Development of Low-Cost Education Platform: RoboMuse 4.0

Authors: XXXXX
University: XXXXX

Ever since the inception of Robotics, it served as a great collaborative platform for the researchers from the fields of mechanical engineering, electrical engineering, and computer science. Robot Operating System (ROS), one of the biggest middleware framework for robotics lead to high paced research and development around the globe. In this paper, we present our work on developing a low-cost ROS enabled education platform for Indian research institutes. This paper discusses our learning of ROS using youBot and development of indigenous platform RoboMuse 4.0.

1. INTRODUCTION

Robotics, ever since its inception have always been a major source of attraction for researchers and academics around the globe. Starting from Shakey[1] to PR2[2], it has always been an interdisciplinary field. With the enhancement of various perception and communication technologies, it became an increasingly difficult for a single developer to work on all the mechanical and electrical aspect of a robot. Robot Operating system changed it all with its inception in 2007. It became extremely easy for the developers to work and focus on a particular aspect of the field and integrate their work with others. The vast open source platform provided the much-needed interface to enable data sharing and allowed researchers to develop on platforms rather than reinventing the wheel. More importantly, ROS provided a simple way to integrate multiple packages and a state of the art communication between those packages. All these features also made ROS a go-to platform for the newcomers in the field of robotics. It was now much simpler to learn about the packages and integrate them to run your own robot within days.

Many robotics companies saw it as a great opportunity and developed several ROS enabled physical robotics platforms like Turtlebot 2, youBot, PR2, Baxter, Husky, etc. These platforms along with ROS attracted a wide range of researchers and students alike and led to a great boom of robotics in the west. Turtlebot 2, for instance, became one of the major robotics educational platforms around the world[3]. Owing to its low cost and great sensor modularity Turtlebot 2 helped researchers to develop and test several new ideas and concepts in a short period of time. The open source platform of ROS made it extremely easy for the students from numerous departments to operate and learn Turtlebot[4]. Hence, accelerating the process of new developments by avoiding the process of redevelopment of wheel. Over the years Turtlebot has been significantly used as an educational platform in undergraduate and graduate courses to develop hands on working skills in students. [5; 6]

DOI: To be inserted after paper is accepted

ROS since its development has not seen much of the development in India. The prime reason for which was the lack of indigenous ROS enabled platform. The high cost of these platforms in India restricted majority of Indian Universities to enjoy the vast benefits of ROS.

With RoboMuse 4.0, we took the opportunity of bringing *our university's* indigenous robot series on ROS and providing it with the same research capabilities as that of other platforms at a much cheaper price in India. RoboMuse 4.0 aims at allowing students to get started in the field and provides researchers with a tool to expand and develop its existing state of the art environment mapping technologies and numerous other features as discussed in the following sections.

This paper is structured as follows: Section 2 gives a general introduction to ROS and the hardware setup used. In section 3, details the implementation of autonomous navigation on KUKA youBot using ROS packages and external sensors like Kinect. In section 4, we talk about the development of ROS enabled RoboMuse 4.0. Section 5 details the development of autonomous navigation on RoboMuse 4.0. Finally, Section 6 presents our conclusions.

Fig. 1: Robomuse 4.0

2. ROS

Robot Operating System (ROS) is an environment that facilitates the development of robotic applications. It includes libraries and tools which provide hardware abstraction, device drivers, visualizers, message-passing, etc. Programs are built as ROS nodes, which connect to a single ROS master. The ROS master is connected to other ROS nodes as slaves. Every node connected to this master can listen to the messages provided by other nodes by simply subscribing to the corresponding topics. In addition to messages, parameters and services can be made available for all the nodes connected to the master. ROS is widely used in programming educational/research robots in addition to many industrial robots. Own-

ing to its open source and modular platform, multiple packages on ROS are being developed for a large number of different platforms like PR2, Baxter, Husky, etc. The key feature that ROS holds within it is the ability to circumvent the task of reinventing the wheel for developers, thereby creating an environment of constant development. Since most of the packages for various functions have already been developed, our primary task was only to understand how these packages function and change them according to our needs and hardware.

2.1 KUKA youBot

The KUKA youBot is an educational robot that has been specifically designed for research and development in mobile manipulation. It consists of an omnidirectional mobile platform, a five degree-of-freedom robot arm, and a two-finger gripper. The KUKA youBot can be controlled using a wide range of open-source software packages as well as other software (C++ API, ROS, Orocos, LabView and much more) [7]. The open interfaces additionally allow users to equip the KUKA youBot with sensors for complex applications.

2.2 ROS on KUKA youBot

In order to operate youBot with ROS, it is necessary for the robot to communicate it states configuration, i.e., odometry, joint angles, etc., with the system. The KUKA youBot API wrapper provides a mapping between messages for the ROS framework and the KUKA youBot API method calls [8]. Using this wrapper, we were able to control the base and arm of the robot. The proprioceptive sensor measurements, i.e., the odometry and joint angles of the KUKA youBot arm and wheels are published on ROS topics. These sensor measurements empower the robot with the knowledge of its own configuration.

2.3 Microsoft Kinect

Kinect is a motion-sensing camera by Microsoft which provides RGB and depth data of the environment. Though capable of working only in the closed environment, it is the most widely used sensor for indoor environment mapping. We used a head-mounted Kinect for 3D SLAM and later for object recognition. The technical specifications of the Kinect have been mentioned in Table 1. Kinect

Resolution	640x480
Frame Rate	30 fps
Horizontal FOV	57^o
Vertical FOV	43^o
Minimum Range	$\approx 0.5m$

Table I. : Technical Specifications of Kinect v1

generates depth data from the technique of analyzing a known pattern, called structured light. According to this technique, a known pattern is projected onto the scene and depth data is inferred from the deformation of this pattern. The Kinect uses infrared laser light, with a speckle pattern to perform depth analysis [9]. It does not use its RGB camera for depth computation.

2.4 ROS on Kinect

In order to interface Kinect through ROS, we used the OpenNI package. This package contains launch files for using OpenNI-compliant devices such as the Microsoft Kinect in ROS. It creates a nodelet graph to transform raw data from the device driver into point clouds, disparity images, and other products suitable for processing and visualization. This enabled us to access the point cloud data and camera feed using ROS messages.

2.5 Simulation on ROS

ROS provides an integrated platform for robot simulation. Robot model along with the sensors can be easily integrated with gazebo simulator in ROS to generate a communication thread between the user and simulator analogous to the one between a real robot and user. Since the simulator mimics the exact same messages generated by our hardware it has an added advantage of developing packages without having the real platform. Each package which we will talk about was first tested on the simulator for the safety of the robot and only after obtaining expected results from the simulation were the packages transferred on to the actual hardware.

3. AUTONOMOUS NAVIGATION ON YOUBOT

3.1 Obstacle Avoidance

Before we discuss obstacle avoidance, it is necessary that we introduce some terminologies:

Footprint: The footprint of the robot is the circle which circumscribes it. Kinect provides depth values only greater than 500 mm. So, to ensure that no obstacle comes within this range, the footprint value was set to 800mm (500 + 300, where 300mm is half of youBot's length).

Costmap: Costmap is a 2D map of the environment which contains information of the obstacles in the form of an occupancy grid. Each point in the map has a cost value assigned to it.

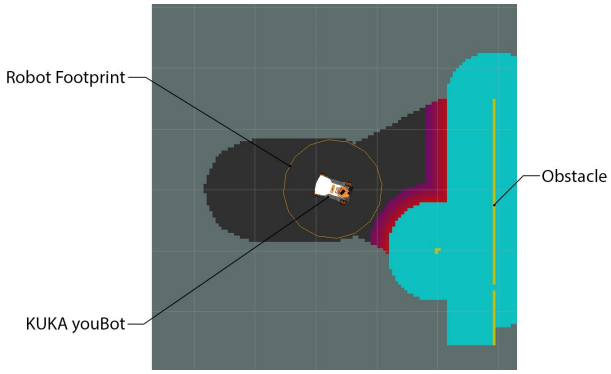


Fig. 2: Local Costmap

—The yellow areas in the map represent the obstacles. These are assigned a cost of 254.

—The blue areas represent the obstacles inflated by the radius of the robot. These regions are assigned a cost of 253.

—The gray areas represent free space and thus, are assigned a cost of 0.

—The red/pink areas have a cost between 0 and 253.

The costmap is further divided into two categories:

Local Costmap: The local costmap (Fig. 2) is the costmap of the environment which is currently in view of the robot. This is the map which is used by the robot for obstacle avoidance.

Global Costmap: The global costmap is generated by combining successive local costmaps along with localisation information. It contains the cost information of the whole map. This map is used for path planning.

For the purpose of obstacle avoidance, the nav2d package was used. This package requires a 2D laser scanner data for generating the costmaps [10]. Since one of the priorities of this project was to achieve all the objectives at an affordable price, buying/using a laser scanner was not an option. Thus, it was decided to emulate the laser scanner data using the depth data from the Kinect. Obstacle avoidance was achieved on the robot by moving in a path which minimizes the value of the cost, while still maintaining the direction as much as possible. The robot continues to move on its path until it enters a region with a non-zero cost (red areas). Once it is inside a red area, the robot moves in the direction in which the cost decreases while the deviation from the path is minimum. For the robot to avoid a collision, its footprint should never intersect with an obstacle and thus, the center of the robot should never enter the blue region.

3.2 Simultaneous Localization and Mapping

Simultaneous Localisation and Mapping (SLAM) is the process constructing a map of an unknown environment while simultaneously keeping track of the robot's location within it. This is a convoluted problem since in order to solve one we need to know the solution to the other. There are several algorithms which can be used to reach an approximate solution like the particle filter and the extended Kalman filter. A key feature in SLAM is detecting previously visited areas to reduce map errors, a process known as loop closure detection [11]. We worked with graph-based SLAM approaches that use nodes as poses and links as odometry and loop closure transformations.

3.2.1 *Real Time Appearance-Based Mapping.* the rtabmap package is an ROS wrapper of RTAB-Map (Real-Time Appearance-Based Mapping), an RGB-D SLAM approach based on a global loop closure detector with real-time constraints [12]. This package can be used to generate an RGB-D map of the environment (Fig. 3) and to create a 2D occupancy grid map for navigation [13].

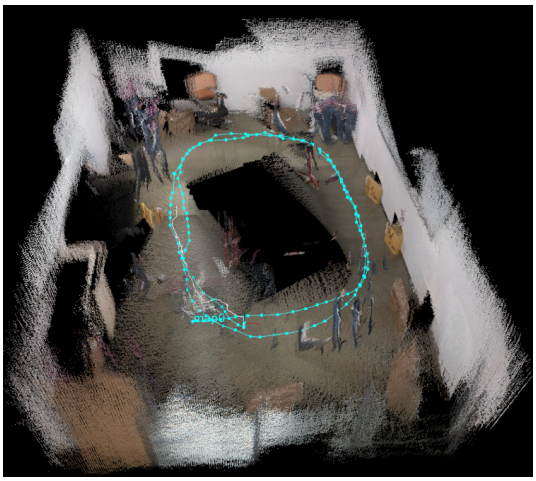


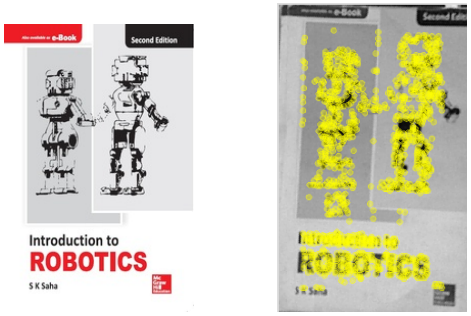
Fig. 3: 3D Reconstruction of Lab showing path of the robot

3.2.2 *Loop Closure.* Loop-closure detection is associated with the problem of detecting when the robot has returned to a past location after having discovered new terrain for a while. Such detection makes it possible to increase the precision of the actual pose estimate. Recognizing previously mapped locations can also be relevant for addressing the global localization problem [14]. For most of the probabilistic SLAM approaches, loop closure detection is done locally, i.e., matches are found between new observations and a limited region of the map, determined by the uncertainty associated with the robots position. Appearance-based loop closure detection approaches generally detect a loop closure by comparing a new location with all previously visited locations, independently of the estimated position of the robot. If no match is found, then a new location is added to the map. Typical loop closure methods apply a second algorithm to measure some type of sensor similarity, and re-set the location priors when a match is detected. For example, this can be done by storing and comparing bag of words vectors of Scale-Invariant Feature Transform (SIFT) features from each previously visited location.

3.3 Object Recognition

Object recognition was achieved using the find_object_2d package for ROS. This package is an interface to OpenCV implementations of SIFT, SURF, FAST, BRIEF and other feature detectors and descriptors for objects recognition. We begin with a sample of images of the object to be located. The task of object recognition can then be divided into the following steps:

Feature Detection: In this step, abstractions of image information are computed. We choose local points in the image that have an interesting or distinguishing property and these are called features (Fig. 4). In this project, these points are identified using the Features from Accelerated Segment Test (FAST) algorithm. As the name suggests, FAST is a computationally efficient algorithm for feature detection which can be used in real-time applications such as SLAM.



(a) Known Object (b) Features detected

Fig. 4: Feature Detection using BRIEF

Feature Description: Once the interesting points in the image were identified, a local image patch around the feature was extracted. The result is known as a feature descriptor. The description can be done in a number of ways. In this project, the Binary Robust Independent Elementary Features (BRIEF) algorithm was used. BRIEF is based on comparisons. From a patch of interesting points, we chose two points and compared the intensities of those two points. If the first point was larger than the second point, we assign the value 1, else 0. This was done for a number of pairs

and we ended up with a string of boolean values. We repeated this process for each interesting point detected and therefore, for each interesting point, we got a string of boolean values.

Feature Matching: Once we have the feature descriptors of the objects to be identified, we tried to match these with those feature descriptors of the current image frame from the Kinect camera.

After the object has been identified in an image frame, its relative pose was estimated using the depth data of the pixels of the object in that frame [15]. This was combined with the localisation information from the SLAM approach to get an approximation of the pose of the object in the map.

3.4 Motion Planning

Once the pose of the object was identified in the map, the final step was to move the robot to that position. The move_base package was used to accomplish this task. This package contains the move_base node which links together a global and a local planner to accomplish the global navigation task. This node maintains two costmaps, a global costmap for the global planner, and a local costmap for the local planner. This node also provides a motion controller which acts as an interface between the path planner and the robot. Using a given map, the global planner creates a kinematic trajectory for the robot to get from a start pose to a final goal pose. Along the way, the local planner creates, around the robot, a value function represented as a grid map. This value function encodes the costs of traversing through the grid cells. The job of the controller is to use this value function to determine the differential velocities: dx, dy and $d\theta$ to send to the robot.

4. DEVELOPMENT OF ROBOMUSE 4.0

4.1 Mobile Base Design and control of RoboMuse 4.0

The methodology used to develop RoboMuse 4.0 can be broadly classified into 3 categories, namely, mechanical, electrical and programming aspects. The mechanical part included the design, analysis and fabrication of the robot. The electrical aspects included the circuit connectivity for the whole robot, motor controlling, sensor wiring, etc. The programming aspect included visual servoing through ROS, collision detection algorithms, etc.

4.1.1 *Design and Analysis.* Before the design phase, the concept of RoboMuse 4 was decided after brainstorming a number of ideas. In the design phase the chassis and the rack were designed in SolidWorks and then load analysis was performed through simulation.

4.1.2 *Chassis.* The base of the robot has been designed based on the dimensional constraints of the actuators (motors) and the tasks to be performed by the robot. The chassis of the base was designed in Solidworks as shown in Fig. 5. Simulation was carried out for deflection and Von Mises stress. The maximum deflection turned out to be in the order of 10^{-2} mm and the maximum stress was around $10^7 N/m^2$ which is much less compared to the Yield Stress of Aluminium ($4x10^8 N/m^2$).

4.1.3 *Drive.* The drive includes the motor, gear box, helical coupler, bearings, shaft, keys and the wheel, as shown in Fig. 6. The gearbox is attached to the motor at one end and a shaft at the other end. The shaft of the motor is linked to that of the wheel by means of a flexible helical coupler. The helical coupler provides flexibility deflections of the shafts and also transmits power efficiently. The wheel is supported by a shaft which in turn are supported by a set

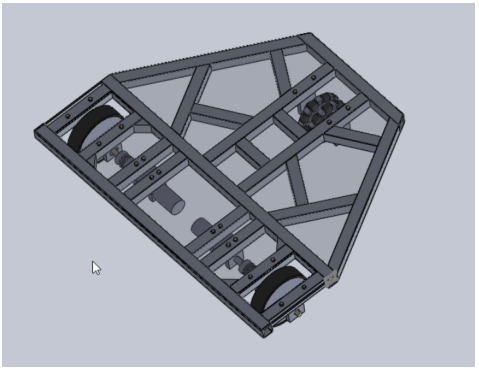


Fig. 5: CAD model of RoboMuse 4 chassis

of bearings at both the ends. The shaft and the wheel are connected with the help of a key to transmit power. A pair of retaining rings was used to prevent the axial motion of the wheel over the shaft.

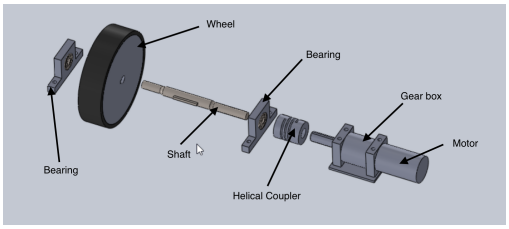


Fig. 6: Exploded view of drive

4.1.4 *Rack.* The main purpose of a rack was to equip the electrical equipment in two levels (bottom most), the laptop or Kinect in the uppermost layer and any objects to be transported or stored in the 3rd highest layer. The rack was designed to fit in the space above the drive. The rack was designed to be easily removable with respect to the chassis. It consists of four aluminium channels at the 4 corners and acrylic sheets as levels. These are shown in Fig. 7. The maximum height of the rack was such that a laptop can be placed on it and it was easily accessible by a human being of height 180 cm. Acrylic sheet of thickness 5mm was used for the different levels of the rack. The levels can be adjusted to any height within the limits of the channels. The rack was designed in Solidworks.

4.2 Electrical Circuits

The electrical circuit design determines the receptiveness of the robot. The simplicity and accessibility of the electrical components will ease the process of integrating new sensors and debugging if needed. We used open source platforms to make it convenient for others to learn and implement new elements into the already existing one. To ensure the safety of the robot and their surroundings an emergency switch was mounted on the robot for completely stopping the robot.

4.2.1 *Power Board.* We have incorporated three 12V lead acid batteries to power all the components of the robot. The power board ensures the desired voltage output was generated as different devices need different voltages. Motors were running on 24 V

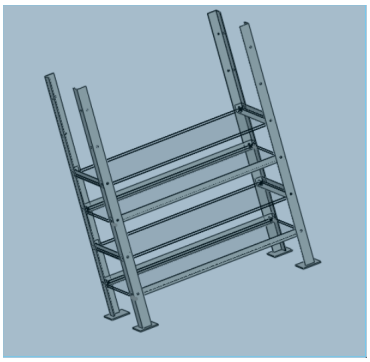


Fig. 7: CAD model of the rack

whereas the microcontrollers and motor drivers needed a 5V operating voltage.

4.2.2 *Micro-Controller.* The core of the circuit was an Arduino ATmega2560 microcontroller with 5V operating voltage. It is an open source hardware that can be programmed with the Arduino Software (IDE). Integrated with 54 Input-Output pins, it can easily incorporate multiple sensors and motors at the same time.

4.2.3 *Motor Driver.* To control the speed of motors it was essential to generate Pulse Width Modulated (PWM) signals which were generated by Sabertooth dual-motor driver with 25A continuous, 6-30V nominal capacity. Sabertooth allowed us to control two motors with analog voltage, radio control, serial and packetized serial. Sabertooth has independent and speed+direction operating modes, making it the ideal driver for differential drive robots.

5. AUTONOMOUS NAVIGATION ON ROBOMUSE 4.0

The process of automating a custom made robot platform and providing it with the functionalities that were implemented on the Youbot is bound to take several months. This is where the ROS framework played a major role. The transportability and reusability of code allowed us to establish the complete system in a matter of 3 weeks. The ROS framework as mentioned earlier allows the implementation of a very neat method of communication between the different packages, sensors and actuators. Therefore, upon establishing this communication on the Kuka Youbot, all we had to was shift the system to the custom made platform, amend and append a few things due to the differences between the robots on a hardware level.

