TurtleBot 2i Final Report

Spring 2017

Forrest Kaminski, Mohamed Abidalrekab, Sam Salin

Table of Contents

Introduction to TurtleBot

TurtleBot is a low-cost, personal robot kit with open-source software.The TurtleBot kit consists of a mobile base, 2D/3D distance sensor, laptop computer or SBC(Single Board Computer), and the TurtleBot mounting hardware kit. In addition, users can mount an Arm or any other sort of extra hardware to perform additional task using the TurtleBot.

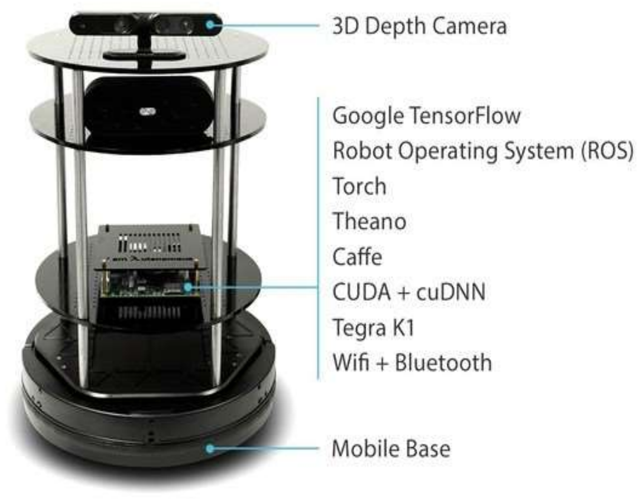


Figure (1): Generic TurtleBot

The TurtleBot hardware mainly consists of embedded modules that are based on ROS robotics software platform. It is equipped with highly computational Intel Joule 570x chip, Accelerometer/Gyro/Compass, Edge Detection & Bumper Sensors and RealSense 3D camera for navigation. These modules and others enable TurtleBot to perform many tasks, such as autonomous driving, obstacles detection and avoidance in real time.

by adding MK3 4DOF (PhantomX pincher) Robotics Arm that can be controlled by Moveit “open source software” , TurtleBot 2i acquires new capabilities like object sorting, and object manipulations which is big advancement compared to previous versions. Overall, TurtleBot 2i is the one of the most popular educational robotics platform.

TurtleBot 2i ROS Specifications

The TurtleBot 2i is powered by the Intel Joule 570X module embedded Linux computer offering an unparalleled amount of processing power, IO connectivity, and innovative features in a compact and power efficient package. The robot features dual 3D camera configurations, using a dedicated long range Intel RealSense camera ZR300-Series for Navigation & Mapping, and the short range Intel RealSense camera SR300-Series as a dedicated Manipulation workspace sensor. Intel RealSense technology was built to pair perfectly with the Intel Joule platform - with machine vision processing benefiting from hardware GPU acceleration from the on-board Iris Graphics solution.

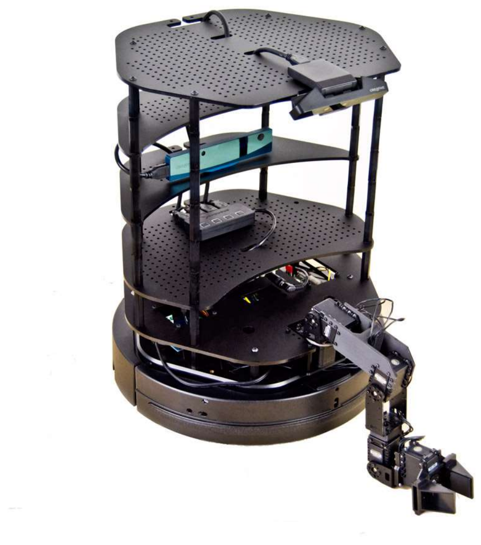


Figure (2): TurtleBot 2i with Pincher MK3 Robotic Arm

The TurtleBot 2i offers the Pincher MK3 Robotic Arm as a fully supported standard option, allowing the robot to interact with small objects, buttons, and tools in the real world. The Arbotix-M Robocontroller provides an interface for the Pincher Mk3 arm, which is implemented using MoveIt, an open source inverse kinematics solution, allowing users to control the arm using only high-level commands.

Hardware Specifications

CPU SOC

- Intel Joule 570X module
- Gumstix Nodana Carrier Board
- 4GB RAM
- 16GB eMMC Storage
- 802.11AC WiFi / Bluetooth 4.0
- Ubuntu 16.04 / ROS Kinetic

Sensors

- Intel RealSense 3D Camera SR300-Series
- Intel RealSense 3D Camera ZR300 Series
- Accelerometer/Gyro/Compass
- Edge Detection & Bumper Sensors

Mobile Robot

- Kobuki Mobile Base
- Modular & Interchangeable Decks
- Pincher MK3 Robo Arm
- Arbotix-M Robocontroller
- Secondary 3S 4500mAh LiPo Battery
- Maximum translational velocity: 70 cm/s 13
- Maximum rotational velocity: 180 deg/s (>110 deg/s gyro performance will degrade)
- Payload: 2kg (without arm), 1kg (with arm)
- Cliff: will not drive off a cliff with a depth greater than 5cm
- Threshold Climbing: climbs thresholds of 12 mm or lower
- Rug Climbing: climbs rugs of 12 mm or lower
- Expected Operating Time: 4-6 hours (operating time varies depending on loadout)
- Expected Charging Time: 2-3 hours (charge time varies depending on loadout)

- Docking/Charging: automatic within a 2mx5m area in front of the docking station

Modules and functionality

Teleoperations and Navigation

The TurtleBot is capable of remote operation on the keyboard through its *turtlebot_telop_keyboard/cmd_vel* (geometry_msgs/Twist) node, which is responsible for controlling commanded velocity in any controllable direction. Within Rviz after bringing up the TurtleBot with the roslaunch *turtlebot2i_bringup turtlebot2i_demo1.launch* command, the TurtleBot will respond to "2D Nav Goals", and autonomously navigate either an old map or a new map dependent on if you brought up the TurtleBot with the *new_rtabmap:=true* launch file argument. When given a 2D Nav goal via Rviz, the TurtleBot navigates as close as it can to the desired location, using either a pre made map or by creating a new map using Simultaneous Localization and Mapping, or SLAM.

The TurtleBot just uses the ROS premade SLAM package, which is a wrapper for OpenSLAM's package. OpenSLAM uses a particle filter to decrease the uncertainty the robot's pose in relation to the map. Continuous OpenSLAM also takes advantage of continuous resampling to help reduce uncertainty as well as reposition. Typically, a LIDAR system is needed to create this point map, however the TurtleBot obviously does not have one, opting instead for a 3D camera in its stead. The 3D camera is only capable of reading distance in a small cone in the front of the robot, so to build an entire map the TurtleBot must rotate a full 360 and at several points.

Figure (3): Example map made in OpenSLAM

SLAM

The SLAM algorithm (Simultaneous Localization and Mapping) is the procedure used on the TurtleBot for creating and updating a map in real time. Most SLAM algorithms, including the one used in OpenSLAM, filter data using particle filters and an extended Kalman filter to reduce uncertainty and make coherent maps. ROS only maps in two dimensions, however SLAM is capable of making topological maps. OpenSLAM relies on LIDAR for distance and angle sensing, however any other distance sensor can be used for SLAM to work properly as long as it has accurate filtered distance feedback. While LIDAR is fastest and will work well on the TurtleBot, it is not implemented, instead opting for the 3D camera.

Map Generation

Creating and saving a map in the TurtleBot is a fairly simple process. As stated in the previous section, the user will need to run the following commands:

```
goros
roslaunch turtlebot2i_bringup turtlebot_demo1.launch new_rtabmap:=true
```

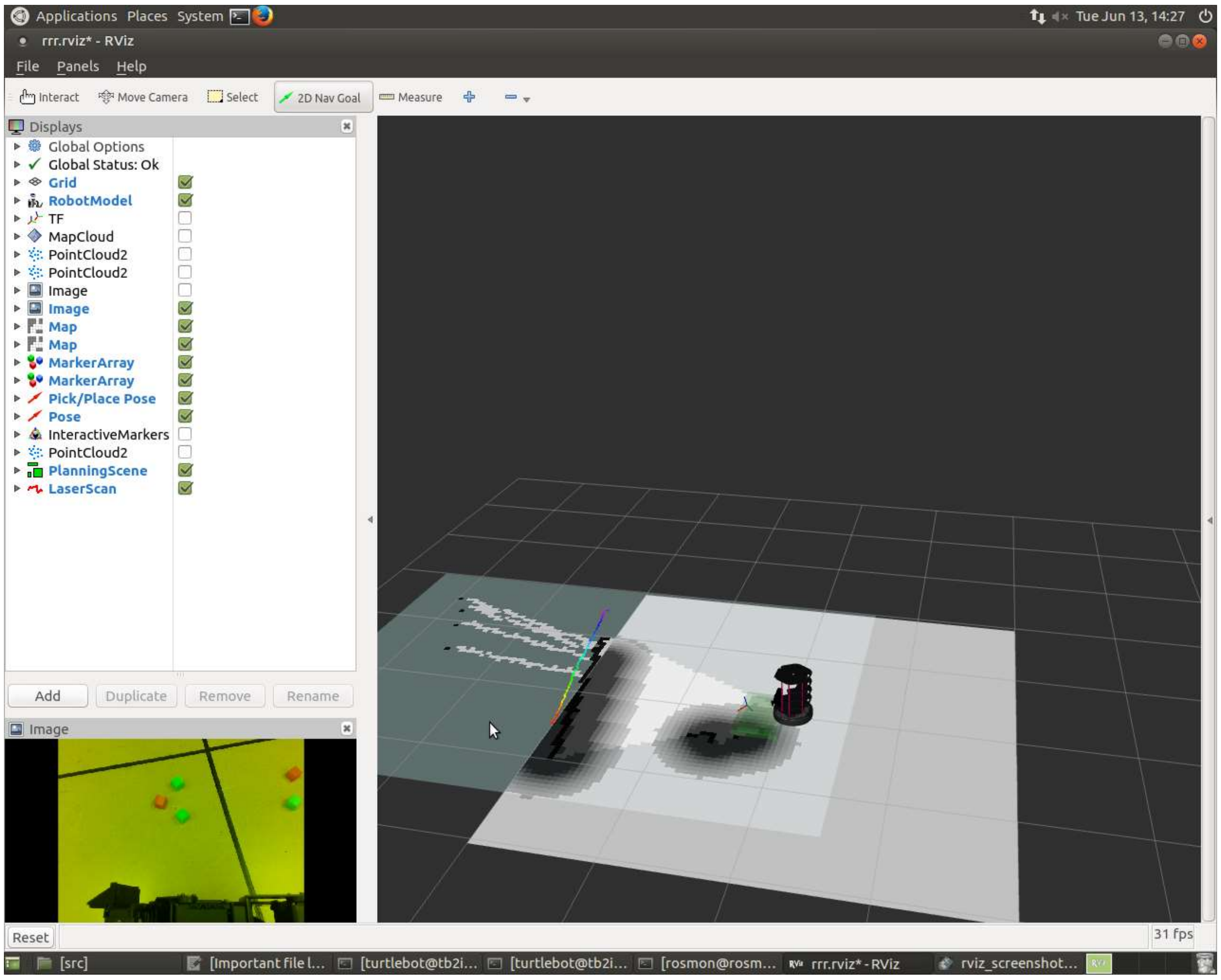
On the host computer run the following commands:

```
goros
rosrun rviz rviz
```

Once Rviz is open, the correct rviz file and you will be able to give 2D nav goals. The robot may also be controlled through a teleop program by running the following command, then following the instructions that appear onscreen.

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Once everything is running, a new map will be generated and you should see a new map form similar to the following:



figure(4): New map being generated in Rviz

Object Detection and Obstacle Avoidance

The pre-installed ROS packages do all of this and the ways in which they are done are hidden from the end users who work operate at a significantly higher level of abstraction. There are bumpers on the sides of the base which are used to detect collisions with obstacles.

Kobuki Base Auto Docking

Same as object detection and obstacle avoidance, the Turtlebots use some sort of proprietary radiofrequency system along with work zone detection to find charging stations and then dock at them. To make the turtlebot dock, enter

roslaunch kobuki_auto_docking activate.launch

After bringing up the robot. The robot base automatically charged when docked but does not charge the battery that powers the Joule. To run the joule the LiPo powering it needs to be charged separately.

Applications

Arbotix Object Manipulation

The TurtleBot arm is an off the shelf Trossen PhantomX pincher arm which has been affixed to the base of the robot.. The PhantomX is a 4 degree of freedom arm specifically designed for ROS applications. It uses 5 AX-12 Dynamixel servos for actuation as well as feedback. The gripper is actuated by the last servo, and is just two plastic plates with foam for additional friction. The TurtleBot is capable of independently actuating all 5 servos, and uses this capability to place the arm in a "pose". ROS will input a 3 dimensional xyz coordinate or a 'pose', and convert that pose using inverse kinematics into a series to angles for the robot arm. Because the distance between each arm segment is fixed, only the angles are needed. The angles for each move are calculated using an inverse kinematic function.

figure(5): PhantomX Robot Arm

Inverse Kinematics

In the figure 6, the concept of both forward and inverse kinematics are shown.

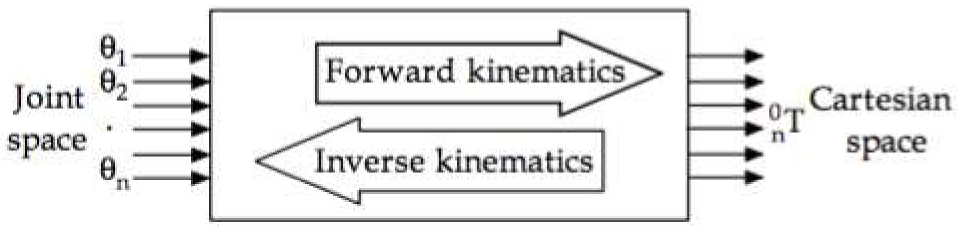


figure (6): forward & inverse kinematics

Inverse kinematics, used in the TurtleBot or otherwise, is a method of motion planning to equate the required angles of degrees of freedom given a specific length. It in essence uses compounded 2D polar coordinates to calculate angles to achieve the absolute distance and angle from the origin point.

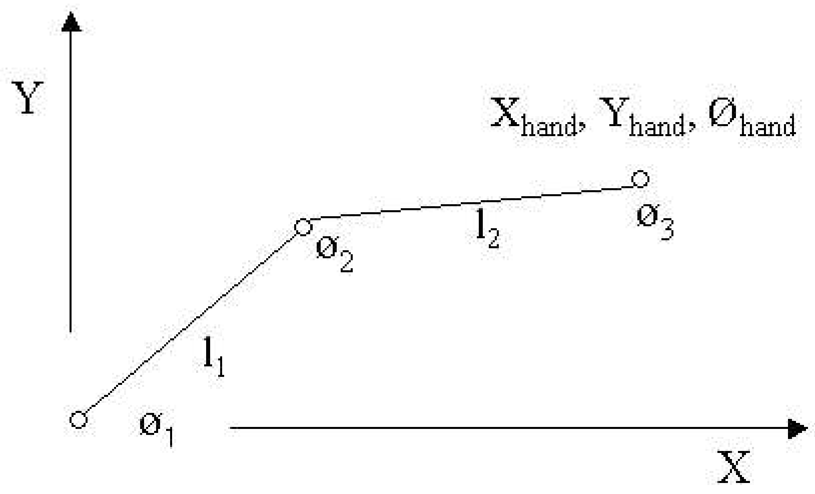
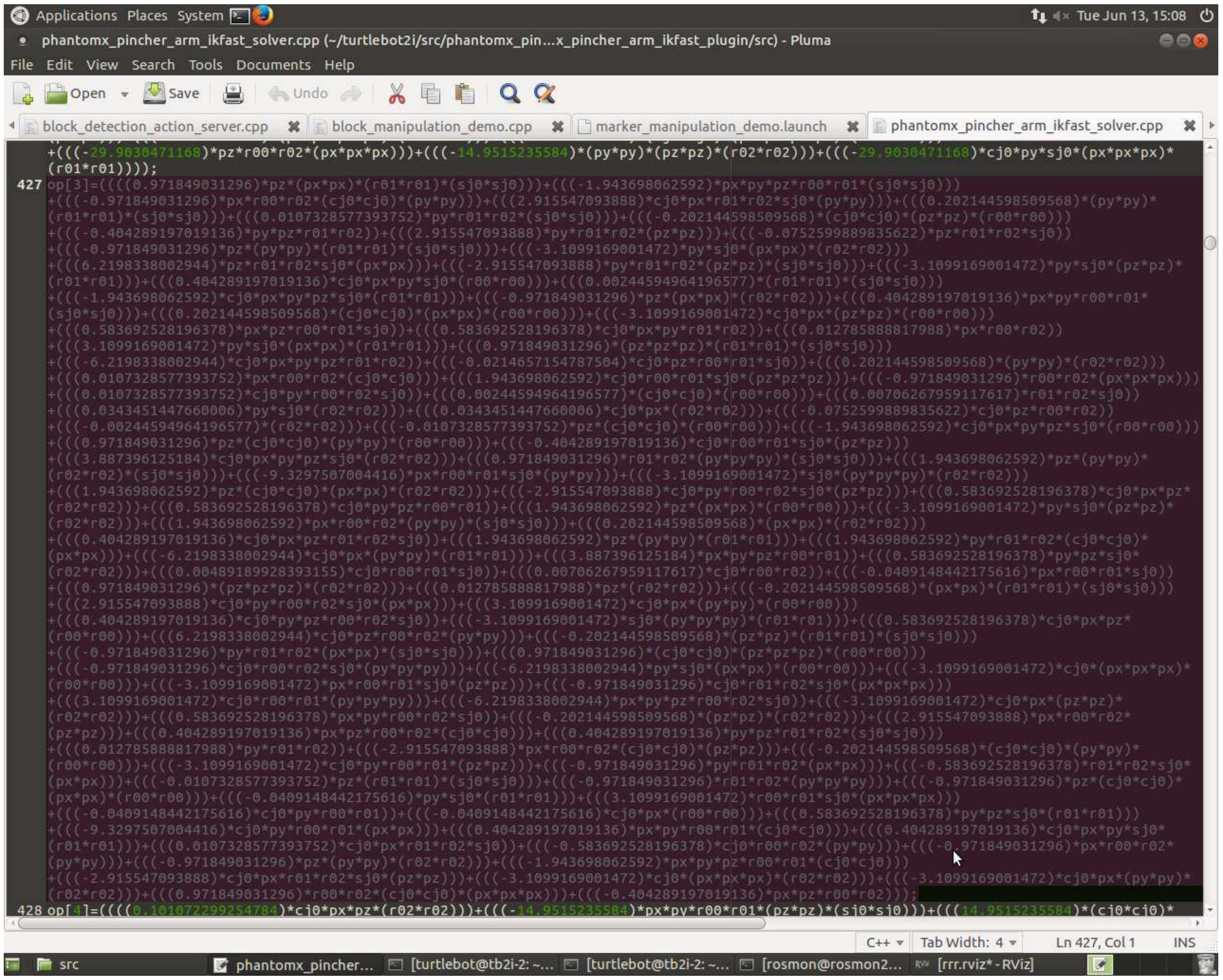


figure (7): XY angle planning example

Inverse kinematics in the provided code look like they were generated by another program and are undecipherable to a human, with thousands of characters per equation for each line of angle calculation. In the picture below, the entire highlighted block is one of 5 lines of code that appear to do all of the inverse kinematics for the arm. There are 5 servos on the arm so presumably each line is in charge of the positioning for one of the servos. It's hard to tell, however, since there are no comments anywhere and variable names are not very informative.



```
+(((-29.9030471168)*pz*r00*r02*(px*px*px)))+(((-14.9515235584)*(py*py)*(pz*pz)*(r02*r02)))+(((-29.9030471168)*cj0*py*sj0*(px*px*px)*
(r01*r01))));
op[3]=((((0.971849031296)*pz*(px*px)*(r01*r01)*(sj0*sj0)))+(((-1.943698062592)*px*py*pz*r00*r01*(sj0*sj0)))
+(((-0.971849031296)*px*r00*r02*(cj0*cj0)*(py*py)))+(((2.915547093888)*cj0*px*r01*r02*sj0*(py*py)))+(((0.202144598509568)*(py*py)*
(r01*r01)*(sj0*sj0)))+(((0.0107328577393752)*py*r01*r02*(sj0*sj0)))+(((-0.202144598509568)*(cj0*cj0)*(pz*pz)*(r00*r00)))
+(((-0.404289197019136)*py*pz*r01*r02))+(((2.915547093888)*py*r01*r02*(pz*pz)))+(((-0.0752599889835622)*pz*r01*r02*sj0))
+(((-0.971849031296)*pz*(py*py)*(r01*r01)*(sj0*sj0)))+(((-3.1099169001472)*py*sj0*(px*px)*(r02*r02)))
+(((6.2198338002944)*pz*r01*r02*sj0*(px*px)))+(((-2.915547093888)*py*r01*r02*(pz*pz)*(sj0*sj0)))+(((-3.1099169001472)*py*sj0*(pz*pz)*
(r01*r01)))+(((0.404289197019136)*cj0*px*py*sj0*(r00*r00)))+(((0.00244594964196577)*(r01*r01)*(sj0*sj0)))
+(((-1.943698062592)*cj0*px*py*pz*sj0*(r01*r01)))+(((-0.971849031296)*pz*(px*px)*(r02*r02)))+(((0.404289197019136)*px*py*r00*r01*
(sj0*sj0)))+(((0.202144598509568)*(cj0*cj0)*(px*px)*(r00*r00)))+(((-3.1099169001472)*cj0*px*(pz*pz)*(r00*r00)))
+(((0.583692528196378)*px*pz*r00*r01*sj0))+(((0.583692528196378)*cj0*px*py*r01*r02))+(((0.012785888817988)*px*r00*r02))
+(((3.1099169001472)*py*sj0*(px*px)*(r01*r01)))+(((0.971849031296)*(pz*pz*pz)*(r01*r01)*(sj0*sj0)))
+(((-6.2198338002944)*cj0*px*py*pz*r01*r02))+(((-0.0214657154787504)*cj0*pz*r00*r01*sj0))+(((0.202144598509568)*(py*py)*(r02*r02)))
+(((0.0107328577393752)*px*r00*r02*(cj0*cj0)))+(((1.943698062592)*cj0*r00*r01*sj0*(pz*pz*pz)))+(((-0.971849031296)*r00*r02*(px*px*px)))
+(((0.0107328577393752)*cj0*py*r00*r02*sj0))+(((0.00244594964196577)*(cj0*cj0)*(r00*r00)))+(((0.00706267959117617)*r01*r02*sj0))
+(((0.0343451447660006)*py*sj0*(r02*r02)))+(((0.0343451447660006)*cj0*px*(r02*r02)))+(((-0.0752599889835622)*cj0*pz*r00*r02))
+(((-0.00244594964196577)*(r02*r02)))+(((-0.0107328577393752)*pz*(cj0*cj0)*(r00*r00)))+(((-1.943698062592)*cj0*px*py*pz*sj0*(r00*r00)))
+(((0.971849031296)*pz*(cj0*cj0)*(py*py)*(r00*r00)))+(((-0.404289197019136)*cj0*r00*r01*sj0*(pz*pz)))
+(((3.887396125184)*cj0*px*py*pz*sj0*(r02*r02)))+(((0.971849031296)*r01*r02*(py*py*py)*(sj0*sj0)))+(((1.943698062592)*pz*(py*py)*
(r02*r02)*(sj0*sj0)))+(((-9.3297507004416)*px*r00*r01*sj0*(py*py)))+(((-3.1099169001472)*sj0*(py*py*py)*(r02*r02)))
+(((1.943698062592)*pz*(cj0*cj0)*(px*px)*(r02*r02)))+(((-2.915547093888)*cj0*py*r00*r02*sj0*(pz*pz)))+(((0.583692528196378)*cj0*px*pz*
(r02*r02)))+(((0.583692528196378)*cj0*py*pz*r00*r01))+(((1.943698062592)*pz*(px*px)*(r00*r00)))+(((-3.1099169001472)*py*sj0*(pz*pz)*
(r02*r02)))+(((1.943698062592)*px*r00*r02*(py*py)*(sj0*sj0)))+(((0.202144598509568)*(px*px)*(r02*r02)))
+(((0.404289197019136)*cj0*px*pz*r01*r02*sj0))+(((1.943698062592)*pz*(py*py)*(r01*r01)))+(((1.943698062592)*py*r01*r02*(cj0*cj0)*
(px*px)))+(((-6.2198338002944)*cj0*px*(py*py)*(r01*r01)))+(((3.887396125184)*px*py*pz*r00*r01))+(((0.583692528196378)*py*pz*sj0*
(r02*r02)))+(((0.00489189928393155)*cj0*r00*r01*sj0))+(((0.00706267959117617)*cj0*r00*r02))+(((-0.0409148442175616)*px*r00*r01*sj0))
+(((0.971849031296)*(pz*pz*pz)*(r02*r02)))+(((0.012785888817988)*pz*(r02*r02)))+(((-0.202144598509568)*(px*px)*(r01*r01)*(sj0*sj0)))
+(((2.915547093888)*cj0*py*r00*r02*sj0*(px*px)))+(((3.1099169001472)*cj0*px*(py*py)*(r00*r00)))
+(((0.404289197019136)*cj0*py*pz*r00*r02*sj0))+(((-3.1099169001472)*sj0*(py*py*py)*(r01*r01)))+(((0.583692528196378)*cj0*px*pz*
(r00*r00)))+(((6.2198338002944)*cj0*pz*r00*r02*(py*py)))+(((-0.202144598509568)*(pz*pz)*(r01*r01)*(sj0*sj0)))
+(((-0.971849031296)*py*r01*r02*(px*px)*(sj0*sj0)))+(((0.971849031296)*(cj0*cj0)*(pz*pz*pz)*(r00*r00)))
+(((-0.971849031296)*cj0*r00*r02*sj0*(py*py*py)))+(((-6.2198338002944)*py*sj0*(px*px)*(r00*r00)))+(((-3.1099169001472)*cj0*(px*px*px)*
(r00*r00)))+(((-3.1099169001472)*px*r00*r01*sj0*(pz*pz)))+(((-0.971849031296)*cj0*r01*r02*sj0*(px*px*px)))
+(((3.1099169001472)*cj0*r00*r01*(py*py*py)))+(((-6.2198338002944)*px*py*pz*r00*r02*sj0))+(((-3.1099169001472)*cj0*px*(pz*pz)*
(r02*r02)))+(((0.583692528196378)*px*py*r00*r02*sj0))+(((-0.202144598509568)*(pz*pz)*(r02*r02)))+(((2.915547093888)*px*r00*r02*
(pz*pz)))+(((0.404289197019136)*px*pz*r00*r02*(cj0*cj0)))+(((0.404289197019136)*py*pz*r01*r02*(sj0*sj0)))
+(((0.012785888817988)*py*r01*r02))+(((-2.915547093888)*px*r00*r02*(cj0*cj0)*(pz*pz)))+(((-0.202144598509568)*(cj0*cj0)*(py*py)*
(r00*r00)))+(((-3.1099169001472)*cj0*py*r00*r01*(pz*pz)))+(((-0.971849031296)*py*r01*r02*(px*px)))+(((-0.583692528196378)*r01*r02*sj0*
(px*px)))+(((-0.0107328577393752)*pz*(r01*r01)*(sj0*sj0)))+(((-0.971849031296)*r01*r02*(py*py*py)))+(((-0.971849031296)*pz*(cj0*cj0)*
(px*px)*(r00*r00)))+(((-0.0409148442175616)*py*sj0*(r01*r01)))+(((3.1099169001472)*r00*r01*sj0*(px*px*px)))
+(((-0.0409148442175616)*cj0*py*r00*r01))+(((-0.0409148442175616)*cj0*px*(r00*r00)))+(((0.583692528196378)*py*pz*sj0*(r01*r01)))
+(((-9.3297507004416)*cj0*py*r00*r01*(px*px)))+(((0.404289197019136)*px*py*r00*r01*(cj0*cj0)))+(((0.404289197019136)*cj0*px*py*sj0*
(r01*r01)))+(((0.0107328577393752)*cj0*px*r01*r02*sj0))+(((-0.583692528196378)*cj0*r00*r02*(py*py)))+(((-0.971849031296)*px*r00*r02*
(py*py)))+(((-0.971849031296)*pz*(py*py)*(r02*r02)))+(((-1.943698062592)*px*py*pz*r00*r01*(cj0*cj0)))
+(((-2.915547093888)*cj0*px*r01*r02*sj0*(pz*pz)))+(((-3.1099169001472)*cj0*(px*px*px)*(r02*r02)))+(((-3.1099169001472)*cj0*px*(py*py)*
(r02*r02)))+(((0.971849031296)*r00*r02*(cj0*cj0)*(px*px*px)))+(((-0.404289197019136)*px*pz*r00*r02)));
op[4]=((((0.101072299254784)*cj0*px*pz*(r02*r02)))+(((-14.9515235584)*px*py*r00*r01*(pz*pz)*(sj0*sj0)))+(((14.9515235584)*(cj0*cj0)*
```

figure (8) : The inverse kinematics file auto generated code

Block Sorting

The Turtlebot Comes with a block sorting demo ready to go. To run the block sorting demo change the active directory to the turtlebot directory after sshing into the machine and then following commands must be entered:

goros
roslaunch turtlebot2i_bringup turtlebot_demo1.launch
roslaunch turtlebot_arm_block_manipulation block_sorting_demo.launch
roslaunch turtlbot_arm_pose_director_demo.launch

The console will then give the option to input *d* or *m* to detect or move blocks respectively.

Autonomous Mapping

Mapping is handled by the provided software. The user can choose to make a new one by entering

new_rtabmap:=true

following the bringup command the or use a pre-existing one by not entering anything. To make use of autonomous mapping capabilities one turtlebot can drive around and map while the others use the data it is streaming out in order to focus on other tasks.

Interfacing with TurtleBot Code

Most of the TurtleBot code is written in ROS C++. To write custom ROS code for the TurtleBot, we've provided some examples below on how to publish information to useful nodes:

Arm Control

The most useful code is found in the pick_and_place_action_server.cpp file. To execute an action on the arm, you have to publish a ROS pose to the target_pose_pub. The code in the file publishes information in the following way:

```
Modiff_target.pose.position.z += std::abs(std::cos(rp))/50.0;
target_pose_pub_.publish(modiff_target);
```

Where the z is the z element of the target position, shifted for absolute value and divided by 50. To access the other elements of the pose, you address them in a similar manner.

Navigation Control

The best example of code controlling the base is in the navigation_utils.cpp file. The robot's base is controlled via a ROS "move_base" node, and filling out all parts and publishing it will move the robot, the following is an example on how to fill out a move base goal:

```
goal.target_pose.pose.position.x = go_posi.x();
goal.target_pose.pose.position.y = go_posi.y();
goal.target_pose.pose.position.z = 0.0;
```

Camera Interface

The Camera nodes can be subscribed to just like any other nodes to view data. For example, the method of importing the point cloud from the forward camera is by calling a ros subscriber:

```
ros::Subscriber sub_;
```

And then linking it to the correct node:

```
sub_ = nh_.subscribe("/camera_sr300/depth_registered/points", 1,
&BlockDetectionServer::cloudCb, this);
```

Future Project Ideas

After having worked with the TurtleBot, there are a few issues that could be worked out in other projects for other students;

1. Arm gripper feedback:

The TurtleBot's arm has no feedback for if it is gripping something or not. A simple addition such as a pressure sensor would be sufficient for gripping an object, but to help direct the arm (as it tends to miss) the addition of a camera with computer vision on the end effector would be better.

2. Application to have one robot hand something to another robot:

Since there are multiple robots with arms, an interesting project could be developing the kinematics to have one turtlebot hand an object to another one. This would demonstrate to students how to communicate between robots in ROS, as well as further develop the robot's arms.

3. Chess:

Since the robots have arms and cameras it might be interesting to have them use computer vision to detect and recognize different chess pieces and then use the arms to actuate them and play a game. Having the game be predetermined would be a good starting point before adding any game-decision making.

4. Controlling Your Robots over the Web:

By using RosBridge you can communicate with the TurtleBots over the internet. Jeeves currently has similar functionality on its navigation webpage. Something similar could be implemented to have it navigate to waypoints autonomously and perform a task.

5. Usage of Teleoperate for Robot Using Hand Gestures:

This will teach you how to build a gesture control device using Arduino and IMU. The gestures are translated into motion commands by ROS nodes. By converting the Teleoperation inputs into hand gestures the TurtleBot will be able to accept commands without the use of any transmitting electronics.

6. Creating a Self-driving Car or even soccer players Using ROS:

One of the more interesting projects in next terms could be autonomous navigation. Given the TurtleBot's map, it's start point and it's end point allowing it to navigate to another place on the same floor while avoiding obstacles could present an interesting challenge. We can build a simulation of the self-driving car using ROS and Gazebo package, perhaps using the 3D camera for obstacle avoidance as well as lane navigation.

Troubleshooting and Errors

Some part of the turtlebot was written in LISP, which is clearly an error

Problems

Many of the issues encountered by this project group originate from either poor documentation by Interbotix or a lack of prowess using the ROS platform. A deep understanding of ROS makes working with the turtlebots and tracking down sources of error significantly easier. There are two batteries that allow the robot to do its thing, one in the base and another one. These batteries are not connected and the battery which powers the joule does not charge when the base auto-docks. Furthermore, the batteries frequently run out of power when being operated.

Appendix

Resources

https://github.com/Interbotix/turtlebot2i/wiki

Caution should be exercised when following instruction directly from this website. Package names used may not be accurate.

https://github.com/Interbotix/turtlebot2i/blob/master/turtlebot2i_misc/ROS%20COMMANDS

Caution should be employe here as well for the same reasons as above.

Videos

https://www.youtube.com/watch?v=2Crktn0_RwE

https://www.youtube.com/watch?v=UGXOFWsyYUQ