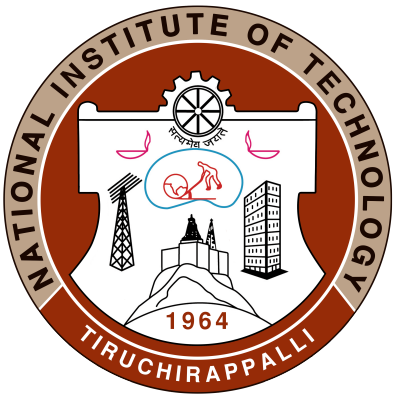
SUMMER'19 INTERNSHIP REPORT

Submitted by
Gudapati Nitish
Roll No.: 111117036

In fullfillment of Summer Internship for award of the degree the degree of
BACHELOR OF TECHNOLOGY
In
MECHANICAL ENGINEERING



National Institute of Technology
Tiruchirappalli (NIT-T)
Tamil Nadu -INDIA

SUMMER'19 INTERNSHIP REPORT

Improving Reliability of Localisation and Navigation and Development of User Interface of Mobile Robot: RoboMuse 4.0

Srikrishna S
Electronics & Communication Engineering, NIT-Trichy

Gudapati Nitish
Mechanical Engineering, NIT-Trichy

Supervisor:
Prof. Dr. S K Saha
Mechanical Engineering, IIT-Delhi

Mentor:
Deepanshu Singh
Mechanical Engineering, IIT-Delhi

Department of Mechanical Engineering
Indian Institute of Technology
Hauz Khas, Delhi - 110019

Indian Institute Of Technology, Delhi
Hauz Khas, New Delhi - 110016
Tel.: +91-11-2659 1135, Fax: +91-11-2658 2053
Email: saha@mech.iitd.ac.in, sahaiitd@gmail.com
URL: http://web.iitd.ac.in/s̃aha

Department of Mechanical Engineering
Dr. Subir Kumar Saha
Professor

Internship Certificate for Mr.S Srikrishna

This is to certify that **Mr. S Srikrishna**, an undergraduate student of Electronics and Communication Engineering, National Institute Of Technology, Tiruchirappalli, had worked on the project **'RoboMuse 4.0'** under my guidance, during the summer of 2019 as part of an internship program from May 14, 2019 in our team. The project work involved Improving the Reliability of Localisation and Navigation, and Development of User Interface of Mobile Robot. S Srikrishna worked on the following:

- Improving the SLAM navigation planner
- Solving the kidnapped robot problem by using aruco markers robots
- Improving odometry error correction using aruco markers
- Testing the efficiency of the robot

Dr. S. K. Saha
Professor
Mechanical Engineering
IIT Delhi

Indian Institute Of Technology, Delhi
Hauz Khas, New Delhi - 110016
Tel.: +91-11-2659 1135, Fax: +91-11-2658 2053
Email: saha@mech.iitd.ac.in, sahaiitd@gmail.com
URL: http://web.iitd.ac.in/s̃aha

Department of Mechanical Engineering
Dr. Subir Kumar Saha
Professor

Internship Certificate for Mr.Gudapati Nitish

This is to certify that **Mr. Gudapati Nitish**, an undergraduate student of Mechanical Engineering, National Institute Of Technology, Tiruchirappalli, had worked on the project **'RoboMuse 4.0'** under my guidance, during the summer of 2019 as part of an internship program from May 14, 2019 to July 10, 2019 in our team. The project work involved Improving the Reliability of Localisation and Navigation, and Development of User Interface of Mobile Robot. Gudapati Nitish worked on the following:

- Improving odometry error correction using aruco markers
- Implementing voice control
- Designing a User Interface using Qt
- Testing the efficiency of the robot

Dr. S. K. Saha
Professor
Mechanical Engineering
IIT Delhi

Indian Institute Of Technology, Delhi
Hauz Khas, New Delhi - 110016
Tel.: +91-11-2659 1135, Fax: +91-11-2658 2053
Email: saha@mech.iitd.ac.in, sahaiitd@gmail.com
URL: http://web.iitd.ac.in/s̃aha

Department of Mechanical Engineering
Dr. Subir Kumar Saha
Professor

Internship Certificate for Mr.S Srikrishna

This is to certify that **Mr. S Srikrishna**, an undergraduate student of Electronics and Communication Engineering, National Institute Of Technology, Tiruchirappalli, had worked on the project **'RoboMuse 4.0'** under my guidance, during the summer of 2019 as part of an internship program from May 14, 2019 in our team. As his Internship Supervisor, I got the opportunity to observe him closely and saw that he was technically sound, diligent, a good team player and eager to learn about Robotics. The goal of the project was to improve the reliability of Localisation and Navigation using ROS and OpenCV, and develop a user friendly GUI for the Mobile Robot in Qt. In particular, S Srikrishna worked on the following:

- Improving the SLAM navigation planner
- Solving the kidnapped robot problem by using aruco markers robots
- Improving odometry error correction using aruco markers
- Testing the efficiency of the robot

Dr. S. K. Saha
Professor
Mechanical Engineering
IIT Delhi

Indian Institute Of Technology, Delhi
Hauz Khas, New Delhi - 110016
Tel.: +91-11-2659 1135, Fax: +91-11-2658 2053
Email: saha@mech.iitd.ac.in, sahaiitd@gmail.com
URL: http://web.iitd.ac.in/s̃aha

Department of Mechanical Engineering
Dr. Subir Kumar Saha
Professor

Internship Certificate for Mr.Gudapati Nitish

This is to certify that **Mr. Gudapati Nitish**, an undergraduate student of Mechanical Engineering, National Institute Of Technology, Tiruchirappalli, had worked on the project **'RoboMuse 4.0'** under my guidance, during the summer of 2019 as part of an internship program from May 14, 2019 to July 10, 2019 in our team. As his Internship Supervisor, I got the opportunity to observe him closely and saw that he was technically sound, diligent, a good team player and eager to learn about Robotics. The goal of the project was to improve the reliability of Localisation and Navigation using ROS and OpenCV, and develop a user friendly GUI for the Mobile Robot in Qt. In particular, Gudapati Nitish worked on the following:

- Improving odometry error correction using aruco markers
- Implementing voice control
- Designing a User Interface using Qt
- Testing the efficiency of the robot

Dr. S. K. Saha
Professor
Mechanical Engineering
IIT Delhi

Abstract

RoboMuse is IIT Delhi's own ingenious mobile platform series to demonstrate 24/7 working scenario, started in 2009. Over the past 8 years, RoboMuse has had 4 iterations with each one adding something unique to the design. Over all these years the RoboMuse was built with the essential aim of being able to function without any human interaction i.e autonomously and a number of applications were developed to demonstrate the requirements.

The robot is capable of navigating through its environment in the presence of static and dynamic obstacles to reach any assigned (feasible) goals. For product development, a record of dynamic specifications of the robot including its repeatability and reproducibility is necessary. Further to demonstrate the autonomous navigation a number of applications such as NavTest, Docking the robot to charge it via different means, Following way-points in a known map and recording live video in the process were developed.

Marker based correction for odometry for better localization was achieved by using aruco markers and transforms. Mapping was done along with multiple markers embedded in it. This map when reloaded along with markers was used for odometry correction of the robot. The robot's navigation was improved by changing the trajectory planner. Also costmap artifacts were removed by selectively clearing the costmap when required. The speech recognition was improved for the robot by tuning the pocketsphinx package. Also an UI was developed for the robot for making it easy for non technical people to use the robot easily.

The entire system was setup on the Robot Operating System framework. The main reasons for this being re-usability of code, modularity and ease of communication between the modules.

Contents

List of Figures

1 Introduction

1.1 Mobile robotics

Mobile robotics is the field of robotics which deals with robots capable of locomotion either using wheels or legs. It is a field of rapid research potential today due to the extensive range of promising future applications. These include robot servants, personal assistant robots, delivery robots, warehouse robots etc. The primary research areas within this field include motion planning, exploration using the robot, Simultaneous Localization and Mapping (SLAM). Mobile robots are generally controlled by software and use sensors and other gear to identify their surroundings. These robots combine the progress in artificial intelligence with physical robotics, which allows them to navigate in their surroundings. The work on improving the robustness of the robot is of primary importance to this field as reliability is crucial to work in complex and uncertain environments. Some examples of mobile robots include Mars rovers, PR2, Turtlebot etc.

1.2 ROS

Robot Operating System is a software framework which offers a streamlined approach to robot development. It contains many software packages and drivers to make the process of robot development much faster and eliminates the continuous reinvention of the wheel, which was very prevalent in robotics until ROS gained notoriety. It offers communication methods between multiple different programs called nodes and allows us to establish these connections in a much more relaxed manner. It is highly modular, and the code is highly reusable. It enables programmers to use many languages of their choice to program the robots using Python, C++, Matlab etc., also it extensively allows for cooperation with various team members because of independent packages which can be used together with each other. It is also supported on multiple hardware platforms including amd64 based processors and arm-based processors. It also is cross-platform in terms of software working on Windows and Linux. It is also highly accessible to all programmers because it is open source and has an active community of users and enthusiasts. ROS has been improving year by year with more packages releasing and more packages becoming more efficient. It has become an essential tool for a roboticist.

1.3 Hardware

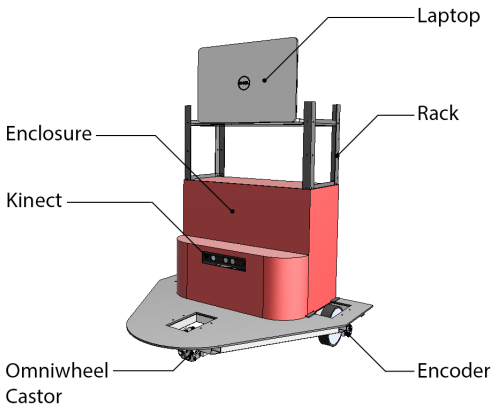


Figure 1: Robomuse 4.0

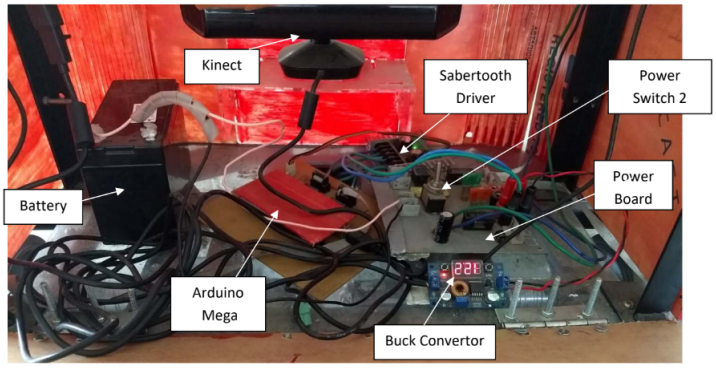


Figure 2: Robot Circuit

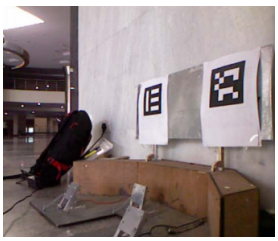


Figure 3: Charging Dock

The robot is a differential drive robot with an omnidirectional wheel as a caster. It houses a Microsoft Kinect sensor for assimilating RGB-D images. It has an Arduino Controller and a laptop for processing. It is powered by Pb-Acid battery and a custom-build power supply circuit. It also features a charging circuit which allows for automatic docking and charging.

1.4 Odometry

Odometry is the process of estimating the distance travelled and path travelled by the robot by mounting sensors on the shaft of the actuator. This can be used in the case of differential drive robots such as robomuse to estimate the x,y, yaw of the robot when it moves on a flat horizontal plane. However, this requires the assumption of pure rolling and no noise. SO we adopt a probabilistic version of this model.

In local frame of Robot

$$v_{xr} = (\omega_L + \omega_R)\frac{r}{2} \tag{1}$$

$$v_{yr} = 0 \tag{2}$$

$$\omega_{zr} = (\omega_R - \omega_L)\frac{r}{R} \tag{3}$$

In global frame

$$\begin{pmatrix} v_x \\ v_y \\ w_z \end{pmatrix} = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_{xr} \\ v_{yr} \\ w_{zr} \end{pmatrix} \tag{4}$$

Taking multiplying by the inverse of the 3x3 matrix on both sides we get

$$\begin{pmatrix} v_x \\ v_y \\ w_z \end{pmatrix} \begin{pmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} v_{xr} \\ v_{yr} \\ w_{zr} \end{pmatrix} \tag{5}$$

Summing v_x, v_y and ω_z we can find the pose of the robot at any instant. This is called as odometry.

We can extend this model of odometry further by considering it in a probabilistic form.

1.5 Mapping

The process of creating a map of the environment using the depth image generated by the Kinect sensor is called mapping. The Robomuse uses an algorithm called RTAB (Real-Time Appearance Based) map to create a map of the surroundings. The mapping algorithm works by converting the depth image to a pseudo-2D laser scan of the surroundings so that the algorithm can determine the distance it is away from objects in its surroundings. RTAB map requires the tuning of parameters so that it can work and is present as a convenient ROS package.

1.6 SLAM

Simultaneous Localization and Mapping is a process where a robot maps an environment where it also uses the features of the environment to find its pose in it. It utilizes Extended Kalman filter and Particle filter to establish this. The data from the wheel odometry of the robot and the data obtained from the Kinect sensor are the input data to this method. It allows the robot to robustly estimate its pose with a probabilistic model rather than a deterministic model. The RTAB map package offers a SLAM extension, so the robot predicts its pose while mapping.

1.7 Navigation

The process of moving the robot autonomously from place to place in a known map is called navigation. The robot is equipped to navigate in the environment using a map of the environment and the wheel odometry data. When given a goal point, the robot plans a global path based on the map which it has. It also plans a local path based on the immediate sensor data it receives from the Kinect sensor. This ensures that the robot can perform static and dynamic obstacle avoidance. It uses A* search algorithm for global path planning and DWA planner for local motion planning. To interface all this, ROS provides a package called the navigation stack, which makes the development of a mobile robot a much faster job. The navigation stack requires parameters which have to be tuned based on the attributes of the robot currently working on, and once the parameters are set, the robot is ready to navigate in a map.

1.8 ArUco markers

ArUco marker is an augmented reality square marker which is surrounded by a black border and inside it contains a binary matrix which determines its unique identifier (id). The black border allows fast detection in the image in a lighter background, and the binary codification facilitates its identification, and it also uses various error correction coding methods for identification.

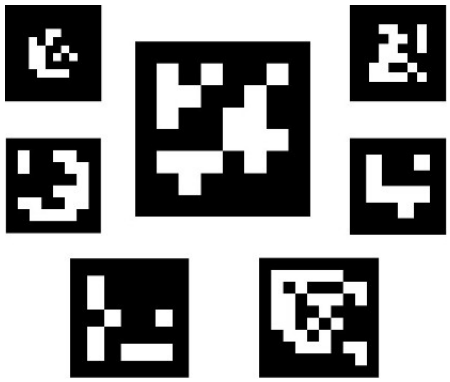


Figure 4: ArUco Markers

1.9 Kidnapped robot problem

The localization techniques developed can be distinguished according to the type of problem. Tracking or local techniques aim at compensating odometric errors occurring during robot navigation. However, they require that the initial location of the robot is known and they cannot recover if they lose track of the robot's existing position. Another set of approaches is global techniques. They are designed to estimate the position of the robot under any uncertainty. These techniques solve the wake-up robot problem, in that they can localize a robot without any prior knowledge about its position. Further, they can handle the kidnapped robot problem, in which a robot is carried to an arbitrary location during its operation. The wake-up problem is a particular case of the kidnapped robot problem in which the robot knows that it has been carried away. Global localization techniques are advanced than local ones. They can cope with situations in which the robot is likely to experience serious positioning errors.

Figure 5: Robot at Home

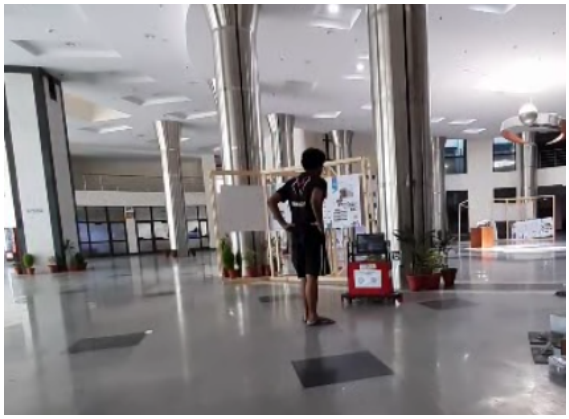


Figure 6: Robot Kidnapped

1.10 PocketSphinx

CMU Sphinx is a package of speech recognition systems containing a series of speech recognizers and an acoustic model trainer. The speech decoders include acoustic models and sample applications. The available resources include software for acoustic model training, Language model compilation and a public domain pronunciation dictionary, cmudict.

PocketSphinx is a lightweight speech recognition engine, tuned explicitly for handheld and mobile devices. It is a part of the CMU Sphinx Open Source Toolkit For Speech Recognition.

1.11 QT

Qt is an open-source widget toolkit for creating graphical user interfaces as well as cross-platform applications. The interfaces are designed to run on various software and hardware platforms such as Linux, Windows, macOS, Android or embedded systems.

2 Marker Correction

2.1 Pose Estimation Algorithm

The aruco_ros package returns the pose of the marker with respect to the camera frame. By utilizing the ROS tf and tf2 packages, this can be converted to be in the form of X, Y, Z translations and Quaternion rotations.

Quaternions for Rotations

Unit quaternions, which are also known as versors, provide a convenient mathematical notation for representing orientations and rotations of objects in three dimensions. Compared to Euler angles they are simpler to compose and avoid the problem of gimbal lock. These are more compact, more numerically stable, and more efficient compared to rotation matrices. Quaternions have applications in computer graphics, computer vision, robotics, navigation, molecular dynamics, flight dynamics, orbital mechanics of satellites and crystallographic texture analysis.

Here, they are used for representing the rotation between the camera and the robot's base and the inverse.

When used to represent rotation, unit quaternions are also called rotation quaternions as they represent the three-dimensional rotation group. When used to represent an orientation (rotation relative to a reference coordinate system), they are called orientation quaternions or attitude quaternions.

2.2 Pose embedded on the map

To estimate the pose of the robot with respect to the marker we have to apply a frame transformation from the robot's camera frame to the frame of the marker. This can be done by using the 6 DoF pose estimate yielded by the marker. The tf package of ROS allows us to perform this transformation with ease it obtains the rotation matrix for the marker and the translation. This is used to compute the location of the camera with respect to the marker.

The camera and the robot's base link are connected via a static transformation. Once this transformation is done, the robot's location with respect to marker can be estimated.

First, let us consider mapping. We receive from the aruco_ros package the transform of robot's camera with respect to the marker. This can be viewed as robot's base link with respect to marker whose transformation matrix we define as ${}^{marker}_{base}T$. Now, we take inverse of this Transform

$$ {}^{marker}_{base}T^{-1} = {}^{base}_{marker}T \tag{6} $$

We use the odometry data to convert ${}^{marker}_{base}T$ to ${}^{map}_{marker}T$ where map is the frame from which we started to map from. If x,y,θ , are the x coordinate, y coordinate and θ is the angle robot makes with x-axis on x-y plane, then

$$ {}^{marker}_{base}T_{odometry} = \begin{pmatrix} cos(\theta) & sin(\theta) & 0 & x \\ sin(\theta) & cos(\theta) & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7} $$

Then,

$$ {}^{map}_{marker}T = {}^{map}_{base}T_{odometry} * {}^{base}_{marker}T \tag{8} $$