From the developers perspective, when you are working with an Industrial Robot (Youbot), certain aspects of the robot are taken care of by the manufacturers, these include providing the buyer with the robot model ,the drive mechanism, the necessary software required to drive the robot and the procure the encoder data. But since Robomuse 4.0 is a custom made robot, we can not enjoy these luxuries.

5.1 Robot Modelling in ROS

Thus to start off with, we need to create the robots model in a format that is understandable by a variety of visualisation softwares (RVIZ, Gazebo). This format is referred to as the Unified Robot Description Format (URDF). This URDF makes the robot aware of the various links it possesses and as the robot moves how the links react. For eg. In a mobile robot, even though the center of the robot might be 50cm away from an obstacle, one end of the robot could be colliding with the obstacle. This is something that is undesirable, therefore to prevent such mishaps we create the URDF of the robot. And another advantage of creating the URDF is that, the robot could be simulated in a virtual world, thereby you can work with the robot and understand its behaviour to a particular code you have written without having the necessity to work with the hardware. Thus with the wide range of parameters that are required for the creation of the URDF you will be able to create quite an accurate model of the robot.

Unified Robot Description Format. The URDF for robot model description. This package contains a number of XML specifications for kinematic and dynamic properties of links and joints, sensors, simulation properties like friction etc. [16]. Extensible Markup Language (XML) is a markup language that defines a set of rules for the development of documents in a format that is understandable by both humans and machines. Thus, the URDF enabled us to articulate the model, add in a few physical properties and simulate in Gazebo.

Transforms. ROS uses the concept of a transform tree, which is a graph consisting of links as nodes and joints/tranforms as edges. Using this methodology, the robot keeps track of the position and orientation of the various links and frames it possesses. One can easily compute the transform from a given link to other by traversing in the graph and multiplying the transforms that one goes through. Many ROS packages require the transform tree of a robot to be published using the 'tf' software library. The 'tf' package contains a node called 'robot_state_publisher' which takes joint positions as inputs and keeps generating/updating the transform tree. So, we had created a node which takes input from sensors and publishes the readings as joint positions.[17]

Links. A link is a physical structure like a cuboid, cylinder or any complex shape. It has three important tags - Visual, Inertial and Collision tags. These tags help in making the model look and behave exactly like the physical link. This way the functioning/motions of the link can be tested without actually having to create it.

Joints. A joint is a connection between two links. URDF allows you to define a variety of joint types, including revolute, fixed, continuous, etc. To define a joint, it is essential that you have a parent and a child link (Thus we can define the tree). The parameters that are required for a joint varies with the type of the joint.

Visualization. Out of the many simulation and visualization softwares supported in ROS, we worked with Gazebo and RVIZ. Both of these have the ability to display the structure of a robot as described in an XML URDF file. Additional tags defining the friction between surfaces, inertia etc have to be added to ensure that the URDF file is compatible with Gazebo.

5.2 DRIVER

A Robot driver in software terms is that piece of code which enables you to drive your robot using the available hardware. Since the driver for the Youbot was already existing there was no requirement to create one. But however since the Robomuse is a custom made robot, every piece of software is to be created by us. So ultimately the driver should be capable of listening to the velocity commands provided to it, by the various packages that are running

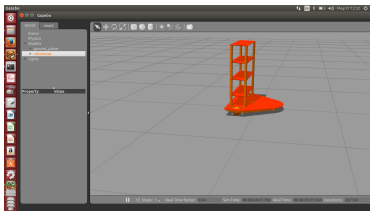


Fig. 8: URDF Model of Robomuse 4.0 in Gazebo

simultaneously and ensuring that the robot moves with the assigned velocity.

Hardware. To understand the driver we need to initially understand the hardware that drives the robot. The Robomuse 4.0 was driven by a Sabertooth Motor Driver attached to a Kangaroo X2 motion controller. The Kangaroo provides a self-tuning feedback motion control to the Sabertooth. Quadrature AMT11 encoders were mounted on the motor shafts which provided feedback signal to the Kangaroo (to help maintain the velocity) and a Micro-Controller which transferred the data to the ROS system. After appropriate tuning, velocity commands were sent to Kangaroo via Serial Protocol through the microcontroller. It was able to control motors through the kangaroo.

Thus a closed loop was established in the following fashion: Encoder data (robot) → Arduino → Data processing from system → Velocity Command from system → Arduino → Kangaroo (robot).

ROS Serial. ROS provides a package *ROS Serial* for interfacing of the micro-controllers with ROS. Micro-controllers can act as ROS nodes, publish and subscribe to ROS topics. Additional libraries also need to be compiled in the Arduino code for it to act like a ROS node. The Arduino acted as a node, which took velocity commands from the ROS system and transferred it to the Kangaroo, while simultaneously transferring the encoder data to the ROS Master running on the PC to calculate the robot's position.

Data Processing - ROS. Two ROS topics, namely *'tf'* and *'odom'* hold information regarding the odometry of the robot. The data received from the encoders is converted to a form which is acceptable by these topics. *'tf'* topic requires data in the form of quaternions while *'odom'* requires the data of the form *geometry_msgs/TransformStamped*. For sending velocity commands to the robot, the topic *'cmd_vel'* was used, which uses messages of type *geometry_msgs/Twist* containing 6 DOF information of both linear and angular velocity. All the nodes which provide path planning sends commands on this node. The node running in the Arduino takes these velocity commands and provides velocity to the wheels through the Kangaroo, thereby making the robot move forward or take a turn.

6. CONCLUSIONS

6.1 Education Platform

Now that the RoboMuse 4.0 has been established with the basic necessities that are required for any mobile robot, it's use as an educational platform is immense. various algorithms and concepts that are associated with mobile robots can be implemented on the robot with minimal effort. The concentration of the students can only be limited to the algorithms and specific tasks rather than the establishment and the mobility of the robot, thereby increasing their efficiency and reducing their effort yet providing them with the knowledge they aimed to acquire.

6.2 Future Prospects

In the future, we expect the Kuka Youbot and the RoboMuse 4.0 to work in coordination with one another. We could replicate an industrial situation by allowing the Youbot to pick a specific object from it's workspace and and place it on the rack of the Robomuse 4.0 after successfully navigating through it's environment. The RoboMuse could deliver this object to a specific location, as instructed. Thus the developed robot system upon further work could help automate an industry.

REFERENCES

SRI International. (n.d.). Retrieved February 06, 2017, from https://www.sri.com/work/timeline-innovation/timeline.php?timeline=computing-digitalinnovation=shakey-the-robot

Overview. (n.d.). Retrieved February 06, 2017, from http://www.willowgarage.com/pages/pr2/overview

TurtleBot 2. (n.d.). Retrieved February 06, 2017, from http://www.turtlebot.com/

Wiki. (n.d.). Retrieved February 06, 2017, from http://wiki.ros.org/Robots/TurtleBot

Boucher, Sol. "Obstacle detection and avoidance using turtlebot platform and xbox kinect." Department of Computer Science, Rochester Institute of Technology 56 (2012).

Claessens, Rik, Yannick Mller, and Benjamin Schnieders. "Graph-based Simultaneous Localization and Mapping on the TurtleBot platform." (2013).

Mobile manipulation in carry-on format, 2015, from http://www.youbot-store.com/

ROS Wrapper. (n.d.). Retrieved January 12, 2016, from http://www.youbotstore. com/wiki/index.php?title=ROS_Wrapper

Developing with Kinect for Windows. Retrieved from https://developer.microsoft.com/enus/windows/kinect/develop

Wiki. (n.d.). Retrieved March 23, 2016, from http://wiki.ros.org/nav2d

Cognitive Robotics at ENSTA :: Indoor Navigation. (n.d.). Retrieved May 12, 2016, from http://cogrob.ensta-paristech.fr/loopclosure.html

RTAB-Map. (n.d.). Retrieved April 12, 2016, from https://introlab.github.io/rtabmap/

Labbe, M., & Michaud, F. (2014). Online global loop closure detection for large-scale multi-session graph-based SLAM. 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. doi:10.1109/iros.2014.6942926

Newman, P., & Ho, K. (n.d.). SLAM-Loop Closing with Visually Salient Features. Proceedings of the 2005 IEEE International Conference on Robotics and Automation. doi:10.1109/robot.2005.1570189

Find-Object. (n.d.). Retrieved April 1, 2016, from https://introlab.github.io/find-object/

URDF Overview, Retrieved from http://wiki.ros.org/urdf.

Transforms, Retrieved from: http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF.