21 Odometry Error Estimation

Finally after a run of 3m , the accumulated error was 30 cm in x direction and 10 cm in y direction.To find out the reason for this error, this experiment was carried out a number of times. Each time the error was found to be same and repititive. So it was concluded that this error were of the type systemic error and not an random error. Later the error was found to be occuring because of the castor wheels and slight misalignment between right and left wheels.

Trajectory Planner:

The base local planner provides a controller and a path planner. Once the start and goal position is received , the planner creates a trajectory for the robot to reach the goal position in map.The controller maintains a cost function around the robot ‘s grid cells and determines the dx , dy, dtheta velocities to send to the robot. It has two planners namley:Trajectory Rollout and Dynamic window(DWA) approach.

Comparison of Trajectory Rollout and DWA Planner:

The basic idea behind both the algorithms are same. Robot‘s control space (dx,dy,dtheta) is sampled discretely and for each sampled velocity, forward simulation is performed from the current state to predict the cost of the trajectory.The cost is based on proximity to obstacles,proximity to goal,proximity to global path and speed .The trajectories colliding with obstacles are discarded and finally the highest scoring trajectory is chosen and the corresponding velocities are given to the robot.

The difference between the algorithm lies in the control space of the robot. Trajectory Rollout samples from the set of achievable velocities over the entire forward simulation period given the acceleration limits of the robot, while DWA samples from the set of achievable velocities for just one simulation step given the acceleration limits of the robot. The Fig 6.22 and Fig 6.23 shows the path planned and the actual path traced by the robot respectively.

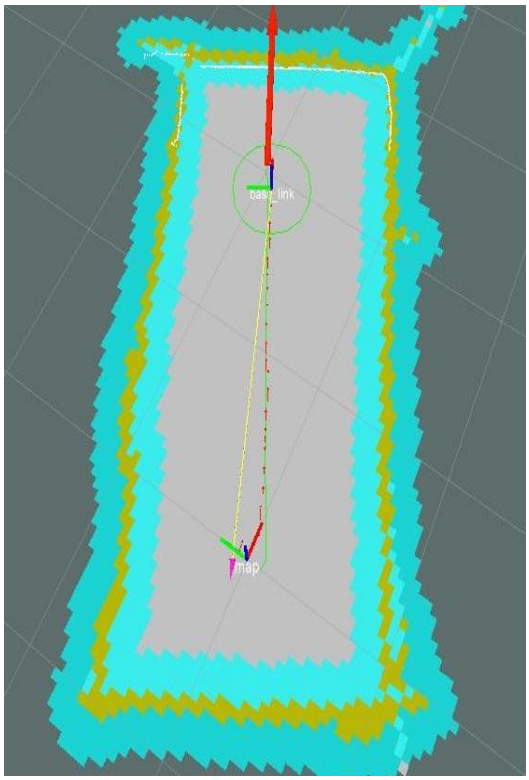

Fig 6.22 DWA Planner

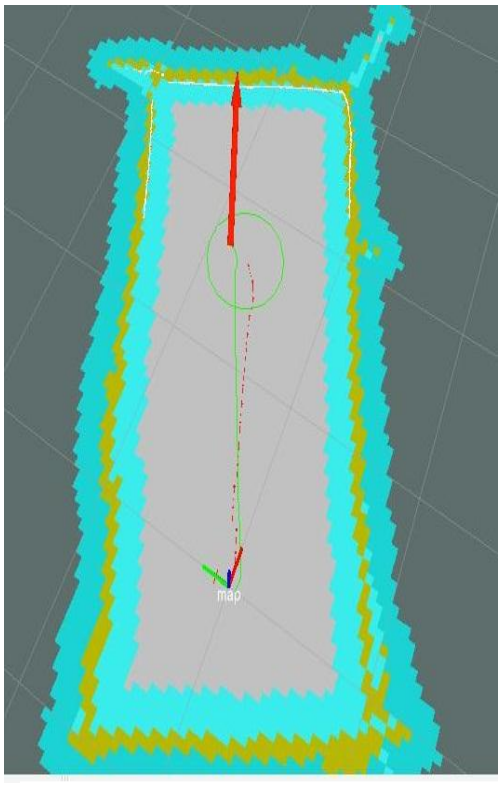

Fig 6.23 Global Planner

The green line shows the path planned by the planner and the red arrows shows the path traced by the real robot. From a series of trials , we found that the path traced by the real robot is more exact in the case of dwa planner when compared to global planner . This is because the sample space took by the dwa planner is smaller for each simulation step and so the resultant path is more exactly achievable in real time. So dwa planner was implemented in this project.

Parameter Tuning:

Initially ,all the parameters were set to default and then a set of related parameters are taken and tuned accordingly.

Robot configuration parameters:

The accleration parameters were set to default values provided by the package.To set the minimum and maximum velocity, the robot was commanded to move a distance of 5m in a series of trials while varying the velocities from 0.05m/s to 1m/s.

Table 6.4 Results of Velocity Limit Test

Trial no	Velocity(m/s)	Observation
1	0.05	No movement because of self weight and friction.
2	0.1	Slight movement but was not accurate.
3	0.15	Proper movement
4	0.2	Proper movement
5	0.25	Proper movement
6	0.3	Proper movement
7	0.35	Proper movement
8	0.4	Proper movement
9	0.45	During breaking , slight drift in wheels
10	0.5	Drift increased while stopping

The table 6.4 shows the observations from the experiment.

From the above observations, the minimum velocity was set as 0.15m/s and maximum velocity as 0.4m/s and similar set of experiments were done for rotation and the minimum in place rotational velocity is set as 0.5m/s whereas theta parameters are set to default. Escape velocities were set to positive to prevent the robot to move in reverse direction when the robot is struck or very close to obstacle.

Goal Tolerance Parameters:

The tolerance in theta were set to default while the tolerence in xy were varied from 0.1m to 0.3m based on the requirement.

Table 6.5 Results of Goal Tolerance Parameter Test

Goal tolerance parameter(m)	Trial number	Actual distance(m)	Observation
0.3	1	0.28	The robot reached the goal with the given tolerance easily.
	2	0.24	
	3	0.27	
0.2	1	0.20	The robot took some time to reach the goal as it adjusted itself to achieve the tolerance.
	2	0.17	
	3	0.175	
0.1	1	0.097	The robot was rotating continuously when it came near the goal, to achieve the higher accuracy but took long time to settle
	2	0.12	
	3	0.11	

The observations were made and tabulate in the table 6.5.

Based on this experiment ,the tolerance parameter in xy was set to 0.2.

Forward Simulation Parameters:

The simulation time was set as 2.0s and simulation granularity was set as 0.025. When these parameter values were increased, the processing time increased and it resulted in delayed response. When these parameter values were decreased, the path generated by the planner was not optimal. Similarly, to decrease the process time, vx and vtheta samples were reduced to 4 and 10 from the default values respectively.

Other Parameters:

The trajectory scoring and oscillation parameters were set to the default values and they were left unchanged.

6.6.4 Real Time Testing:

Test Arena:

To test the performance of navigation module, an arena of dimensions 3.7m x 1.7m was built.

Fig 6.24 shows the actual arena built for the testing. Fig 6.25 shows the map generated by the robot using laser scan and odometry.

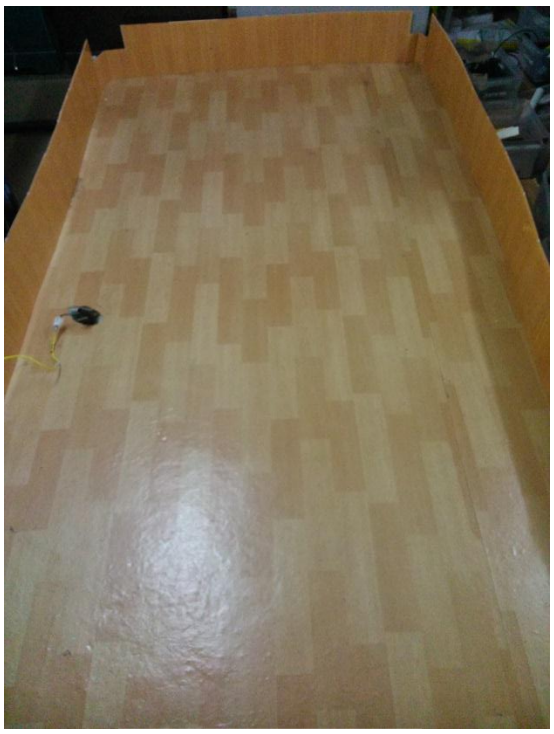

Fig 6.24 Test Arena

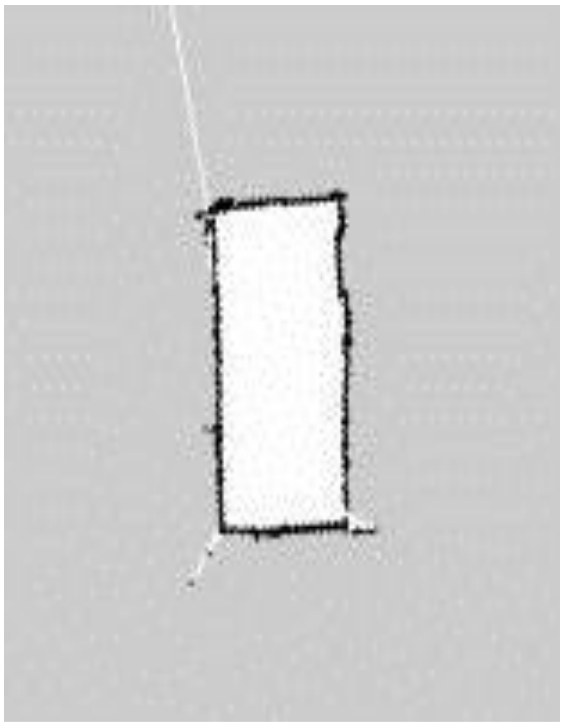

Fig 6.25 Generated Map of Test Arena

The robot was controlled through joystick and was commanded to move all over the arena .The data were recorded using rosbag command and sent to the map server node to create a map. The resolution of the map was set to 0.05m per pixel.

Test 1:

The arena was clear and there were no obstacles .The robot was commanded through Rviz to move around the arena. The test includes

i) Straight motion from end to end.

ii) Diagonal motion

Fig 6.26 shows the test arena with no obstacles.

Fig 6.26 Test Arena with No Obstacles

Observation:

The response was immediate and the motion was smooth. The localization was proper. The robot was successful in reaching all the goal positions within the tolerance limit. On an average, the robot took 9 seconds to reach from one end to the other. But when the goal position were given in the corner of the arena , it took some time to settle as the arena borders were close to the robot once the robot reaches the corner. So, it took 12 seconds to settle in the final goal position.

Test 2:

Static obstacles were placed in the arena in various locations and the robot was commanded to different goal position. The test includes setting the following positions and observing the robot motion.

i) Goal position very close to obstacles

ii) Goal position away from obstacles

iii) Goal position

Fig 6.27 shows the test arena with the obstacles in it.

Fig 6.27 Arena with Obstacles

Observation:

The robot was successful in reaching the goal positions which were far from the obstacles. When the goal position was close proximity to obstacle, the robot took a longer path to avoid the obstacle. Though there was enough space for the robot to move and reach the goal the robot sometimes could not find a valid path and aborted its motion. The robot was not able to detect obstacles whose height was very less than the distance between the laser and ground. So it hit those obstacles while reaching its goal position

Test 3:

Dynamic obstacles were introduced into the arena while the robot was moving and the robot motion was observed. The test includes

i) Introducing obstacles after the path to goal is planned.

ii) Moving people moving inside the arena

Observation:

The robot was able to detect the dynamic obstacle and re-plan its path efficiently. The robot was unable to stop immediately when the obstacle was introduced suddenly within a range of 0.1m.When the obstacle was at a distance of greater than 0.1 m, it was able to stop and re-plan its path and was able to reach the goal. Similarly when the obstacles, were introduced on the sides perpendicular to the bot, it was unable to detect the obstacle because the obstacle is out of range of laser and so while turning, it hit the obstacle

sometimes. When people were moving inside the arena, the robot was able to detect them as obstacles and re plan its path. But it took longer time to reach the goal since the planner was continuously replanning its path.

Results and validation:

From the above conducted tests, it was clear that the robot was able to avoid static and dynamic obstacles and also reach the goal position with a tolerance of 0.02m. The drawbacks are listed below:

i) Obstacles of very less height were not detected.
ii) Obstacles like tables, chairs should not be present in arena as the robot will hit them because they have empty space in the middle but the robot motion is not possible through the space since in z-direction they will be hit.
iii) When the arena has more number of obstacles, the time taken to reach the goal takes longer than the normal time.
iv) The castor wheels which we used decreased the accuracy of the final pose of the robot.
v) Dynamic obstacles at very close proximity to the robot were hit by the robot since there was drift in the wheels of the robot before it stopped.

CHAPTER 7

PROJECT MANAGEMENT

To manage time efficiently and to finish the project in time, teamwork software has been used to allocate tasks accordingly. Through this tool, we were able to keep track the progress of the project entirely.

7.1 TASK ASSIGNMENT

7.1.1 Person Detection and Tracking

Table 7.1 Task Assignment for Person Detection and Tracking

S.no	Task	Start date	Date due	Assigned to	Assigned by	Status
1	Research on face detection algorithms	08 Jan (2016)	11 Jan (2016)	Niranjan J. Gokul N.	Ravi K.	Completed 12 Jan (2016)
2	Identifying the sensor to be used for detecting a person	12 Jan (2016)	13 Jan (2016)	Niranjan J. Gokul N.	Ravi K.	Completed 13 Jan (2016)
3	Identifying the position of the sensor on the robot	14 Jan (2016)	16 Jan (2016)	Niranjan J. Gokul N.	Ravi K.	Completed 15 Jan (2016)
4	Customization of face detection code	17 Jan (2016)	25 Jan (2016)	Niranjan J. Gokul N.	Ravi K.	Completed 25 Jan (2016)
5	Testing of the face detection code	26 Jan (2016)	27 Jan (2016)	Niranjan J. Gokul N.	Ravi K.	Completed 27 Jan (2016)
6	Research on face tracking algorithms	28 Jan (2016)	29 Jan (2016)	Niranjan J, Gokul N.	Ravi K.	Completed 29 Jan (2016)
7	Customization of face tracking algorithm specific to project requirements	30 Jan (2016)	01 Feb (2016)	Niranjan J. Gokul N.	Ravi K.	Completed 02 Feb (2016
8	Testing the algorithm in the real time environment	01 Jan (2016)	03 Feb (2016)	Niranjan J. Gokul N.	Ravi K.	Completed 04 Feb (2016)
9	Documenting the code	04 Feb (2016)	05 Feb (2016)	Niranjan J. Gokul N.	Ravi K.	Completed 05 Feb (2016)

The table 7.1 shows the task assignment for person detection and tracking.

7.1.2 Arm Manipulation

Table 7.2 Task Assignment for Arm Manipulation

S.No.	Task	Start Date	Date Due	Assigned To	Assigned By	Status
1	Upgrade the Ros arm manipulation package and remove the errors	01 Feb (2016)	02 Feb (2016)	Niranjan J.	Ravi K.	Completed 02 Feb (2016)
2	Identify different ways of smoothing of the arm movement if its already not	03 Feb (2016)	04 Feb (2016)	Niranjan J.	Ravi K.	Completed 04 Feb (2016)
3	Check how good the movement of the arm is when the rosbag is played Back.	04 Feb (2016)	05 Feb (2016)	Niranjan J.	Ravi K.	Completed 05 Feb (2016)
4	Write a code for recording the topics published while the arm is in torque using rosbag.	08 Feb (2016)	09 Feb (2016)	Niranjan J.	Ravi K.	Completed 10 Feb (2016)
5	Record the different movements of the arm required to be done by the robot	10 Feb (2016)	10 Feb (2016)	Niranjan J.	Ravi K.	Completed 11 Feb (2016)
6	Testing of the recording with the arm hardware	11 Feb (2016)	12 Feb (2016)	Niranjan J.	Ravi K.	Completed 14 Feb (2016)
7	Update the code in HDFC repository.	16 Feb (2016)	17 Feb (2016)	Niranjan J.	Ravi K.	Completed 17 Feb (2016)

The table 7.2 shows the task assignment for arm manipulation.

7.1.3 Autonomous Navigation for the robot

Table 7.3 Task Assignment for Autonomous Navigation for the robot

S.No	Task	Start Date	Date Due	Assigned To	Assigned By	Status
1	Implement a node for writing and reading values between differential and ros dynamixel controller	04 Feb (2016)	07 Feb (2016)	Gokul N.	Ravi K.	Completed 08 Feb (2016)
2	Customize the steered wheelbase controller code(with ros control) to differential drive.	08 Feb (2016)	10 Feb (2016)	Gokul N.	Ravi K.	Completed 10 Feb (2016)
3	Test the differential drive and ros control with two free servos through teleoperation.	11 Feb (2016)	15 Feb (2016)	Gokul N.	Ravi K.	Completed 14 Feb (2016)
4	Customize the navigation stack to work with differential drive	16 Feb (2016)	23 Feb (2016)	Gokul N.	Ravi K.	Completed 23 Feb (2016)
5	Test the robot base with the customized navigation stack with the default tuned parameters.	24 Feb (2016)	27 Feb (2016)	Gokul N.	Ravi K.	Completed 28 Feb (2016)
6	Create a differential drive controller for robot	29 Feb (2016)	3 Mar (2016)	Gokul N.	Ravi K.	Completed 4 Mar (2016)
7	Tune the parameters to have the robot follow the exact path which is planned by the navigation stack.	4 Mar (2016)	9 Mar (2016)	Gokul N.	Ravi K.	Completed 9 Mar (2016)
8	Research on dynamic obstacle handling within navigation stack.	10 Mar (2016)	11 Mar (2016)	Gokul N.	Ravi K.	Completed 11 Mar (2016)
9	Customize the code so that the robot can handle dynamic obstacles.	12 Mar (2016)	17 Mar (2016)	Gokul N.	Ravi K.	Completed 20 Mar (2016)
10	Test the code on the robot base with random obstacles on its way.	18 Mar (2016)	25 Mar (2016)	Gokul N.	Ravi K.	Completed 25 Mar (2016)

The table 7.3 shows the task assignment for autonomous navigation of the robot.

7.1.4 Lip synchronization with voice

Table 7.4 Task Assignment for Lip synchronization with voice

S.No	Task	Start Date	Date Due	Assigned To	Assigned By	Status
1	Identify the servo which would be used for the lip synchronization	07 Mar (2016)	07 Mar (2016)	Niranjan J.	Ravi K.	Completed 07 Mar (2016)
2	Explore the code from ISRA to work in ROS.	07 Mar (2016)	08 Mar (2016)	Niranjan J.	Ravi K.	Completed 08 Mar (2016)
3	Create Text to Speech rospy node	08 Mar (2016)	09 Mar (2016)	Niranjan J.	Ravi K.	Completed 09 Mar (2016)
4	Test the Text to Speech node	09 Mar (2016)	10 Mar (2016)	Niranjan J.	Ravi K.	Completed 10 Mar (2016)
5	Creating a control of servo motor for lip movement	11 Mar (2016)	15 Mar (2016)	Niranjan J.	Ravi K.	Completed 17 Mar (2016)
6	Synchronizing servo movement with speech using threading concept	16 Mar (2016)	19 Mar (2016)	Niranjan J.	Ravi K.	Completed 20 Mar (2016)
7	Testing and rectifying the synchronization of lip movement	20 Mar (2016)	25 Mar (2016)	Niranjan J.	Ravi K.	Completed 26 Mar (2016)

The table 7.4 shows the task assignment for lip synchronization with voice.

7.1.5 SMACH for the robot

Table 7.5 Task Assignment for SMACH for the robot

S.No	Task	Start Date	Date Due	Assigned To	Assigned By	Status
1	Exploring SMACH	17 Feb (2016)	18 Feb (2016)	Niranjan J.	Ravi K.	Completed 19 Feb (2016)
2	Testing SMACH	19 Feb (2016)	20 Feb (2016)	Niranjan J.	Ravi K.	Completed 21 Feb (2016)
3	Creating flow of the robot states	21 Feb (2016)	23 Feb (2016)	Niranjan J.	Ravi K.	Completed 24 Feb (2016)
4	Defining SMACH states	24 Feb (2016)	26 Feb (2016)	Niranjan J.	Ravi K.	Completed 27 Feb (2016)
5	Planning SMACH for face detection and arm manipulation	26 Feb (2016)	27 Feb (2016)	Niranjan J.	Ravi K.	Completed 28 Feb (2016)
6	Planning SMACH states for the robot	28 Feb (2016)	29 Feb (2016)	Niranjan J.	Ravi K.	Completed 29 Feb (2016)

The table 7.5 shows the task assignment for SMACH for the robot.

7.2 TIMELINE

Fig 7.1 shows the Gantt chart of the project timeline.

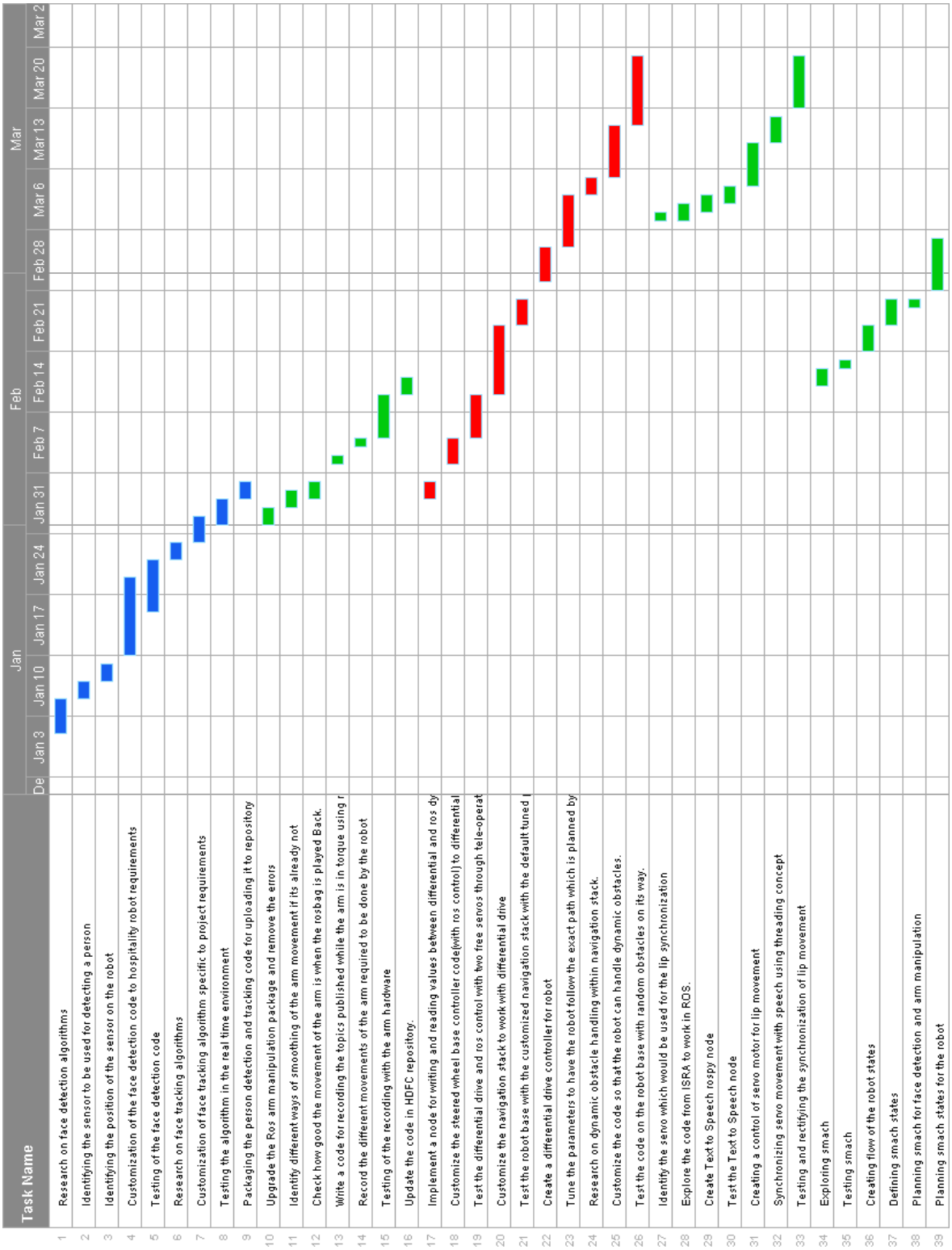

Fig 7.1 Project Gantt-Chart

CHAPTER 8

CONCLUSION

SAYABOT with limited functionalities was tested in a bank like environment. And from the people response, it was clear that people like to have robot in those kind of environment. Initially people were excited to interact with the robot but the response from the robot was slow, which annoyed a part of people. Several tests were conducted and the parameters for the operation of the robot were tuned for better functioning of the robot. During the process of customizing the software for the robot , several unknown errors occurred which were unexpected and that delayed the whole process. The objectives of the project were not achieved completely since certain modules failed to satisfy the requirements as expected. These modules must be reworked in the future to attain the requirements and better performance.

8.1 RESULT

Detecting human faces in an environment with lot of peoples moving around was less accurate than expected. Speech synthesis performed as expected but people expected to have intelligent responses rather than a recorded speech. Navigation module was successful in achieving the requirements but it could not achieve higher velocities because of the drift in wheels. So it took longer time to reach the goal position. The integration of all the modules was tedious and it failed several times because of the failure of sub modules. SAYABOT will be able to do the functionalities as mentioned in the requirements but fails when the actions are performed repeatedly because of overloading of servos and calibration errors.

8.2 FUTURE WORK

The overall system worked well but it was not as robust as expected. Each of the modules has improvements that can be made.

Perception:

Face detection didn't work proper when there were more than two people and the users were moving fast. This must be addressed and it must make such that it can detect people and track one of them more accurately.

Arm manipulation:

The movement of the arm is not smooth when it does actions repeatedly and positioning accuracy is less. Changes should be made in the record and play back function to get smoother motion and position accuracy. Intelligent motion planning should also be done to avoid arm colliding with the user or an obstacle.

Speech synthesis:

The algorithm fails when sentences of more than 15 words are given .This must be addressed to make the algorithm work for sentences with more than 15 words. Speech recognition should be developed over this module to make the user convenient in interacting with the robot.

Navigation:

The biggest problem with our navigation is the inability to respond to obstacles of very less height and obstacles like tables which have space in the middle but they are actually a potential obstacle. The tolerance for the goal position should be improved from 0.2m to 0.05m spending more time tuning parameters in the ROS navigation

BIBLIOGRAPHY

1. Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng, “ROS: an open-source Robot Operating System”, Retrieved from https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf
2. PAUL VIOLA., and MICHAEL J. JONES, “Robust Real-Time Face Detection,” International Journal of Computer Vision, Vol. 57, No.2, 2004, pp. 137-154.
3. J. Bohren and S. Cousins, “The SMACH high-level executive [ros news],” Robotics Automation Magazine, IEEE, vol. 17, no. 4, Dec 2010, pp. 18–20.
4. ROS (Robot Operating System). http://www.ros.org/
5. Opencv. http://opencv.willowgarage.com/.
6. Austin1 September 2015, “Smooth Face Tracking with OpenCV”, [Web log post]. Retrieved from http://synaptitude.me/blog/smooth-face-tracking-using-opencv/
7. Alexander Tomovie , “Path Planning Algorithms For the Robot Operating System”,retrived from https://www.micsymposium.org
8. Dieter Fox, Wolfram Burgard, Sebastin Thrun , “The Dynamic Window Approach to Collision Avoidance” retrieved from https://www.ri.cmu.edu
9. Johann Borenstein, Liqiang Feng , “ UMBmark:A benchmahrk Test for Measuring Odometry Errors in Mobile Robots” ,SPIE Conference on Mobile Robotics ,retrived from https://www.johnloomis.org
10. Johann Borenstein , Liqiang Feng, “Measurement and correction of systematic Odometry Errors in Mobile Robots” ,IEEE transaction on Robotics and Automation ,Vol.12, No.6 , 1996.

APPENDIX A

Main Page | Namespaces | Files

File List

current_position_publisher.py

Go to the documentation of this file.

```
1
2
3  import rospy
4  from math import fabs
5  from dynamixel_msgs.msg import JointState
6  from std_msgs.msg import Float64
7  import rosbag
8
9  joint1_pub = rospy.Publisher('/joint1_command', Float64, queue_size = 10)
10 joint2_pub = rospy.Publisher('/joint2_command', Float64, queue_size = 10)
11 joint3_pub = rospy.Publisher('/joint3_command', Float64, queue_size = 10)
12 joint4_pub = rospy.Publisher('/joint4_command', Float64, queue_size = 10)
13 joint5_pub = rospy.Publisher('/joint5_command', Float64, queue_size = 10)
14 joint6_pub = rospy.Publisher('/joint6_command', Float64, queue_size = 10)
15 gripper_pub = rospy.Publisher('/gripper_command', Float64, queue_size =
   10)
16
17
18 def callback1(data):
19         current_pos = Float64()
20         current_pos = data.current_pos
21         var = data.motor_ids
22         joint1_pub.publish(current_pos)
23
24
25 def callback2(data):
26         current_pos = Float64()
27         current_pos = data.current_pos
28         var = data.motor_ids
29         joint2_pub.publish(current_pos)
30
31 def callback3(data):
32         current_pos = Float64()
33         current_pos = data.current_pos
34         var = data.motor_ids
35         joint3_pub.publish(current_pos)
36
37 def callback4(data):
38         current_pos = Float64()
39         current_pos = data.current_pos
40         var = data.motor_ids
41         joint4_pub.publish(current_pos)
42
43 def callback5(data):
44         current_pos = Float64()
45         current_pos = data.current_pos
46         var = data.motor_ids
47         joint5_pub.publish(current_pos)
48
49 def callback6(data):
50         current_pos = Float64()
51         current_pos = data.current_pos
52         var = data.motor_ids
53         joint6_pub.publish(current_pos)
54
55 def callback7(data):
56         current_pos = Float64()
57         current_pos = data.current_pos
58         var = data.motor_ids
59         gripper_pub.publish(current_pos)
60
61
```

APPENDIX

```
62          '''
63          if var == 0:
64                  joint1_pub.publish(current_pos)
65          if var == 1:
66                  joint2_pub.publish(current_pos)
67          if var == 2:
68                  joint3_pub.publish(current_pos)
69          if var == 3:
70                  joint4_pub.publish(current_pos)
71          if var == 4:
72                  joint5_pub.publish(current_pos)
73          if var == 5:
74                  joint6_pub.publish(current_pos)
75          if var == 6:
76                  gripper_pub.publish(current_pos)
77      '''
78
79  def recorder():
80          rospy.init_node('recorder', anonymous=True)
81          rospy.Subscriber("/joint1_controller/state", JointState,
    callback1)
82          rospy.Subscriber("/joint2_controller/state", JointState,
    callback2)
83          rospy.Subscriber("/joint3_controller/state", JointState,
    callback3)
84          rospy.Subscriber("/joint4_controller/state", JointState,
    callback4)
85          rospy.Subscriber("/joint5_controller/state", JointState,
    callback5)
86          rospy.Subscriber("/joint6_controller/state", JointState,
    callback6)
87          rospy.Subscriber("/gripper_controller/state", JointState,
    callback7)
88
89  if __name__ == '__main__':
90          try:
91                  recorder()
92                  rospy.spin()
93          except rospy.ROSInterruptException:
94                  pass
95
```

APPENDIX B

Main Page | Classes | Files

File List | File Members

face_tracking.cpp

Go to the documentation of this file.

```
  1 #include <ros/ros.h>
  2 #include <ros/console.h>
  3 #include <iostream>
  4 #include <stdio.h>
  5 #include <std_msgs/Float64.h>
  6 #include <std_msgs/Bool.h>
  7
  8
  9 // opencv image processing libraries
 10 #include <opencv2/imgproc/imgproc.hpp>
 11 #include <opencv2/highgui/highgui.hpp>
 12 #include <opencv2/opencv.hpp>
 13 #include "opencv2/objdetect/objdetect.hpp"
 14
 15 // to facilitate the processing of the images in ROS
 16 #include <image_transport/image_transport.h>  // for publishing and
    subscribing to images in ROS
 17 #include <cv_bridge/cv_bridge.h>              // to convert between ROS
    and OpenCV Image formats
 18 #include <sensor_msgs/image_encodings.h>
 19
 20 static const std::string OPENCV_WINDOW = "Sayabot Face Tracking";
 21 static const int frame_centre = 210;
 22 int face_X_position;
 23
 24 static int limit = 0;
 25 static float data = 0;
 26 static int e = 0, el = 0;
 27 static float kp = 0.00055, kd = 0.0001;
 28 static float present_X = 0, prev_X = 0;
 29 static int count_X = 0;
 30
 31 int msleep(unsigned long milisec)
 32 {
 33   struct timespec req = {0};
 34   time_t sec = (int)(milisec / 1000);
 35   milisec = milisec - (sec * 1000);
 36   req.tv_sec = sec;
 37   req.tv_nsec = milisec * 1000000L;
 38   while (nanosleep(&req, &req) == -1)
 39     continue;
 40   return 1;
 41 }
 42
 43 class FaceDetectionTracking
 44 {
 45 public:
 46   ros::NodeHandle nh_;
 47   image_transport::ImageTransport it_;
 48   image_transport::Subscriber im_sub_;
 49   image_transport::Publisher im_pub_;
 50   ros::Publisher servo_pub = nh_.advertise<std_msgs::Float64>
    ("/pan_controller/command", 100); // Publisher for servo message
 51   ros::Publisher output_pub = nh_.advertise<std_msgs::Bool>
    ("/sayabot_hospitality_perception/face_detection/output",100);
    //Publishes face_detection output
 52   cv::CascadeClassifier face_cascade_;
 53
 54   cv::HOGDescriptor hog_;
 55
 56   FaceDetectionTracking() : it_(nh_)
 57   {
```

APPENDIX

```
58          // Path to the haarcascade_frontalface_alt.xml
59      cv::String face_cascade_name =
60  ("/home/niranjan/ros_workspaces/sayabot_ws/src/saya_hospitality_perceptio
    n/saya_hospitality_face_tracking/include/saya_hospitality_face_tracking/h
    aarcascade_frontalface_alt.xml");
61
62      // Load the hog descriptor
63      hog_.setSVMDetector(cv::HOGDescriptor::getDefaultPeopleDetector());
64
65      // Load face detector HAAR cascades
66      if (!face_cascade_.load(face_cascade_name))
67        ROS_ERROR("--(!)Error loading face detector cascade \n");
68
69      // Subscrive to input video feed and publish output video feed
70      im_sub_ = it_.subscribe("usb_cam/image_raw", 1,
    &FaceDetectionTracking::imageCallback, this);
71      im_pub_ =
    it_.advertise("/sayabot_hospitality_face_tracker/output_video", 1);
72
73      cv::namedWindow(OPENCV_WINDOW);
74    }
75
76    ~FaceDetectionTracking()
77    {
78      cv::destroyWindow(OPENCV_WINDOW);
79    }
80
81    void imageCallback(const sensor_msgs::ImageConstPtr& msg)
82    {
83      cv_bridge::CvImagePtr cv_ptr;     // Make image processable in ROS
84
85
86      std_msgs::Float64 servo_msg;      // Float64 message for
    dynamixel_servo/command topic
87
88      std_msgs::Bool output_msg;
89
90      try
91      {
92        cv_ptr = cv_bridge::toCvCopy(msg,
    sensor_msgs::image_encodings::BGR8);
93      }
94      catch (cv_bridge::Exception& e)
95      {
96        ROS_ERROR("cv_bridge exception: %s", e.what());
97        return;
98      }
99      cv::Mat im_bgr = cv_ptr->image;
100
101     std::vector<cv::Rect> detected_faces;
102     cv::Mat im_gray;
103     cv::cvtColor(im_bgr, im_gray, CV_BGR2GRAY); // converting RGB image
    to gray_scale
104     cv::equalizeHist(im_gray, im_gray);
105
106     // HAAR Face detector
107
108     face_cascade_.detectMultiScale(im_gray, detected_faces, 1.1, 3, 0 |
    CV_HAAR_SCALE_IMAGE, cv::Size(30, 30));
109
110     if ((detected_faces.size()) == 0)
111     {
112       ROS_INFO_STREAM("No person detected");
113       limit++;
114       output_msg.data = 0;
115       output_pub.publish(output_msg);
116       ROS_INFO_STREAM(limit);
117       if (limit >= 2)
118       {
119         data = 2.5;
120         ROS_INFO_STREAM("Data");
121         ROS_INFO_STREAM(data);
122         servo_msg.data = data;
123         servo_pub.publish(servo_msg);  // Message publishes to
```

APPENDIX

```
joint_controller/command
        limit = 0;
      }
    }

    // Draw detected detected_faces to screen as circles
    for (unsigned i = 0; i < detected_faces.size(); i++)
    {
      ROS_INFO_STREAM("height=");
      ROS_INFO_STREAM(detected_faces[i].height);

      if (detected_faces[i].height > 125 && detected_faces[i].height < 300)
      {
        limit = 0;
        ROS_INFO_STREAM("Person detected");
        output_msg.data = 1;
        output_pub.publish(output_msg);
        cv::Point center(detected_faces[i].x + detected_faces[i].width * 0.5,
                         detected_faces[i].y + detected_faces[i].height * 0.5);

        // Drawing circle around faces
        cv::ellipse(im_bgr, center, cv::Size(detected_faces[i].width * 0.5, detected_faces[i].height * 0.5), 0, 0, 360,
                    cv::Scalar(255, 0, 255), 4, 8, 0);

        ROS_INFO_STREAM("X=");
        ROS_INFO_STREAM(detected_faces[i].x);

        face_X_position = detected_faces[i].x;
        ROS_INFO_STREAM(face_X_position);
        ROS_INFO_STREAM(frame_centre);

                // Logic for tracking the face
        if (detected_faces[0].width > detected_faces[1].width)
        {
          present_X = detected_faces[0].x;
        }
        else
        {
          present_X = detected_faces[1].x;
        }
        if (prev_X <= present_X + 10 && prev_X >= present_X - 10)
        {
          count_X++;
        }
        else
        {
          count_X = 0;
        }

        e = 210 - detected_faces[i].x;
        if (e < 0)
        {
          e = -e;
        }
        /*
        ROS_INFO_STREAM("E");
        ROS_INFO_STREAM(e);
        ROS_INFO_STREAM("prev_X");
        ROS_INFO_STREAM(prev_X);
        ROS_INFO_STREAM("Present_X");
        ROS_INFO_STREAM(present_X);
        */

        ROS_INFO_STREAM("Count");
        ROS_INFO_STREAM(count_X);

        prev_X = present_X;

        if (count_X >= 1)
        {
          if (present_X < 200)
```

APPENDIX

```
194             {
195               // data = data + kp*e + kd*(e - e1);
196               data = data + kp * e;
197               if (data <= 1|| data >= 4)
198               {
199                 data = 4;
200                 ROS_INFO_STREAM("Data");
201                 ROS_INFO_STREAM(data);
202                 servo_msg.data = data;
203                 servo_pub.publish(servo_msg);
204               }
205               else
206               {
207                 ROS_INFO_STREAM("Data");
208                 ROS_INFO_STREAM(data);
209                 servo_msg.data = data;
210                 servo_pub.publish(servo_msg);
211               }
212               e1 = e;
213             }
214             else if (present_X > 220)
215             {
216               // data = data - kp*e + kd*(e - e1);
217               data = data - kp * e;
218               if (data <= 1 || data >= 4)
219               {
220                 data = 1;
221                 ROS_INFO_STREAM("Data");
222                 ROS_INFO_STREAM(data);
223                 servo_msg.data = data;
224                 servo_pub.publish(servo_msg);
225               }
226               else
227               {
228                 servo_msg.data = data;
229                 ROS_INFO_STREAM("Data");
230                 ROS_INFO_STREAM(data);
231                 servo_pub.publish(servo_msg);
232               }
233               e1 = e;
234             }
235           }
236         }
237
238         else
239         {
240           ROS_INFO_STREAM("Person detected:OUT OF RANGE");
241           limit++;
242           output_msg.data = 0;
243           output_pub.publish(output_msg);
244           ROS_INFO_STREAM(limit);
245           if (limit >= 2)
246           {
247             data = 2.5;
248             ROS_INFO_STREAM("Data");
249             ROS_INFO_STREAM(data);
250             servo_msg.data = data;
251
252             servo_pub.publish(servo_msg);
253             limit = 0;
254           }
255           ROS_INFO_STREAM("DATA:");
256           ROS_INFO_STREAM(data);
257         }
258       }
259
260       cv::imshow(OPENCV_WINDOW, im_bgr);
261       cv::waitKey(3);
262     }
263 };
264
265 int main(int argc, char** argv)
266 {
267   ros::init(argc, argv, "hog_haar_person_detection1");
268   FaceDetectionTracking object;
269   ros::spin();
270   return 0;
271 }
```

APPENDIX C

Main Page | Classes | Files

File List | File Members

face_tracking_action_server.cpp

Go to the documentation of this file.

```
  1 #include <ros/ros.h>
  2 #include <actionlib/server/simple_action_server.h>
  3 #include <saya_hospitality_perception_msgs/FaceTrackingAction.h>
  4 #include <std_msgs/Float64.h>
  5 #include <std_msgs/Bool.h>
  6
  7 static int count_;
  8 class FaceTrackingActionNode
  9 {
 10 protected:
 11
 12   ros::NodeHandle nh_;
 13   // NodeHandle instance must be created before this line. Otherwise
    strange error may occur.
 14
    actionlib::SimpleActionServer<saya_hospitality_perception_msgs::FaceTrack
    ingAction> as_;
 15   std::string action_name_;
 16   // create messages that are used to published feedback/result
 17   saya_hospitality_perception_msgs::FaceTrackingFeedback feedback_;
 18   saya_hospitality_perception_msgs::FaceTrackingResult result_;
 19
 20   ros::Subscriber sub_;
 21   bool goal_;
 22
 23 public:
 24
 25   FaceTrackingActionNode(std::string name) :
 26     as_(nh_, name, false),
 27     action_name_(name)
 28   {
 29
 30     as_.registerGoalCallback(boost::bind(&FaceTrackingActionNode::goalCB,
    this));
 31
    as_.registerPreemptCallback(boost::bind(&FaceTrackingActionNode::preemptC
    B, this));
 32
 33     //ROS_INFO_STREAM("as_.started");
 34     as_.start();
 35
 36     ROS_INFO_STREAM("--------FaceTrackingAction_Server is started-------
    ");
 37   }
 38
 39   ~FaceTrackingActionNode(void)
 40   {
 41   }
 42
 43   void goalCB()
 44   {
 45     //ROS_INFO_STREAM("Goal Started");
 46     // accept the new goal
 47     goal_ = as_.acceptNewGoal()->goal;
 48     ROS_INFO_STREAM(goal_);
 49     if(goal_)
 50     {
 51         ROS_INFO_STREAM("--------FaceTrackingAction_Server Goal Recieved-
    -------");
 52     sub_ =
    nh_.subscribe("/sayabot_hospitality_perception/face_detection/output", 1,
    &FaceTrackingActionNode::analysisCB, this);
```

APPENDIX

```
53          }
54    }
55
56    void preemptCB()
57    {
58      ROS_INFO("%s: Preempted", action_name_.c_str());
59      // set the action state to preempted
60      as_.setPreempted();
61    }
62
63
64
65    void analysisCB(const std_msgs::Bool msg)
66    {
67            if (!as_.isActive())
68        return;
69
70
71           // ROS_INFO_STREAM("AnalyaisCB is called");
72       // ROS_INFO_STREAM("MSG");
73        ROS_INFO_STREAM(msg);
74        if (msg.data)
75        { count_ += 1;
76                                ROS_INFO_STREAM(count_);
77
78                    feedback_.feedback = true;
79                                }
80                  else{
81                          count_ -= 1;
82                          feedback_.feedback = false;
83                  }
84                  if(count_ >3)
85                  {
86                          result_.result= true;
87                          count_ = 0;
88                          as_.setSucceeded(result_);
89
90          }
91          if(count_ < -3){
92                  result_.result = false;
93                  count_ = 0;
94                  as_.setSucceeded(result_);
95
96          }
97          }
98
99
100   };
101
102
103   int main(int argc, char** argv)
104   {
105     ros::init(argc, argv, "FaceTrackingAction_Server");
106
107     FaceTrackingActionNode FaceTrackingAction_(ros::this_node::getName());
108     ros::spin();
109
110     return 0;
111   }
```

APPENDIX D

Main Page | Namespaces | Files

File List

lip_movement_2.py

Go to the documentation of this file.

```
#!/usr/bin/env python

import roslib
import rospy
import rospy
from std_msgs.msg import Float64
import threading
import os

from sound_play.msg import SoundRequest
from sound_play.libsoundplay import SoundClient

lip_pub = rospy.Publisher('/lip_controller/command', Float64, queue_size = 10)

word = "Good Morning.. Saya bot Welcomes you. What can I do for you?"
#word_length = len(word)
#print word_length
#x = word_length/(word_length-10)

def speak(word):
        soundhandle = SoundClient()
        rospy.sleep(2)
        soundhandle.stopAll()
        rospy.loginfo("speech")
        soundhandle.say(word)
        print ("said")
        rospy.loginfo("Done")


def word_count(word_in):
        word_array_string = word_in.split()
        word_array_len = []
        for individual_words in word_array_string:
                word_array_len.append(len(individual_words))
        return word_array_len

'''
        l = l + 1
        if l is 2:
                i = 0.25
        elif l is 3 | l is 4:
                i = 0.24
        elif l is 5 | l is 6:
                i = 0.23
        elif l is 7 | l is 8:
                i = 0.22
        elif l is 9 | l is 10:
                i = 0.21
        elif l > 10:
                i = 0.20
'''



def lip_sync(word_array):
    rospy.sleep(2.5)
    l = 0
    i = 0.2
    print i
    for word_lengths in word_array:
        print word_lengths
```

APPENDIX

```
        l = l + 1
        if l is 2:
                i = 0.21
        elif l is 3 | l is 4:
                i = 0.20
        elif l is 5 | l is 6:
                i = 0.19
        elif l is 7 | l is 8:
                i = 0.18
        elif l is 9 | l is 10:
                i = 0.17
        elif l > 10:
                i = 0.16
        print ("l IS "), l

        if word_lengths == 1:
            pass

        elif word_lengths == 2:
            pass

        elif word_lengths == 3:
            lip_pub.publish(1.75)
            rospy.sleep(i)
            lip_pub.publish(1.65)
            rospy.sleep(i)

        elif word_lengths == 4:
            lip_pub.publish(1.77)
            rospy.sleep(i)
            lip_pub.publish(1.65)
            rospy.sleep(i)

        elif word_lengths == 5:
            lip_pub.publish(1.78)
            rospy.sleep(i)
            lip_pub.publish(1.65)
            rospy.sleep(i)

        elif word_lengths == 6:
            lip_pub.publish(1.8)
            rospy.sleep(i)
            lip_pub.publish(1.65)
            rospy.sleep(i)

        elif word_lengths == 7:
            lip_pub.publish(1.8)
            rospy.sleep(i)
            lip_pub.publish(1.65)
            rospy.sleep(i)

        elif word_lengths == 8:
            lip_pub.publish(1.8)
            rospy.sleep(i)
            lip_pub.publish(1.65)
            rospy.sleep(i)

        elif word_lengths == 9:
            lip_pub.publish(1.8)
            rospy.sleep(i)
            lip_pub.publish(1.65)
            rospy.sleep(i)
            lip_pub.publish(1.7)
            rospy.sleep(i)
            lip_pub.publish(1.65)
            rospy.sleep(i)

        elif word_lengths == 10:
            lip_pub.publish(1.8)
            rospy.sleep(i)
            lip_pub.publish(1.65)
            rospy.sleep(i)
            lip_pub.publish(1.8)
            rospy.sleep(i)
            lip_pub.publish(1.65)
```

APPENDIX

```
137                 rospy.sleep(i)
138
139             elif word_lengths == 11:
140                 lip_pub.publish(1.8)
141                 rospy.sleep(i)
142                 lip_pub.publish(1.65)
143                 rospy.sleep(i)
144                 lip_pub.publish(1.7)
145                 rospy.sleep(i)
146                 lip_pub.publish(1.65)
147                 rospy.sleep(i)
148
149
150             elif word_lengths == 12:
151                 lip_pub.publish(1.8)
152                 rospy.sleep(i)
153                 lip_pub.publish(1.65)
154                 rospy.sleep(i)
155                 lip_pub.publish(1.8)
156                 rospy.sleep(i)
157                 lip_pub.publish(1.65)
158                 rospy.sleep(i)
159
160
161             elif word_lengths == 13:
162                 lip_pub.publish(1.85)
163                 rospy.sleep(i)
164                 lip_pub.publish(1.65)
165                 rospy.sleep(i)
166                 lip_pub.publish(1.7)
167                 rospy.sleep(i)
168                 lip_pub.publish(1.65)
169                 rospy.sleep(i)
170
171             elif word_lengths >= 13:
172                 lip_pub.publish(1.8)
173                 rospy.sleep(i)
174                 lip_pub.publish(1.65)
175                 rospy.sleep(i)
176                 lip_pub.publish(1.78)
177                 rospy.sleep(i)
178                 lip_pub.publish(1.65)
179                 rospy.sleep(i)
180
181
182 def start():
183             lip_pub.publish(1.65)
184             #word  = input("Type what to say :\n")
185             individual_word_lengths = word_count(word)
186             print individual_word_lengths
187             #speak(word)
188             #lip_sync_fake_algorithm(individual_word_lengths)
189             thread_word_say = threading.Thread(target=speak, args=(word, ))
190             thread_lip_sync = threading.Thread(target=lip_sync, args=
(individual_word_lengths, ))
191
192             thread_word_say.start()
193             thread_lip_sync.start()
194             thread_word_say.join()
195             thread_lip_sync.join()
196
197
198 if __name__ == '__main__':
199             rospy.init_node('lip_movemnet', anonymous=True)
200             start()
201
202
203
204
```

APPENDIX E

Main Page | Namespaces | Classes | Files

File List

sayabot_test.py

Go to the documentation of this file.

```
#!/usr/bin/python

import roslib
import rospy
import smach
import smach_ros
import os
from std_msgs.msg import Bool
import states.face_detection_state as fds
import states.arm_maipulation_state as ams

def main():
    rospy.init_node('smach_example_state_machine')

    # Create a SMACH state machine
    sm = smach.StateMachine(outcomes=['Stop'])

    with sm:
        # Add states to the container
        smach.StateMachine.add('Face_Detection', fds.Face_Detection(),
                               transitions={'face_detected':'Detected',
'No_face_detected':'Not_Detected'})
        smach.StateMachine.add('Detected', fds.Detected(),
                               transitions={'Arm
Manipultaion':'Arm_Manipulation'})

        smach.StateMachine.add('Not_Detected', fds.Not_Detected(),
                               transitions={'Redoing':'Face_Detection'})

        smach.StateMachine.add('Arm_Manipulation',
ams.Arm_Manipulation(),
                               transitions={'Executed':'Face_Detection'})

    sys = smach_ros.IntrospectionServer('sayabot_hospitality', sm,
'/Sayabot_Hospitality')
    sys.start()

    outcome = sm.execute()

    rospy.spin()
    sys.stop()

if __name__ == '__main__':
    main()
```

APPENDIX F

Global Costmap Parameters:

```
global_costmap:
  global_frame: /map
  robot_base_frame: /base_link
  update_frequency: 2
  static_map: true
  rolling_window: true
```

Base Local Planner Parameters:

```
TrajectoryPlannerROS:
  max_vel_x: 0.4
  min_vel_x: 0.1
  max_rotational_vel: 0.5
  min_in_place_rotational_vel: 0.1

  acc_lim_th: 3.2
  acc_lim_x: 4
  acc_lim_y: 4

  dwa: true
  holonomic_robot: false
  meter_scoring: true

  sim_time: 2.0
  vx_samples: 4
  vy_samples: 0
  vtheta_samples: 10

  #goal tolerance parameters

  yaw_goal_tolerance: 0.3
  xy_goal_tolerance: 0.2
  latch_xy_goal_tolerance: true
```

APPENDIX G

```
<?xml version="1.0"?>

<robot name="robot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <link
    name="base_link">
    <inertial>
      <origin
        xyz="-0.0431943387874082 0.000534457647598531 -0.0113530325125777"
        rpy="0 0 0" />
      <mass
        value="0.995121181496884" />
      <inertia
        ixx="0.0215839766020311"
        ixy="-8.11137633533795E-05"
        ixz="0.000490635995095507"
        iyy="0.0166672854273941"
        iyz="-2.45340814753691E-05"
        izz="0.0376813643686826" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://hdf_model/meshes/base_link.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.792156862745098 0.819607843137255 0.933333333333333 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://hdf_model/meshes/base_link.STL" />
      </geometry>
    </collision>
  </link>

   <!--left wheel parameters-->

  <link
    name="front_left_wheel_link">
    <inertial>
      <origin
        xyz="-0.015 1.9151347174784E-15 -2.56739074444567E-16"
        rpy="0 0 0" />
```

APPENDIX

```
      <mass
        value="0.333988998188438" />
      <inertia
        ixx="0.000588196374434614"
        ixy="-3.31390711600779E-19"
        ixz="2.50851803456101E-20"
        iyy="0.000322598581729387"
        iyz="-7.66352703869503E-36"
        izz="0.000322598581729387" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://hdf_model/meshes/front_left_wheel_link.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.792156862745098 0.819607843137255 0.933333333333333 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <cylinder length="0.03" radius="0.058" />
      </geometry>
    </collision>
  </link>


  <joint
    name="front_left_wheel_joint"
    type="continuous">
    <origin
      xyz="-0.0870000000000004 0.16124 -0.0330000000000298"
      rpy="0 0 -1.57079632679489" />
    <parent
      link="base_link" />
    <child
      link="front_left_wheel_link" />
    <axis
      xyz="1 0 0" />
     <limit
      effort="10"
      velocity="10" />
    <dynamics
      damping="1"
      friction="0" />
```

```
</joint>


 <transmission name="front_left_wheel_trans" type="SimpleTransmission">
    <type>transmission_interface/SimpleTransmission</type>
    <actuator name="front_left_wheel_motor">
       <mechanicalReduction>1</mechanicalReduction>
  </actuator>
    <joint name="front_left_wheel_joint">
    <hardwareInterface>VelocityJointInterface</hardwareInterface>
  </joint>
 </transmission>

<!--Right wheel link-->

<link
  name="front_right_wheel_link">
  <inertial>
    <origin
      xyz="-0.015 4.16333634234434E-17 -2.08166817117217E-17"
      rpy="0 0 0" />
    <mass
      value="0.333988998188438" />
    <inertia
      ixx="0.000588196374434614"
      ixy="-1.21501638934715E-19"
      ixz="1.28042447991415E-20"
      iyy="0.000322598581729387"
      iyz="-1.6940658945086E-21"
      izz="0.000322598581729387" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://hdf_model/meshes/front_right_wheel_link.STL" />
    </geometry>
    <material
      name="">
      <color
        rgba="0.792156862745098 0.819607843137255 0.933333333333333 1" />
    </material>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
        <cylinder length="0.03" radius="0.058" />
    </geometry>
  </collision>
```

```
</link>


<joint
  name="front_right_wheel_joint"
  type="continuous">
  <origin
    xyz="-0.086657001701574 -0.16124 -0.033"
    rpy="0.0260968832194341 0 1.5707963267949" />
  <parent
    link="base_link" />
  <child
    link="front_right_wheel_link" />
  <axis
    xyz="-1 0 0" />
</joint>

 <transmission name="front_right_wheel_trans" type="SimpleTransmission">
    <type>transmission_interface/SimpleTransmission</type>
    <actuator name="front_right_wheel_motor">
       <mechanicalReduction>1</mechanicalReduction>
  </actuator>
    <joint name="front_right_wheel_joint">
    <hardwareInterface>VelocityJointInterface</hardwareInterface>
  </joint>
 </transmission>



<!--laser link-->
<link
  name="laser_link">
  <inertial>
    <origin
      xyz="8.9889926402964E-05 -7.07366090337206E-05 0.0316235280435242"
      rpy="0 0 0" />
    <mass
      value="0.142186104854555" />
    <inertia
      ixx="7.9867958456067E-05"
      ixy="1.06075864789783E-07"
      ixz="-6.50241068735055E-08"
      iyy="8.00805631158864E-05"
      iyz="2.50611691787948E-07"
      izz="4.89630225093579E-05" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://hdf_model/meshes/laser_link.STL" />
```

```
      </geometry>
      <material
        name="">
        <color
          rgba="0.792156862745098 0.819607843137255 0.933333333333333 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://hdf_model/meshes/laser_link.STL" />
      </geometry>
    </collision>
  </link>
  <joint
    name="laser_joint"
    type="fixed">
    <origin
      xyz="0.28106822908108 0 0"
      rpy="0 0 0" />
    <parent
      link="base_link" />
    <child
      link="laser_link" />
    <axis
      xyz="0 0 0" />
  </joint>
</robot>
```