Development of a Low-Cost Education Platform: RoboMuse 4.0

Ayush Shukla
Indian Institute of Technology, Delhi
New Delhi, India
shuklaayush247@gmail.com

Rishabjit Singh
Indian Institute of Technology, Delhi
New Delhi, India
rishabjit@gmail.com

Rishabh Agarwal
University of Maryland College Park
Maryland, USA
rishabhagarwal880@gmail.com

Muhammad Suhail
National Institute of Technology
Tiruchirappalli, India
muhammadsuhail441@gmail.com

Subir K Saha
Indian Institute of Technology, Delhi
New Delhi, India
sahaiitd@gmail.com

Santanu Chaudury
Indian Institute of Technology, Delhi
New Delhi, India
schaudhury@gmail.com

ABSTRACT

Ever since the inception of Robotics, it has served as a great collaborative platform for researchers from the fields of mechanical engineering, electrical engineering, and computer science. Robot Operating System (ROS), one of the biggest middleware framework for robotics has 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 begins with our learning of ROS using KUKA youBot and later goes on to discuss in detail the development of the indigenous platform: RoboMuse 4.0 and its integration with ROS.

KEYWORDS

Mobile robotics, ROS, research platform

1 INTRODUCTION

Robotics has 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 task for a single developer to work on all the mechanical, electrical and computational aspects of a robot. ROS changed it all with its creation in 2007. It became extremely easy for developers to work and focus on a particular aspect and integrate their work with others. This open source platform provided the much-needed interface to enable data sharing and allowed researchers to build upon each other's work rather than going about reinventing the wheel. More importantly, ROS provided a simple way to integrate multiple packages and a state of the art communication system between those packages. Many robotics companies have developed several ROS enabled platforms like Turtlebot 2, youBot, PR2, Baxter, Husky, etc. Turtlebot 2 became one of the major robotics educational platforms around the world [3]. Owing to its low cost and sensor modularity, Turtlebot 2 helped researchers to develop and test several new ideas and concepts in a short period of time. The open source nature of ROS made it extremely easy for students from various departments to operate and learn Turtlebot [4]. Over the years, Turtlebot has been used as an educational platform in undergraduate and graduate courses to develop hands on working skills in students [5, 6].

In contrast to the west, ROS in India has not been widely adopted. The prime reason is the lack of an indigenous ROS compatible platform. The high cost of such platforms has restricted majority of Indian Universities to enjoy the vast benefits of ROS.

IIT Delhi has its own indigenous mobile robot series RoboMuse. RoboMuse 1 started as a line following mobile robot capable of moving from point to point with automatic charging. It was adopted from one of the robots built for of Robocon 2008 by the IIT Delhi team. The next iteration, RoboMuse 2 focused on improving the reliability and robustness of the same task by incorporating, white wooden straps for line following instead of plastic tapes which were getting damaged frequently. It was also endowed with improved circuitry for better charging. RoboMuse 3 was the same technology but with the task of picking up a plastic bottle and dumping it in a basket located at a distance. Functional videos of all the three versions are available on YouTube[7][8][9].

In RoboMuse 4.0 (Fig. 1), we aim to provide the same research capabilities as that of other commercial platforms but at a much cheaper price in India. RoboMuse 4.0 allows students to get started in robotics and provides researchers with a tool to expand on ROS's state of the art features and in-numerous packages, some of which have been discussed in the following sections.

This paper is structured as follows: Section 2 gives a gentle introduction to ROS and the hardware setup used. Section 3 describes the implementation of autonomous navigation using ROS packages. In section 4, we talk about the development of RoboMuse 4.0. Section 5 describes the process of setting up ROS on RoboMuse 4.0. Finally, Section 6 presents our conclusions.