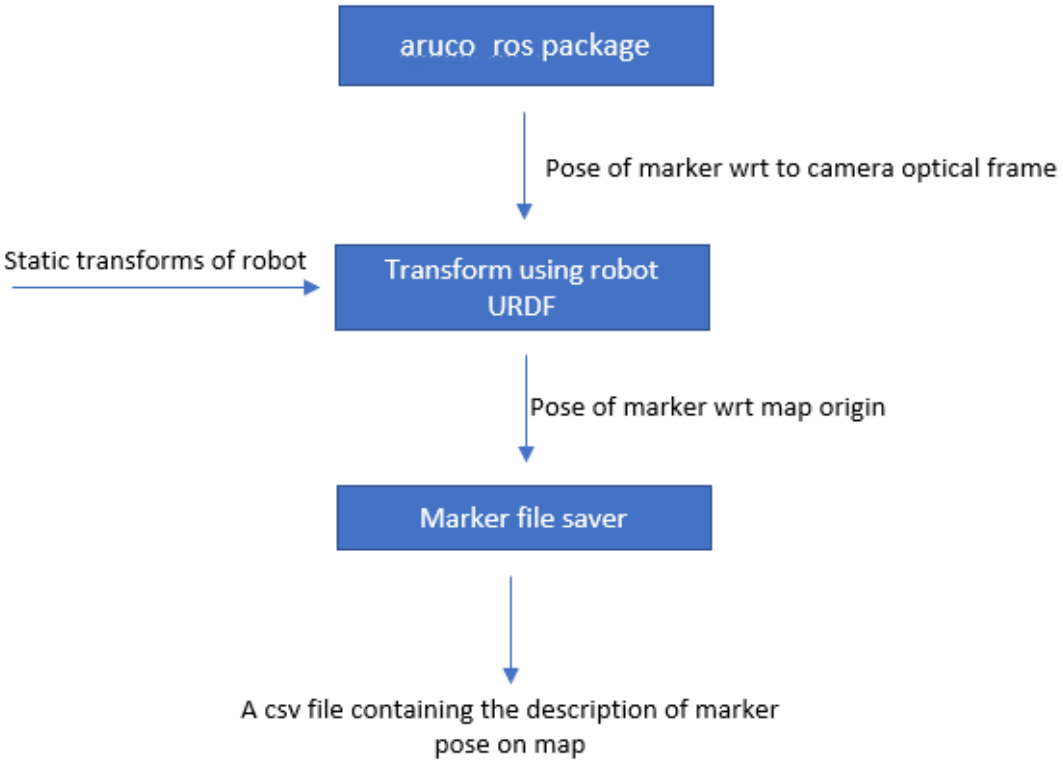


Figure 7: Mapping

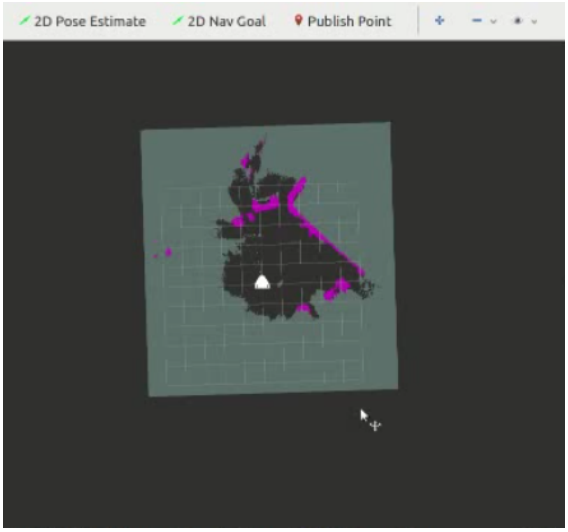


Figure 8: Robot initialized

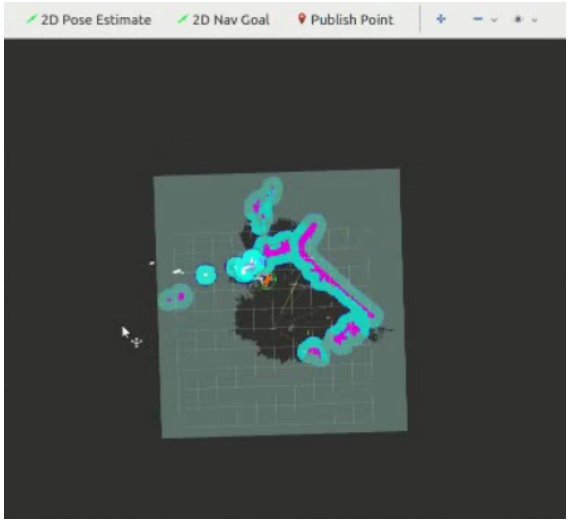


Figure 9: Robot Recovered to Marker location

2.3 Loading markers during navigation

While navigating autonomously using the navigation stack package the robot loads the already generated map while this is being loaded, the CSV file containing the marker pose with respect to the map can also be loaded. This can be generated as a tf frame. We also run the aruco_ros package, which gets us the tf frames of the marker with respect to the camera on the robot. We can take the inverse of this transform and estimate the robot's pose with respect to the marker. Now we can utilize the existing data of where the marker is on the map and generate a pose estimate of the robot with only the saved marker data and estimated pose from the marker. This is a powerful tool.

2.4 Correction

One application of the tool we discussed is as a correction procedure for correcting odometry of the robot as error accumulates over time. This can correctly estimate the robot's position and correct it. This can also be extended as a solution to the kidnapped robot problem. As a way to find the estimate of the pose of the robot when it is turned on in an unknown environment. Now for correction given ${}^{map}_{marker}T$ Again we obtain ${}^{marker}_{base}T$ from aruco_ros. Now we have to estimate ${}^{map}_{base}T$. To compute this all we need to do is:

$$ {}^{map}_{base}T = {}^{map}_{marker}T * {}^{marker}_{base}T \tag{9} $$

2.5 Filtering algorithm

As expected from any direct sensor data, the pose data is noisy, and this is undesirable. We can implement a six-dimensional Kalman filter to filter out the noise and get a better pose estimate making the robot more reliable. The initial noise covariance matrix can be obtained from the initially obtained data during mapping the area this allows for faster convergence to correct value when estimating pose while navigating.

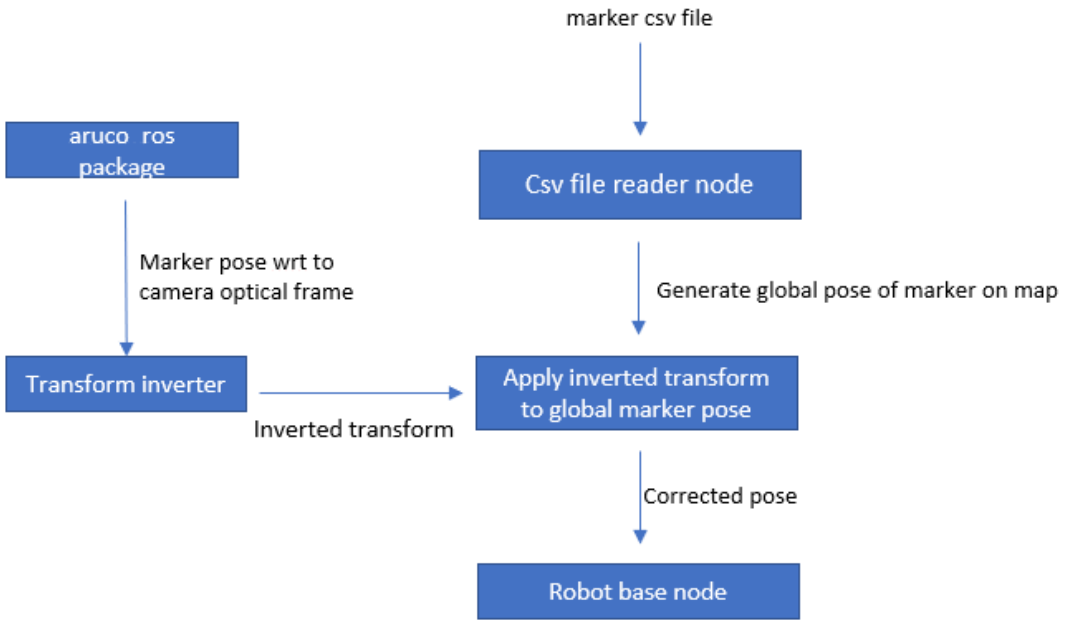


Figure 10: Navigation

3 Multi Marker Extension

The single marker-based pose correction can be extended to multiple markers by using simple adjustments to the filesystem and using the fact that each aruco marker possesses an ID. These IDs can be used to give each marker its location on the map uniquely. This is achieved through various things.

3.1 Filesystem

When the map is being created, a new directory is created by using the date and time of starting the program to preserve uniqueness. Inside this directory, several CSV files each named after the ID of the corresponding marker are generated and the poses of these with respect to the map are stored. This allows for correcting the Localization at many more locations and also allows kidnapped robot problem to be solved from broader area coverage.

3.2 Dictionary style implementation

The implementation of multiple markers needed the dynamic generation of tf frames when new markers were encountered. This was achieved by generating a new dictionary element whenever a new marker was encountered. The dictionary would start empty and fill as encountered, and when a marker is encountered, the dictionary is checked to ensure ID. The dictionary corresponded to the saving aspect within the file system so that the proper data is saved into the proper CSV file.

3.3 Loading into navigation

The file system is reaccessed based on the date and time given as parameters. Now a dictionary is created anew with data from the existing map markers present. Whenever a marker is encountered, the previously discussed algorithm estimates the pose of the robot. The presence of multiple markers introduces a challenge of choosing pose when multiple markers are in the visual frame. With previous data, we were able to estimate that markers which are closer by are more reliable for pose estimation, and the closer marker is used to estimate the pose. The additional advantage of putting multiple markers on the map allows generating waypoints to points based on the markers rather than defining ourselves by teleoperating the robot. It also allows the robot to increase its motion to a vast range, where corrections happen at regular intervals.

4 Improvement in Navigation

4.1 Navigation - Planner Improvement

4.1.1 Global Planner

The global planner generates the trajectory from the source to the destination. The global-planner requires a map of the entire environment to calculate the best route.

There are three global planners - carrot planner, navfn and global planner. The carrot planner checks if the given goal is an obstacle, and picks an alternative goal close to the original one, by back-propagating along the vector between the robot and the goal point. Eventually, it passes this valid goal as a plan for the local planner or controller. Hence, this planner does not do any global path planning but is helpful when the robot has to move close to the given goal even if unreachable. The navfn planner uses Dijkstra's algorithm to find a global path with a minimum cost between the start point and endpoint. The global planner is built as a more flexible replacement of navfn with more options like support for A*, toggling quadratic approximation and toggling grid path.

4.1.2 Local Planner

The local planner creates new waypoints considering the dynamic obstacles and the vehicle constraints to transform the global path into suitable waypoints. So, the map is reduced to the surroundings of the vehicle and is updated as it is moving around to recalculate the path at a specific rate. It's not possible to use the whole map as the sensors cannot update the map in all regions, and a large number of cells would increase the cost of computation. Therefore, the local planning generates avoidance strategies for dynamic obstacles with the updated local map and the global waypoints and tries to match the trajectory as much as possible to the provided waypoints from the global planner.

The most commonly used local planners for mobile robot applications are base_local planner and DWA planner (dynamic window approach).

The basic idea of these local planner algorithms is as follows:

- Sample the robot's control space (dx, dy, dtheta) discreetly.
- For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some time. This

may be done for the entire forward simulation or limited steps.

- Evaluate each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal or impossible trajectories.
- Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
- Rinse and repeat.

Base local planner

The base_local_planner package provides a controller that drives a mobile base in the plane. This controller serves to connect the path planner to the robot. It is also known as Trajectory Rollout algorithm. Using a map, the planner creates a kinematic trajectory for the robot to get from a start to a goal location. Along the way, the planner creates, at least locally around the robot, a value function, represented as a grid map. Trajectory Rollout algorithm samples from the set of achievable velocities over the entire forward simulation period given the acceleration limits of the robot. This value function encodes the costs of traversing through the grid cells. The controller uses this value function to determine dx, dy, dtheta velocities to send to the robot.

DWA planner

The dwa_local_planner package provides a controller that drives a mobile base in the plane. This controller serves to connect the path planner to the robot. Using a map, the planner creates a kinematic trajectory for the robot to get from a start to a goal location. Along the way, the planner creates, at least locally around the robot, a value function, represented as a grid map. This algorithm samples from the set of achievable velocities over a small range, mostly one step forward simulation period, given the acceleration limits of the robot. This value function encodes the costs of traversing through the grid cells. The controller uses this value function to determine dx, dy, dtheta velocities to send to the robot.

Comparison

DWA differs from base_local planner in the way the control space of the robot is sampled. The base_local planner 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. This means that DWA is a more efficient algorithm because it samples a smaller space. However, it may be outperformed by Trajectory Rollout for robots with low acceleration limits because DWA does not forward simulate constant accelerations.

Result

However, in practice, we found that DWA planner performed better in most of our test cases than base_local_planner. This is due to the reason that the robot moves in a closed environment. Hence there is no requirement of entire forward simulation, which thereby increases the efficiency of the computation. Moreover, the robot is designed for considerably high accelerations and rarely reaches extremely low speeds. With these considerations in mind, the DWA planner is implemented as the local planner.

4.2 Recovery Behaviour

Recovery behaviour is the response of the robot when it encounters an obstacle so close that the robot is stuck or is hindered of the forward motion. This case usually occurs when a dynamic obstacle suddenly appears on the map. The predefined recovery behaviour of the robot for this was to stop and make a complete revolution around itself to check for possible clearance for movement and then generate an alternative path. It is unnecessary to check for all possible clearances for any such cases. Hence, the recovery behaviour was changed. Now, the robot directly generates a path from its location to the goal if sufficient clearance is present between the robot and the obstacle. Otherwise, it makes a rotation until it finds enough clearance to cross the obstacle. This reduces the time of travel when such cases occur, which are highly predominant in a closed indoor environment.

4.3 Artifacts in costmap - costmap clearing while stationary

The term research artifacts refer to the uncontrolled and unintentional systematic biases, that can threaten the validity of one's conclusions.

A problem with the cost map in 'RTAB' mapping is that when it detects an obstacle, it fixes it on the cost map permanently. So, even if the object is moved or removed, it still stays in the cost map. So, every time the program had to be restarted to clear the cost map of obstacles. To counter this, we made a recovery behaviour by resetting or clearing the cost map if the robot is stuck for a while, stopped or after reaching intermediate goals. This costmap clearing procedure is done by a separate node which runs independently it subscribes to the twist of the robot and goal status. Using this data, it calls the ros service to clear the costmap offered by the navigation stack.

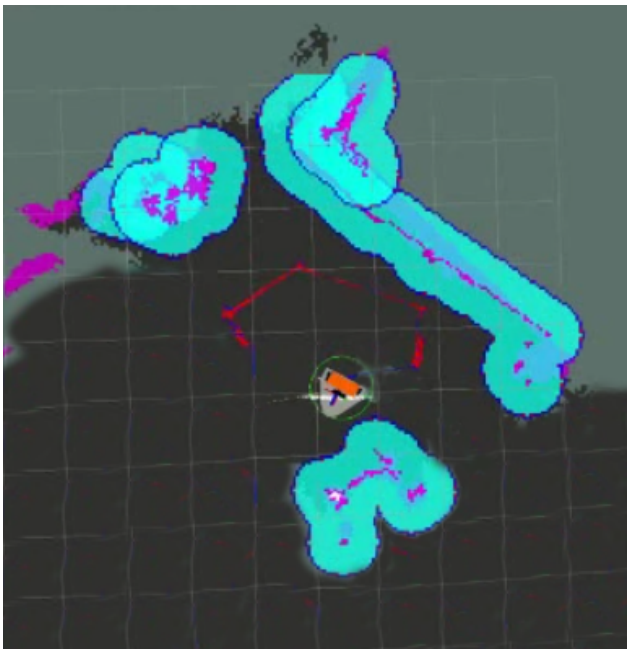


Figure 11: Cost map Artifact

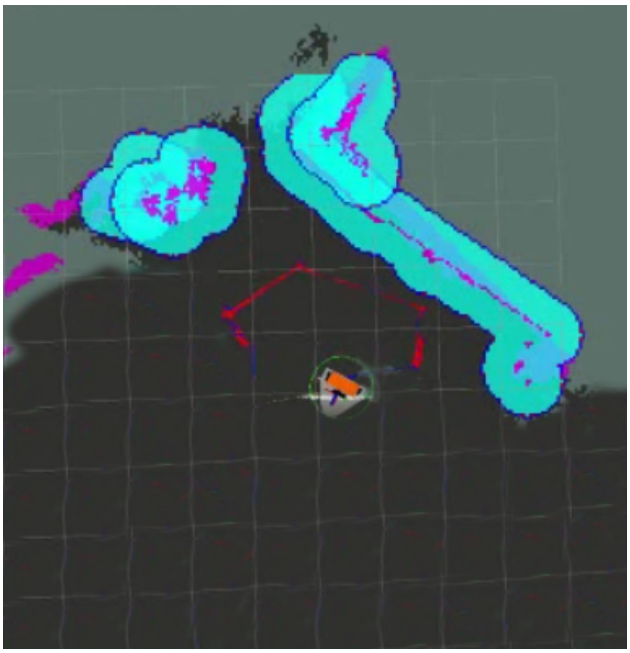


Figure 12: Cost map Cleared

5 Voice Control - PocketSphinx

5.1 Building a phonetic dictionary

A phonetic dictionary provides a mapping of vocabulary words to sequences of phonemes. A dictionary file robomuse.dic was created with a set of vocabulary words for starting the robot, stopping, goal points, docking, etc.

The recognizer searches for a word in both the dictionary and the language model and hence, for a word to be recognized, a language model file with all the words in the dictionary is created. Espeak is used to create the phonetic dictionary for the supported languages.

5.2 Building a language model

The language model is an essential component of the configuration which tells the sequences of words possible to be recognized by the decoder.

Several models include keyword lists, grammar, statistical language models and phonetic language models. They have different capabilities and performance properties.

5.3 Keyword lists

Pocketsphinx supports a keyword spotting mode whose advantage is that a threshold for each keyword can be specified so that keywords can be detected in continuous speech. Other modes try to detect the words from a grammar even if you used words which are not in the grammar.

The threshold must be specified for every key phrase. For shorter key phrases smaller thresholds like 1e-1, for longer keyphrases, the threshold must be larger, up to 1e-50 are set. For very long keyphrases (larger than 10 syllables) it is split and spotted for parts separately. The threshold is tuned to balance between false alarms and missed detections.

5.4 Grammar

A grammar describes a simple type of language for command and control. They are created with the Java Speech Grammar Format (JSGF) with a file extension .gram or .jsgf.

5.5 Language models

Statistical language models contain probabilities of the words and word combinations. Those probabilities are estimated from sample data and automatically have some flexibility. Every combination from the vocabulary is possible, although the probability of each combination varies.

5.6 Building a statistical language model - text preparation

An extensive collection of clean texts is made with expanded abbreviations, numbers converted to words and non-word items are cleared. A reference text that is used to generate the language model is prepared with input in the form of normalized text files, with utterances delimited by ¡s¿ and ¡/s¿ tags. A vocabulary file is then generated.

5.7 Using the language model for control of Robomuse 4.0

The language model created is then used for the control of Robomuse. The asr.launch file accesses the robomuse.dic dictionary file, robomuse.lm language model and asr.gram grammar file. The input, learning rule and hmm are set to default. The node asr_test.py is created to handle the jspf grammar node. This node is responsible for the control of the robot by calling the required launch files under robomuse_drivers. Another node send_audio.py is created for publishing audio inputs.

6 QT - User Interface

6.1 Python scripts

The python scripts are used for programming the effects of clicking buttons on the qt screen. The qt screen is generated from the design file which is of .ui form and can be generated form the qt designer software which allows drag and drop facilities to buttons and various other UI elements this allows for fast development. The python scripts inherently link to shell scripts which run other programs to run the robot.

6.2 Shell scripts

The shell scripts are used as timing elements and also used to call shell applications like espeak TTS service to allow the robot to say things. The essential function of the shell scripts is to time the launching of various nodes form the launch files. The timing facilitates the mitigation of errors while running and pipelines the entire process. The timing, however currently is done manually and times are measured for the proper launch of each file and time is set. However, this must be made such that it is synchronous in the future.

6.3 Launch files

Launch files have a collection of a logical cluster of nodes which have to be run together for a particular application. They also allow setting parameters for that particular applications so the same nodes can be run for different applications with different parameters. It allows for a greater extent of control and customization. This higher degree of customization and control can be leveraged for most of the operations. One example is the usage of the nodes of the RTAB map package where the nodes configured with different parameters can be used for the applications of mapping and then for navigation.

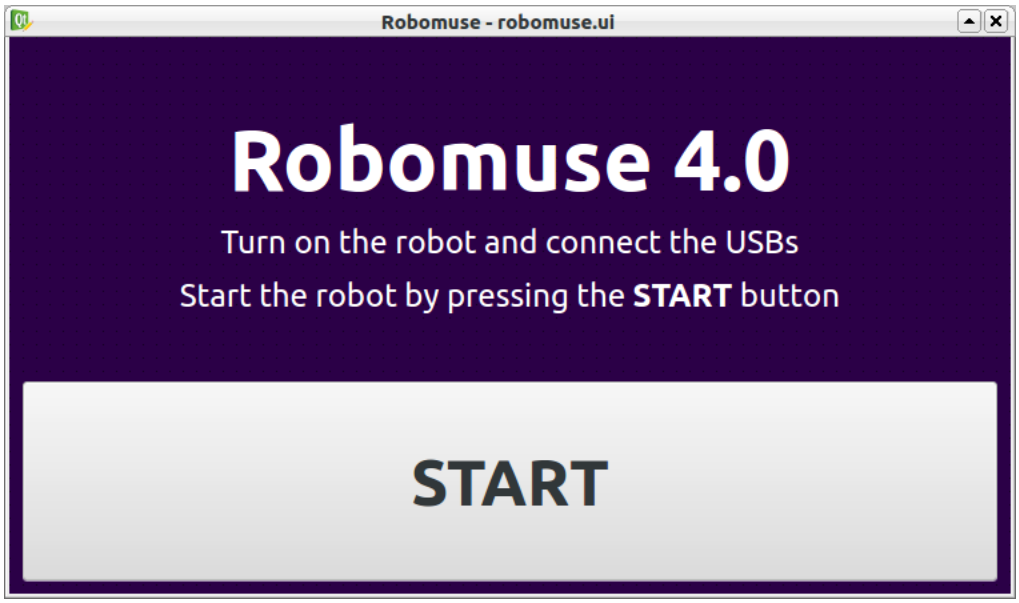


Figure 13: Start UI

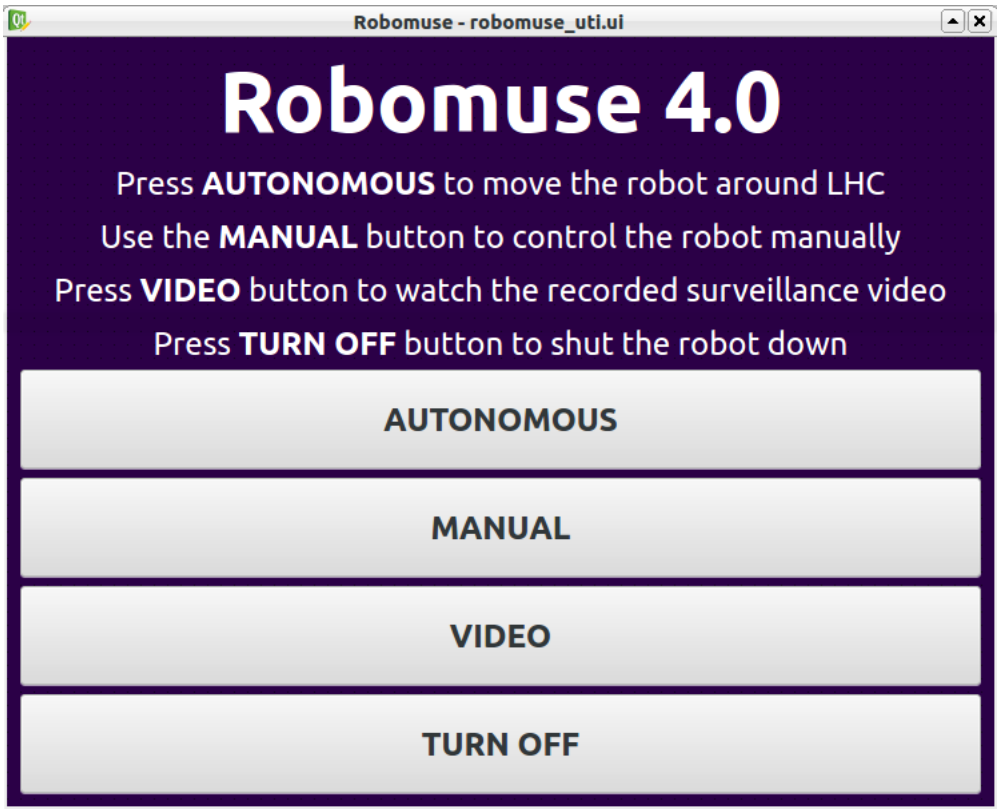


Figure 14: Utilities UI

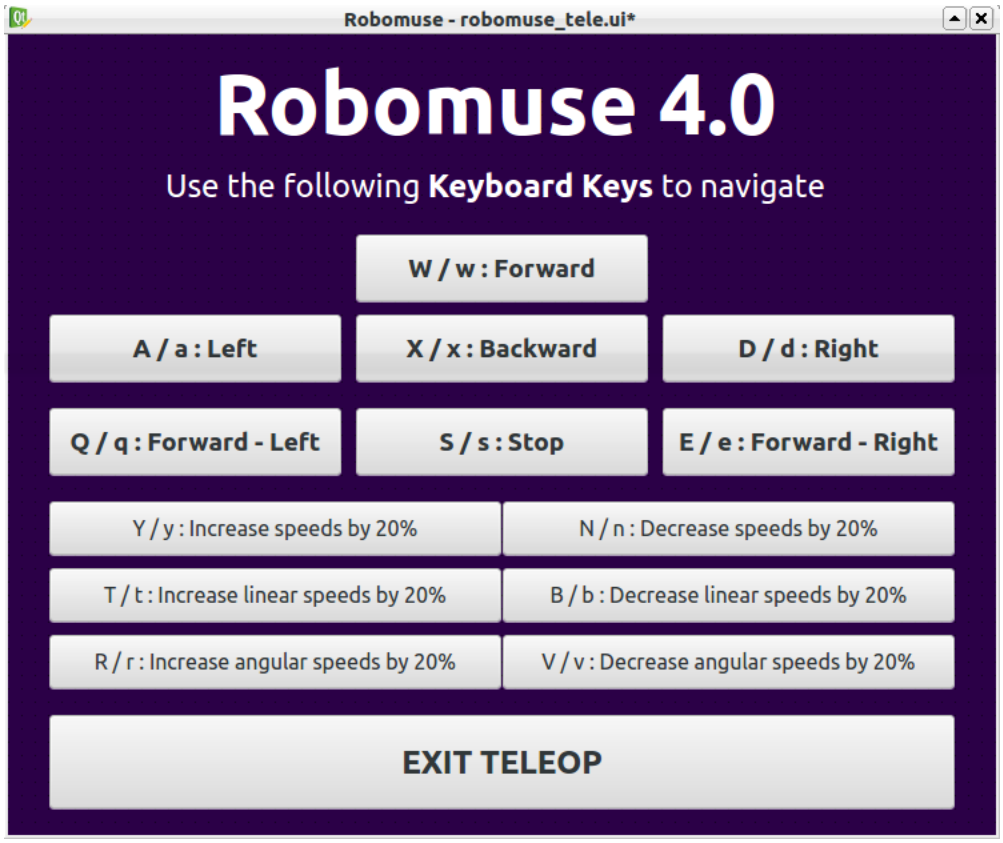
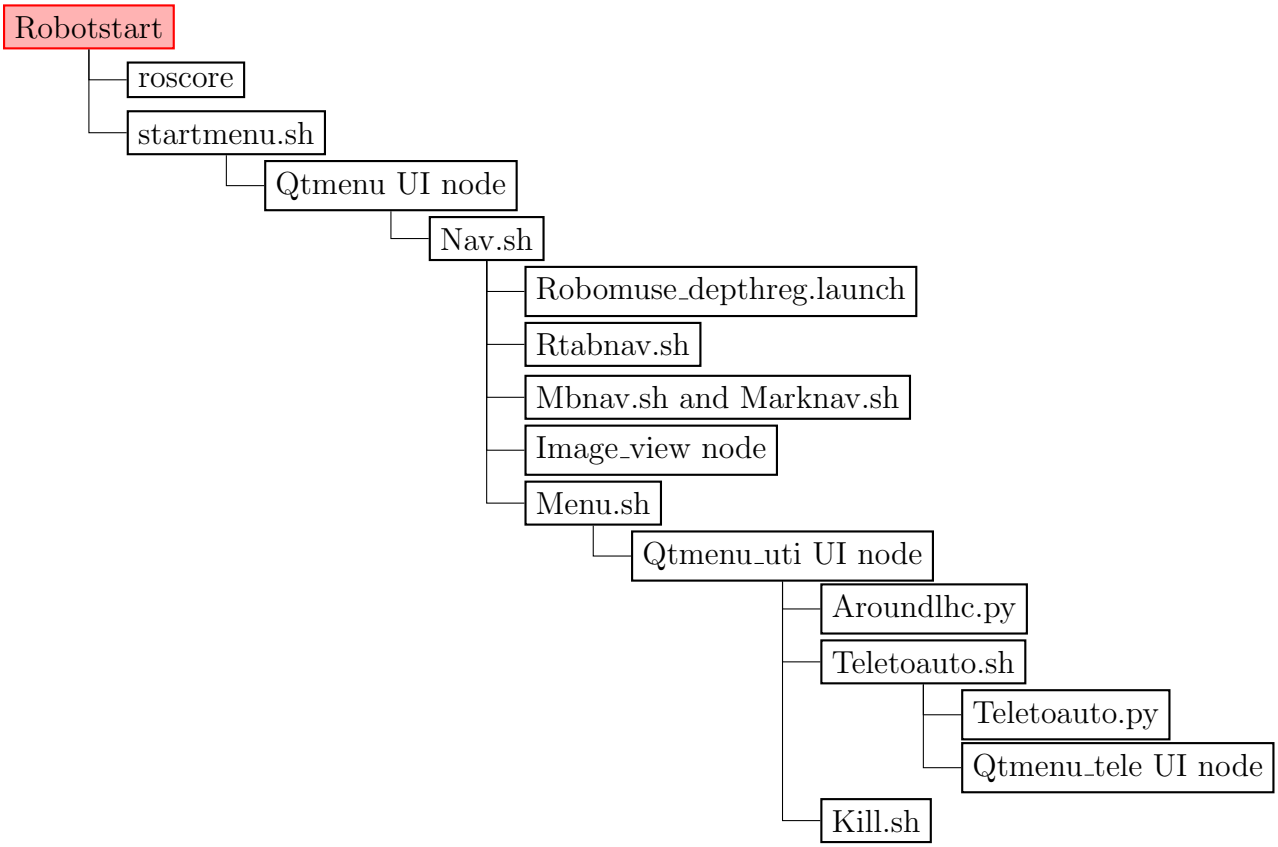


Figure 15: Teleoperation UI

This screen is launched when manual control is used. It is given for the guidance of the user about the control of the robot and teleoperation settings.

6.4 Hierarchical structure and explanation of each node



Robotstart - initial UI for starting the robot
roscore - starts the rosmaster and ensures ros runs
startmenu.sh - a shell script to start the robot and run other shells
Qtmenu UI node - (robomuse UI) launches the UI for the selection menu
Nav.sh - runs shell scripts and launch files required to configure hardware
Robomuse_depthreg.launch - sets up the Kinect and Arduino
Rtabnav.sh - sets up the slam
Mbnav.sh - sets up the navigation stack
Marknav.sh - sets up marker correction scripts
Image_view node - shows the camera feed
Menu.sh - launches the menu
Qtmenu_uti UI node - robokmuse_uti.ui
Aroundlhc.py - node sends waypoints to the robot
Teletoauto.sh - allows manual control scripts
Teletoauto.py - keyboard input
Qtmenu_tele UI node - robomuse_tele.ui
Kill.sh - kills all proceses to stop the robot

7 Conclusion

The work done over the summer saw the robot become more applicable in the real world. With more time and effort spent on the robot, the robustness of the system can be improved. Earlier odometry errors tend to pile up in the robot and it eventually gets lost. ArUco markers are implemented and hence the error is corrected on detection of a nearby marker. Furthermore, to improve this condition the *robot_localization* package can be implemented and IMU can be used for sensor fusion of the odometry. Currently, the mapping of the robot is done manually with teleop. This mapping can be automated in a restricted or closed boundary environment to enhance the purpose. Object detection with their positioning in the map can be done to make the map more comprehensive, interactive and practical. With these done the robot will be made much more robust and practically appealing. These would certainly enhance the platform for many other applications and research possibilities in Robomuse.

References

[1] ROS Wiki - http://wiki.ros.org/Documentation

[2] http://kaiyuzheng.me/documents/navguide.pdf

[3] https://docs.opencv.org/trunk/d5/dae/tutorial_aruco_detection.html

[4] https://cmusphinx.github.io/wiki/tutorialpocketsphinx/

[5] https://doc.qt.io/qt-5/qtdesigner-manual.html