2 ROBOT OPERATING SYSTEM (ROS)

ROS is an environment that facilitates the development of robotic applications. It includes libraries and tools which provide hardware abstraction, device drivers, visualizers, loggers, etc. Programs are built as ROS nodes, which connect to a single ROS master. Every node connected to this master can listen to the messages provided by other nodes by simply subscribing to the corresponding topics [10]. In addition to messages, parameters and services can also be

AIR '17, June 28-July 2, 2017, New Delhi, India

ACM ISBN 978-1-4503-5294-9/17/06…$15.00
https://doi.org/10.1145/3132446.3134902

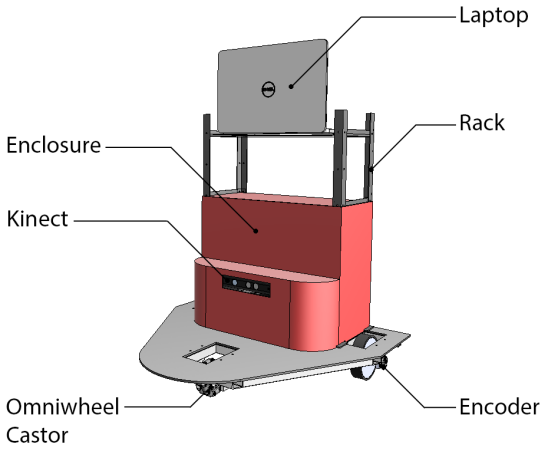


Figure 1: Robomuse 4.0

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. Owing to its open source and modular platform, packages are continuously being developed in ROS for a number of different platforms.

Since most of the packages for various functions had 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 on ROS

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 DoF robot arm, and a two-finger gripper. KUKA provides a ROS wrapper for youBot. Using this, robot's state configuration, i.e., odometry, joint angles, gripper position, etc. becomes available in ROS topics. Also, youBot's base and arm can be moved by publishing appropriate messages on the topics subscribed by youBot driver node [11].

2.2 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 one of the most widely used sensor for indoor environment mapping. Kinect was used for 3D SLAM and later for object recognition. The technical specifications of Kinect have been mentioned in Table 1.

Table 1: Technical Specifications of Kinect v1

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

Kinect generates depth data by analyzing the distortion of a known pattern, a technique called structured light. According to this, a speckle pattern of IR laser light is projected onto the scene and depth data is inferred from the deformation of this pattern [12].

2.3 ROS on Kinect

In order to interface Kinect through ROS, we used the OpenNI package. This package contains the necessary drivers required to convert raw RGB/IR streams from OpenNI compliant devices into depth registered point clouds. This enabled us to access the depth data through ROS topics.

2.4 Simulation in ROS

ROS provides an integrated infrastructure for robot simulation. Robot model along with the attached 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 USING ROS

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 had to be inflated by some fixed value.

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.

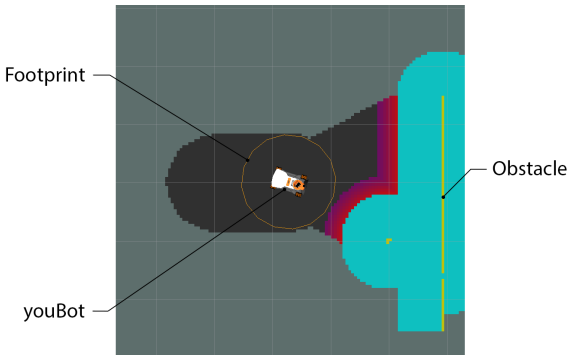


Figure 2: Local Costmap

- Yellow areas in the map represent the obstacles. These are assigned a cost of 254.
- Blue areas represent the obstacles inflated by the radius of the robot. These regions are assigned a cost of 253.
- Gray areas represent free space and are assigned a cost of 0.
- 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 [13]. This data was emulated using Kinect as a real laser scanner was an expensive option. 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. Further, the nav2d package enables you to maneuver safely by the use of parameters like inflation radius which is the distance around the robot in which the cost function is applied, thereby the robot tries to avoid any path falling within this region.

3.2 Simultaneous Localization and Mapping

Simultaneous Localisation and Mapping (SLAM) is the process of 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 [14].

3.2.1 Real Time Appearance-Based Mapping. The 'rtabmap_ros' package is a 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 [15]. This package can be used to generate an RGB-D map of the environment (Fig. 3) and projecting it to create a 2D occupancy grid map for navigation [16].

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. This detection increases the accuracy of the robot pose estimate. RTAB-Map uses a bag-of-words approach to determine whether the current view corresponds to a previously visited location or a new one [17].

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:

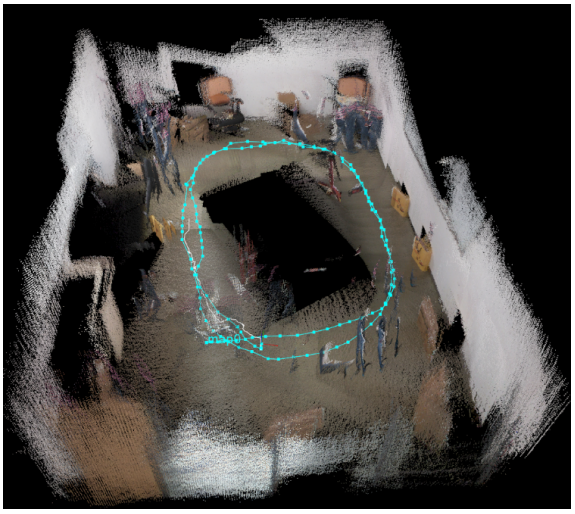
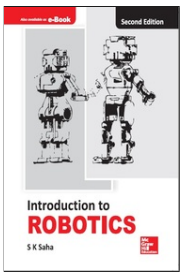
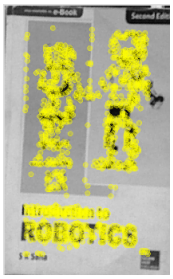


Figure 3: 3D Map Reconstruction showing path of the robot

Feature Detection: In this step, abstractions of image information are computed. We choose local points in the image that have an interesting/distinguishing property and these are called features (Fig. 4). In the work presented, these points are identified using the Features from Accelerated Segment Test (FAST) algorithm.



(a) Known Object



(b) Detected Features

Figure 4: Feature Detection using FAST

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. In the work presented in this paper, 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. This process is repeated for each feature point.

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 [18]. 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 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. ROS enables the attachment of the global and local planners as plugins, thereby enabling easy modification and implementation. Currently, the Dijkstra's algorithm is used to create the global plan while a Trajectory Roll Out Approach is used to create the local plan.

3.5 Integrating the ROS packages

Each of the different packages acted as a subsystem having distinct sets of inputs and outputs (Fig. 5). Thus, it was essential to integrate these subsystems appropriately to establish a single robust system having the required functionality.

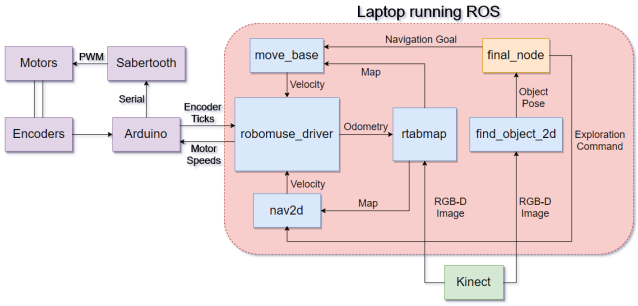


Figure 5: Subsystem Block Diagram

The SLAM subsystem (rtabmap) requires odometry from Encoders and RGB-D data from the Kinect as inputs. It generates both a 2D gridmap and 3D reconstructed map of the environment, along with the robot's pose as its output.

The Object Recognition subsystem (find_object_2d), requires the image of the object to be recognised along with real-time visual data from Kinect as inputs. It adds the pose of the object to the tf_tree if and when the object is identified in the current frame.

For the Motion Planning (move_base) subsystem, we require the environment map along with the goal coordinates as inputs. The job of this node is to provide a local and global motion plan along with publishing velocity commands as outputs. This enables the robot to move towards its goal while traversing around obstacles.

Finally, a ROS node (final_node) was written which acted as a master communicating with nav2d_operator, move_base, tf_tree, etc. to run the robot. The logic of this node is given in Fig. 6.

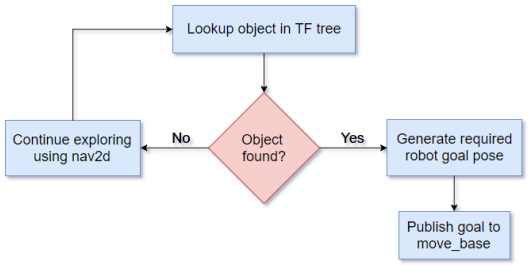


Figure 6: Algorithm of final_node

The final_node continuously looks for the object in the tf_tree. If the object is found, then the desired robot pose in front of the object is obtained using rigid body transformations, and a navigation goal is published to 'move_base'. If object is not found, exploration is continued by providing appropriate command to 'nav2d_operator'.

4 DEVELOPMENT OF ROBOMUSE 4.0

4.1 Mechanical Design

The development of RoboMuse 4.0 can be broadly classified into three categories: mechanical, electrical and programming aspects. The mechanical part included the design, analysis and subsequent fabrication of the robot. The electrical part consisted of designing the circuits, controlling motors, interfacing different sensors, etc. Finally, the programming aspect was the development and integration of different ROS packages, setting up the micro-controller, programming the logic, etc.

4.1.1 Chassis. The base of the robot (Fig. 7) was designed based on the dimensional constraints of the actuators (motors) and the tasks to be performed by the robot.

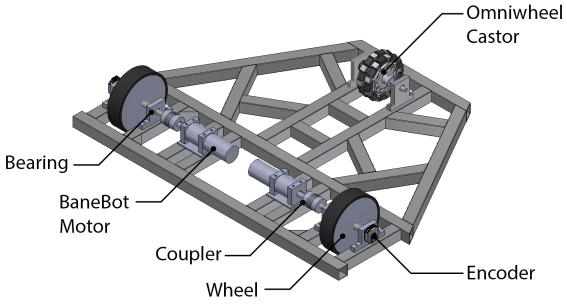


Figure 7: CAD model of RoboMuse 4 chassis

4.1.2 Drive. The drive assembly includes the motor, gear box, helical coupler, bearings, shaft, keys, wheel and encoder, as shown in Fig. 8. The gearbox is attached to the motor at one end and a shaft at the other end. The shaft of the motor is connected to the shaft of the wheel by means of a flexible helical coupler. The helical coupler provides flexibility to handle deflections of the shaft in addition to transmitting power efficiently. The shaft of the wheel is supported by a set of bearings at both ends. The wheel is attached to the shaft with the help of a key to prevent slippage while transmitting motion. A pair of retaining rings were used to prevent the axial motion of the wheel over the shaft [19].

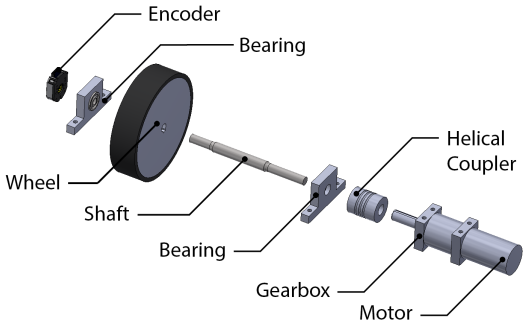


Figure 8: Exploded view of drive

4.1.3 Rack. The main purpose of a rack was to equip the electrical components in two levels at the bottom along with a laptop at the top shelf. It was designed to be easily removable from the chassis. It consists of four aluminium channels at the four corners and acrylic sheets as the shelves. The height of the rack was such that a laptop can be placed on it which is easily accessible by a person of height 180 cm. Acrylic sheet of thickness 5mm was used for the different shelves of the rack. These can be adjusted to any height within the limits of the aluminium channels.

4.2 Electronics

Emphasis was laid on making programming and debugging convenient for the robot users. Hence, we tried to incorporate only widely used open-source platforms. A brief description of the electrical sub-systems used is given below:

4.2.1 Power Supply. The robot is powered by two 12V 7.2Ah lead-acid batteries. One battery is used for driving the motors while the other to power the Kinect through a DC-DC Buck converter. The converter ensures that the Kinect input voltage stays exactly 12V even when the battery voltage is above 12.5V.

4.2.2 Micro-Controller. The core of the circuit is an Arduino ATmega2560 microcontroller operating at 5V from the laptop. It consists of 54 I/O pins including 6 external interrupts. The two Encoders are connected to four of these interrupts.

4.2.3 Encoders. CUI AMT11 Incremental Quadrature Encoders having resolution of 2048 PPR were mounted on the shafts of each wheel for odometry calculation.

4.2.4 Motor Driver. Motor speed control was achieved using PWM signals generated by the Sabertooth 2x32 dual-motor driver. It has a continuous current carrying capacity of 32A for both motors which was more than enough for our use. UART protocol was used to communicate between the Sabertooth and Arduino.

In addition to the above, an emergency-stop switch was also mounted on the top shelf for safety considerations.

4.3 Cost Analysis

Table 2: Bill of Materials

Part	Price(Rs.)
Body Manufacturing + Material	20,000/-
BaneBot Motors	12,000/-
Microsoft Kinect	5,000/-
Arduino Board	2,000/-
Sabertooth Motor Driver	9,000/-
CUI AMT Encoders	3,200/-
Batteries	1,500/-
PCB and misc. electronics	1,200/-
Total	54,200/-

Table 2 contains the cost of different components and the expenses occurred in realizing the robot. Turtlebot 2 with a laptop costs $2115 [20], which is nearly Rs. 1,40,000/- without customs and shipping. A similar setup for RoboMuse 4.0 would cost less than Rs. 90,000/- (including Rs. 35,000/- laptop).

It must be noted that this robot was made with expensive components which can be replaced with cheaper ones without affecting the functionality of the robot. For example, the size of the robot is excessively large, a smaller robot could be manufactured in under Rs. 12,000/-. Also, one could use cheaper motors, motor driver and encoders potentially reducing the cost of the complete robot down to Rs. 30,000/- from the current Rs. 54,000/-.

5 SETTING UP ROS ON ROBOMUSE 4.0

The packages required for autonomous robot navigation were first setup on KUKA youBot. Once we understood the working of these packages, the next step was to set up RoboMuse in ROS. This meant modeling the robot in ROS compatible format and developing drivers to interface with the robot hardware. After this was accomplished. the navigation packages were shifted to RoboMuse with minimal changes. The only parameters that had to be changed were dimensional specifications which are used for obstacle avoidance and kinematic specifications used for motion planning.

5.1 Robot Modeling in ROS

ROS uses Unified Robot Description Format (URDF) representation of robot for simulation and visualization in Gazebo/RViz. This URDF makes the robot aware of its own links and how they react with the motion of the robot. After going through the tutorials on ROS Wiki, an accurate URDF model of the robot was created.

Unified Robot Description Format. URDF file contains a number of XML specifications for robot model, sensors, etc. It also stores

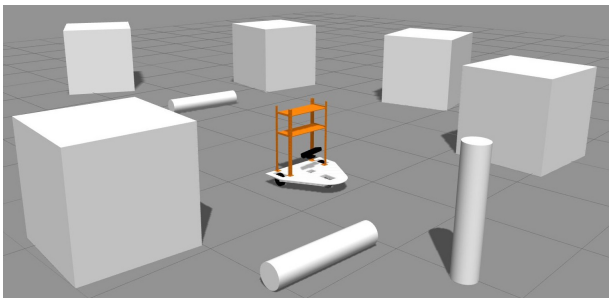


Figure 9: URDF Model of Robomuse 4.0 in Gazebo ROS

kinematic and dynamic properties of the links/joints in addition to simulation properties like friction, inertia, collision geometry, etc [21]. RoboMuse URDF model in Gazebo is shown in Fig. 9.

Transforms. ROS uses the concept of a transform tree for storing physical information. This is essentially a graph having links as nodes and joints as edges. The transform from one link to another can be easily computed by traversing this graph and successively multiplying the transforms that one goes through. The transform tree is published on the /tf topic through the 'tf' package. This package contains a node called 'robot_state_publisher' which takes joint positions as input and keeps updating the transform tree [22].

5.2 Driver

Driver is the low-level piece of code which interfaces the software with the robot hardware. RoboMuse driver had two primary tasks. First, it had to listen to the velocity commands provided by the ROS packages and ensure that the robot moves with the desired velocity. Second, it had to provide robot odometry using data from encoders.

Hardware. Following is the overall architecture of the robot: Encoder data (robot) → Arduino → Data processing from Laptop → Velocity Command → Arduino → Sabertooth Motor Driver. Two independent PID loops were implemented on Arduino to control the speed of each wheel.

Interfacing. ROS provides a package 'rosserial' for interfacing different micro-controllers (MCU) through a serial protocol. MCUs act as ROS nodes, publishing and subscribing to ROS topics. In RoboMuse, the Arduino acted as a ROS node, subscribing to velocity commands from the ROS system while simultaneously publishing the encoder data to the ROS Master running on the laptop, thus facilitating odometry calculation.

Data Handling in ROS. Two ROS topics, namely /odom and /tf store the odometry of the robot. The data received from the encoders is converted to ROS messages compatible with these topics. The /cmd_vel topic is used for sending velocity commands to the robot. Both nav2d_operator and move_base nodes publish commands to this topic at appropriate times. The node running inside the Arduino subscribes to /cmd_vel topic and subsequently provides signal to the motor driver to run the robot at the desired speed.

6 CONCLUSIONS

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 unbounded. Various algorithms and concepts associated with mobile robots can be easily implemented on the robot. Additionally, students can focus only on the algorithmic side of the tasks rather than worrying about the mobility and sturdiness of the robot. This would lead to increased efficiency and reduced effort for the students while still providing them with the knowledge that they aimed to acquire.

Robomuse 4.0 would be placed at IIT Delhi so that the students here can continue research in the field of mobile robotics. The next step would be to construct a smaller version of the robot having additional sensors which would enable it to traverse smaller environments. The target would also be to make it cheaper than the RoboMuse 4.0.

REFERENCES

[1] SRI International. Retrieved February 06, 2017, from https://www.sri.com/work/timeline-innovation/timeline.php?timeline=computing-digitalinnovation=shakey-the-robot
[2] Overview. (n.d.). Retrieved February 06, 2017, from http://www.willowgarage.com/pages/pr2/overview
[3] TurtleBot 2. (n.d.). Retrieved February 06, 2017, from http://www.turtlebot.com/
[4] Wiki. (n.d.). Retrieved February 06, 2017, from http://wiki.ros.org/Robots/TurtleBot
[5] Boucher, Sol. "Obstacle detection and avoidance using turtlebot platform and xbox kinect." Department of Computer Science, Rochester Institute of Technology 56 (2012).
[6] Claessens, Rik, Yannick Muller, and Benjamin Schnieders. "Graph-based Simultaneous Localization and Mapping on the TurtleBot platform." (2013).
[7] RoboMuse 1 [2008]. YouTube, 05 Aug. 2016. Retrieved from https://www.youtube.com/watch?v=81Cg3fyKex0
[8] Robomuse 2 2009. YouTube, 05 Aug. 2016. Retrieved from https://www.youtube.com/watch?v=OqucA-Bvf20
[9] RoboMuse-3 (2014). YouTube, 28 Jun. 2014. Retrieved from https://www.youtube.com/watch?v=oFzqNdUiiUg
[10] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.
[11] ROS Wrapper. (n.d.). Retrieved January 12, 2016, from http://www.youbotstore.com/wiki/index.php?title=ROS_Wrapper
[12] Developing with Kinect for Windows. Retrieved from https://developer.microsoft.com/enus/windows/kinect/develop
[13] Wiki. (n.d.). Retrieved March 23, 2016, from http://wiki.ros.org/nav2d
[14] Cognitive Robotics at ENSTA :: Indoor Navigation. (n.d.). Retrieved May 12, 2016, from http://cogrob.ensta-paristech.fr/loopclosure.html
[15] RTAB-Map. (n.d.). Retrieved April 12, 2016, from https://introlab.github.io/rtabmap/
[16] 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
[17] M. Labbe and F. Michaud, "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation," in IEEE Transactions on Robotics, vol. 29, no. 3, pp. 734-745, June 2013. doi: 10.1109/TRO.2013.2242375
[18] Find-Object. Retrieved April 1, 2016, from https://introlab.github.io/find-object/
[19] Agarwal, R., et al. "Touchless human-mobile robot interaction using a projectable interactive surface." System Integration (SII), 2016 IEEE/SICE International Symposium on. IEEE, 2016.
[20] Turtle bot 2 purchase, Clearpath robotics, Retrieved from http://store.clearpathrobotics.com/collections/robotics-kit/products/turtlebot-2
[21] URDF Overview, Retrieved from http://wiki.ros.org/urdf.
[22] Transforms, Retrieved from: http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF.