My Independent Study Report

Tortugabot, a basic guide

Author: Ira Lauer

Supervisor: Prof. Michael Beetz

Advisor: Gayane Kazhoyan

February 8, 2016

Contents

Introduction

The Institute for Artificial Intelligence (IAI) at the University of Bremen have created the Tortugabot as a teaching and introductory tool for robotic programming. Through the work of many people, Tortugabot has created opportunities for undergraduates to have access and become familiar with the programming and software tools that the IAI uses on a day to day basis.

Along with producing the document about Tortugabot, I was also tasked with creating a map of the 2nd floor IAI office with Tortugabot. By experimenting with Tortugabot and the algorithms that it runs, I attempted to create the most accurate map possible and the results of this project can be seen at the end of the document.

What is Tortugabot?

The Tortugabot is the IAI's own take on a Turtlebot[1] which is an open source robotics platform from Open Source Robotics Foundation, Inc (as a note, "Tortuga" means "tortoise" in Spanish).

The Tortugabot is a relatively simple robot with several components. It has a 39cm by 29 base and stands 44 cm tall with 3 wheels, two of which are driven by servos. The Tortugabot has 2 sensors, one "Kinect" style sensor and a laser scanner. All of these parts are controlled by an attached netbook running Ubuntu and ROS, Robot Operating System.

ROS, or the Robot Operating System, is a framework for producing and running software on robots. ROS provides libraries, tools and methods to allow programs to work across codebases and programming languages. On Tortugabot, ROS is used as the main driver of the robot and the platform on which programs are written to control the robot. ROS allows all of Tortugabot's parts to communicate with one another and with the user.

What makes up Tortugabot?

The Tortugabot is made out of many materials and components, they are as follows:

- Frame: The frame is a custom aluminum and acrylic construction with 5 levels. The bottom level contains Velcro mounting points for the computer and batteries and has motor driver chips, servos and wheels mounted to the bottom. The second level holds the USB hub. The third and fourth levels hold the Kinect sensor and laser scanner

[1] For more information about turtle bot see http://www.turtlebot.com/

respectively. The fifth and top level is empty with the exception of an emergency brake button.

- Computer: The computer, which functions as the main controller of Tortugabot is Ideapad flex 10 netbook running 32 bit Ubuntu 14.04. Important software on this netbook includes: ROS Indigo, the code that launches the Tortugabot and the code for mapping with the laser scanner
- Servos: The two servos are mounted to the bottom of Tortugabot's frame and are rated at 60 amps and are each mounted to a 13 cm diameter rubber and plastic wheel.
- Motor Driver Chips: This is a board on the bottom of Tortugabot which allows the servos to be controlled by USB
- Laser Scanner: The Tortugabot features at Hokuyo Laser model number ust-10lx[2]. It is a uses a semiconductor laser diode to have a scanning range for 0.02 to 10 m in a 270 degree range. The laser also has a scanning frequency of 40Hz and a measuring accuracy of +/- 40mm and an angular resolution of .25 degrees. It connects to computers through an Ethernet port but on Tortugabot it is connected to the USB hub through an Ethernet to USB converter.
- Wheels: The Tortugabot has a total of 3 wheels. It has two 13 cm rubber and plastic wheels attached to the two servos and a third wheel that is on a free spinning axles behind the servos.
- "Kinect Sensor": The Tortugabot has an ASUS Xtion Pro[3], Kinect style sensor, which is used for RGB image sensing.
- USB hub: To bring everything together for communication purposes, the USB hub connects the computer with the laser Scanner, Kinect sensor, and servos.
- Remote Control: Tortugabot uses a Sony Dual Shock 3 controller.
- Batteries: Tortugabot uses two 2500 mAh LiPoly batteries running at a total of 22.2V to power itself.
- Red toggle: On top of the Tortugabot is a red toggle, which functions a physical kill switch. It is currently non-functional as the robot is small enough to not require it.

[2] https://acroname.com/products/HOKUYO-UST-10LX-LASER?sku=R359-UST-10LX
[3] https://www.asus.com/3D-Sensor/Xtion_PRO/

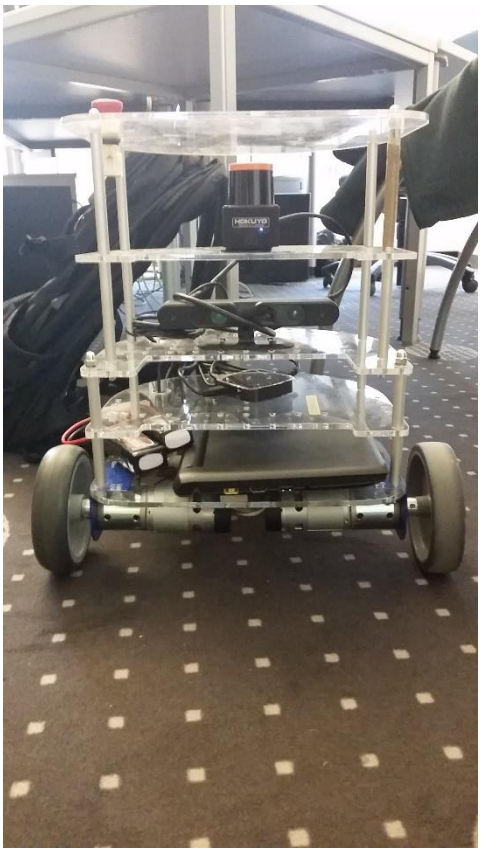

Figure 1 Front of Tortugabot

Figure 2 Back of Tortugabot

Figure 3 Top of Tortugabot

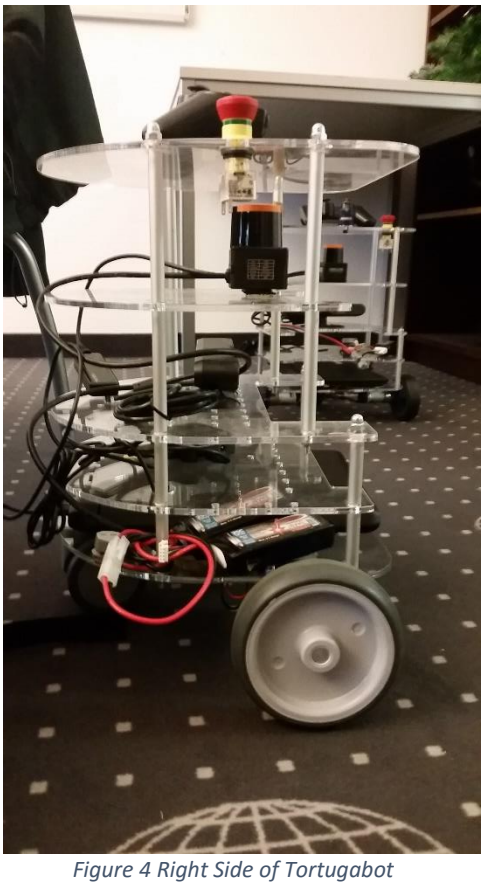

Figure 4 Right Side of Tortugabot

How does software interact with the Tortugabot?/ What exactly is ROS?[4]

ROS is a framework that provides tools and libraries for writing and running code across many computers and several different programming languages. ROS has three different levels of concepts which are all used on the Tortugabot. They are the filesystem level, the computational graph level, and the community level and each works in close conjunction with each other. The Filesystem level contains the physical code which is used to run a robot and includes the packages, package manifests, and definitions of service and message types. Packages are the atomic unit of a ROS project and are used to organize software for any ROS project. Packages may contain the runtime processes, nodes, libraries, configuration files or anything that is required to successfully run the included software. The Package manifest is a .xml file that describes the package itself, including data such as name, version, a description of the package, dependencies and other metadata about the package.

The Computation Graph level of ROS is how a ROS system is created at runtime. The Computational Graph is a peer to peer network of individual ROS processes that communicate with each other through several means. The fundamental unit of the Computational Graph is the node, which is an individual process running at run time. A node can control an individual piece of a robot or run a pathfinding algorithm or whatever computation that is needed. Each node talks to each other using messages, which is a user-defined data structure, by publishing and subscribing to different topics. A topic is a name given to a stream of messages from a node. There may be several publishers and subscribers to one topic and there may be a node that publishes and subscribes to many topics. Along with topics there is also services which are used in request and reply interactions and are defined by a request data structure and a reply data structure. While nodes talk directly to each other, the ROS Master acts as a directory for the names of topics and services and facilitates the initial connection between nodes. The ROS Master also includes a parameter server, which serves as a central hub for data in the Computational Graph. Finally there are bags which are formats for recording for later playback ROS message data.

The third level of ROS concepts in the ROS Community which provides many resources and ways for different groups and communities to exchange resources. The Community level includes that distributions of ROS itself, Repositories from different groups and the ROS Wiki.

---

[4] From http://wiki.ros.org/ROS/Concepts

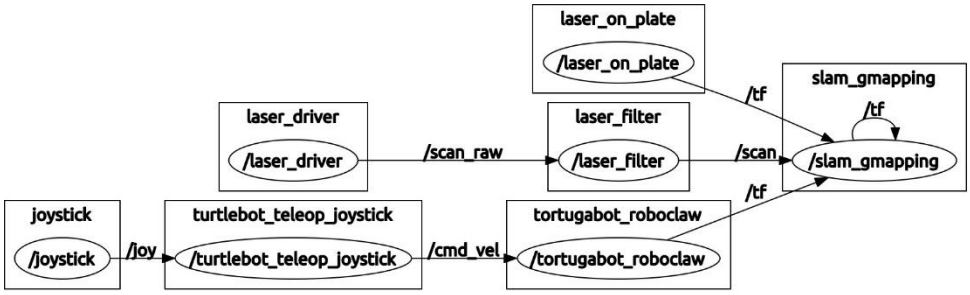

Figure 5 The computational graph of Tortugabot from the minimal_laser.launch and gmapping.launch files. Each box in the graph is an individual node. The strings on the arrows are topics being broadcast and listened to by each node.

What is the code for Tortugabot[5]?

The Tortugabot project is made up of several packages. Of most interest are the packages tortugabot_bringup and navigate_map, as these will likely be the most used packages of a user of Tortugabot. tortugabot_bringup contains that launch files of Tortugabot's movement and control and navigate_map contains the mapping and navigation programs for Tortugabot.

What are transforms/TF/coordinate system of ROS?

Robotic systems have many different links. Each link has a coordinate frame. A coordinate frame is a Cartesian coordinate system based upon a location in space. It is a point in space from which X, Y, and Z axes are based from. For example both the base plate and the laser scanner have coordinate frames.

The relationship between two frames is represented by relative pose which is a translation followed by a rotation.[6] If A and B are two frames, B's pose in relation to A is given by a translation from A's origin to B and then a rotation of B's axes in A's coordinate frame. The translation is given in by a 3d vector which just contains the values X, Y and Z and the rotation is given by a quaternion. A quaternion is basically a 3D vector and a rotation around this vector. This view of one coordinate frame inside of another one is called a transform.

[5] https://github.com/code-iai/tortugabot
[6] http://wiki.ros.org/tf/Overview/Transformations

Transforms are published and used by many nodes so ROS has TF,[7] which is a distributed coordinate frame tracking system. TF listens to and publishes the poses and transforms of different nodes.

On Tortugabot, with Gmapping running, there are 4 coordinate frames. Firstly there is the coordinate frame of the generated map. On Tortugabot there are 2 frames, one for the base plate and one for the laser scanner and the transform of the two is published. There is also the odometry coordinate frame which is the location of the Tortugabot when the odometry frame is created (usually from the minimal_laser.launch.)

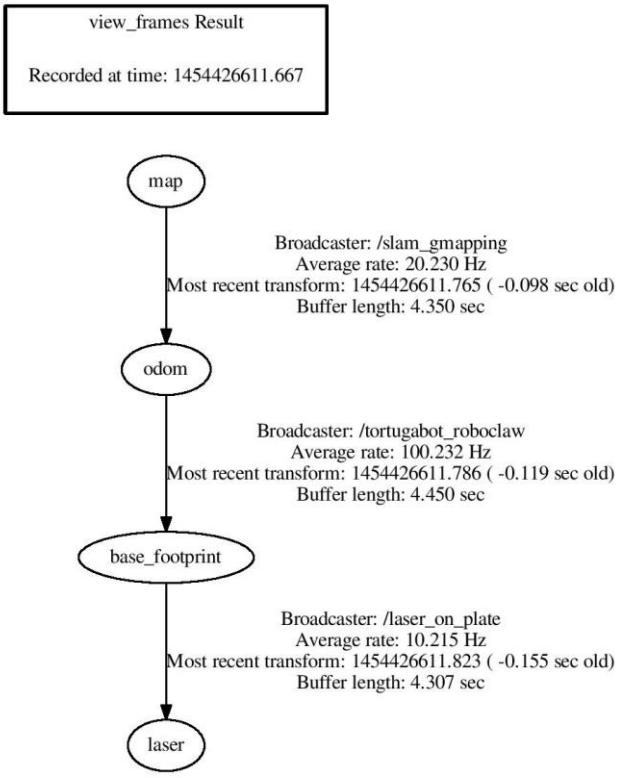

Figure 6 Transform Tree of Tortugabot

What are the different components of a Tortugabot launch file?

In order to start any robot running ROS one need to run a launch file. Located within a package, these files will be used by ROS's roslaunch to create nodes and initialize the robot.

[7] http://wiki.ros.org/tf

The .launch file requires a specific XML format. The following is the minimal_laser.launch[8], which creates nodes for the laser scanner along with Nodes required for Tortugabot to move.

```
<?xml version="1.0" encoding="utf-8"?>
<launch>
  <include file="$(find tortugabot_bringup)/launch/base_and_joy.launch"/>
  <node name="laser_driver" pkg="urg_node" type="urg_node"
args="_ip_address:=192.168.200.11">
    <remap from="scan" to="scan_raw" />
    <remap from="camera_depth_frame" to="laser" />
  </node>

  <node name="laser_filter" pkg="laser_range_filter" type="filter.py">
    <!-- <remap from="scan_filtered" to="scan" /> -->
    <!-- <remap from="scan" to="scan_raw" /> -->
  </node>

  <node name="laser_on_plate" pkg="tf" type="static_transform_publisher"
        args="-0.05 0 0.36 0 0 0 base_footprint laser 100"/>
</launch>
```

We will take a look at this launch file one piece at a time.

- `<?xml version="1.0" encoding="utf-8"?>`
  `<launch>`
  This piece of code defines what kind of file it is. It is a launch file encoded in utf-8 in the XML format.
- `<includefile=`
  `"$(findTortugabot_bringup)/launch/base_and_joy.launch"/>`
  This line brings in the base_and_joy.launch file which initializes the PlayStation controller along with the servos.
- `<node name="laser_driver" pkg="urg_node" type="urg_node"`
  `args="_ip_address:=192.168.200.11">`
  This line of code creates a node named “laser_driver” which is of type urg_node. urg_node is the driver for the Hukuyo laser scanner which is called with the parameter/argument of the its ip address.
- ```
    <remap from="scan" to="scan_raw" />
    <remap from="camera_depth_frame" to="laser" />
  </node>
  ```
  This bit of code renames the topics from the laser driver node and closes the node block.

- ```
  <node name="laser_filter" pkg="laser_range_filter" type="filter.py">
    <!-- <remap from="scan_filtered" to="scan" /> -->
    <!-- <remap from="scan" to="scan_raw" /> -->
  </node>
  ```
  This node block creates a node with the name Laser_filter from the laser_range_filter package from python code named filter.py. It filters out obstacles that are close to the Tortugabot such as the front poles of Tortugabot’s structure. This node also renames two topics.

[8] From https://github.com/code-iai/Tortugabot

- ```
    <node name="laser_on_plate" pkg="tf"
  type="static_transform_publisher"
          args="-0.05 0 0.36 0 0 0 base_footprint laser 100"/>
  </launch>
  ```
  This node block creates a node laser_on_plate publishes a constant transformation from base_footprint frame and the laser. It also closes the file.

What happens when a launch file is run and how do I run one?

As the launch file is run, it calls code to be execute by ROS.

In order to run a launch file, all one has to do is run the following command.

```
roslaunch [package-name] [launch-file]
```

If one is in the directory of the needed [launch-file] all one has to do is the following:

```
roslaunch [launch-file]
```

What is rviz?

rviz[9] is the 3D visualization environment for ROS. rviz allows a user to see a graphical representation of a robot's current state and allows the users to give commands to their robot through a computer . With Tortugabot a user can use rviz to see the laser scan, position of Tortugabot in relation to the world frame, and with Gmapping, see the created map.

rviz works by taking data from topics and so a user must input which topics rviz will listen to. To do this a user must click on the add button on the bottom left of the rviz window. This will bring up a list of new displays and a user can choose one of these displays or click on the tab "Add Displays by Topic." If a display is chosen through the first tab, the user must specify a topic by clicking on the added display in the display list, which is on the left of the window, and choose a topic from the drop down menu that is next to the word "topic." If a display was added by topic then the display is ready to go.

[9] http://wiki.ros.org/rviz
```

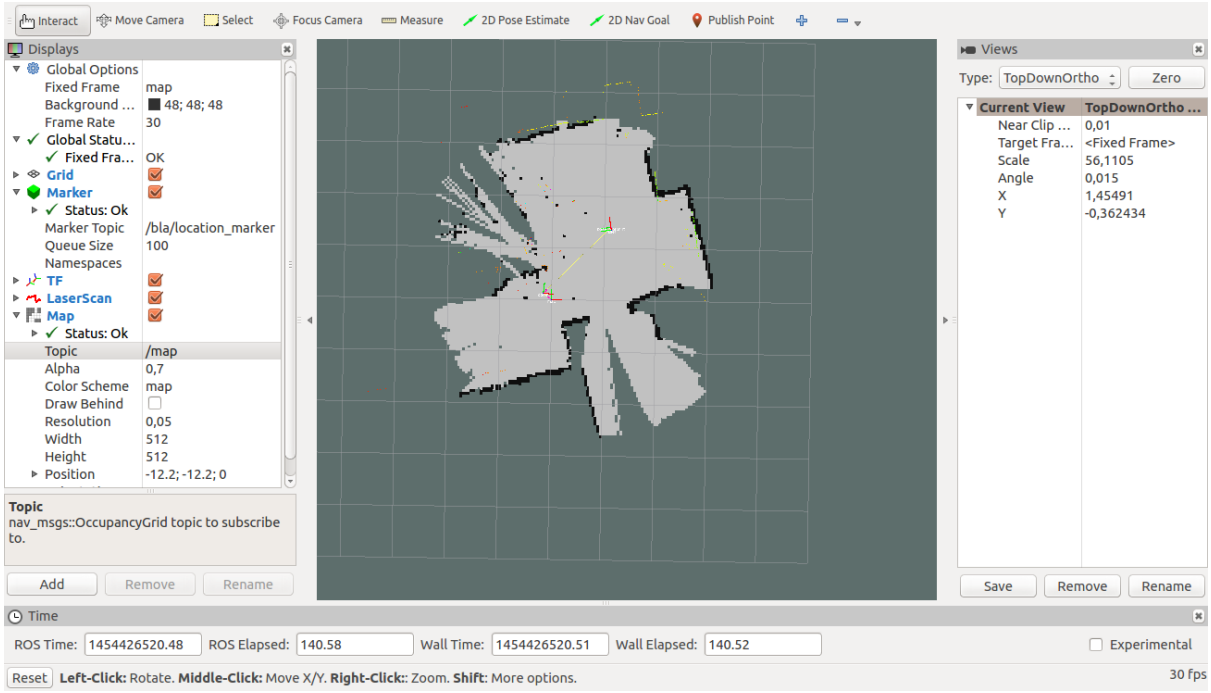

Figure 7 RVIZ with Tortugabot

What is Gmapping?

Gmapping[10] is a program that provides laser-based SLAM (Simultaneous Localization and Mapping). It creates a node that uses the laser scanner to not only build maps but also position Tortugabot on the created map. The Gmapping node uses OpenSlam's Gmapping which uses Rao-Blackwellized particle filters from the laser scanner along with the odometry of Tortugabot to map and localize a robot.

The Gmapping node on Tortugabot subscribes to the the tf (transforms) topic, which produces the transformations of Tortugabots frames (laser, base, and odometry) and the scan topic, which is the data from the laser scanner. Gmapping publishes on the map topic, which is the created map from the subscribed topics.

[10] http://wiki.ros.org/gmapping

What does the Gmapping launch file look like?

We are going to break down the gmapping.launch file and explain what each piece means

- ```
  <launch>

        <arg name="scan_topic"  default="scan" />

        <arg name="base_frame"  default="base_footprint"/>

        <arg name="odom_frame"  default="odom"/>
  ```

  This block describes the file as a launch file and specifies the input arguments for the gmapping node. The arguments are the data from the laser scanner, the base frame transformation and the odom_frame transformation.

- `<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">`
  This line creates the gmapping node. It is of type slam_gmapping and is named slam_gmapping. The following block of code is the parameters for the gmapping node and will be handled one at a time.
- `<param name="base_frame" value="$(arg base_frame)"/>`

  `<param name="odom_frame" value="$(arg odom_frame)"/>`

  The Base_frame is the frame of the base of Tortugabot. odom_frame is the frame attached to the Odometry system.

- `<param name="map_update_interval" value="5.0"/>`
  This is how often the map is updated. It is in seconds. Lower values lead to more rapid updates but can only work if the compute processing power is available. Overall lower the interval, more accurate the map.
- `<param name="maxUrange" value="3.0"/>`
  Maximum usable range of the laser in meters.
- `<param name="maxRange" value="8.0"/>`
  Maximum range of the laser in meters.
- `<param name="sigma" value="0.05"/>`
  The sigma used by the greedy endpoint matching.
- `<param name="kernelSize" value="1"/>`
  The kernel in which to look for a correspondence.
- `<param name="lstep" value="0.05"/>`
  `<param name="astep" value="0.05"/>`
  Lstep is the optimization step in translation. A step is the optimization in rotation.
- `<param name="iterations" value="5"/>`
  The number of iterations of the scanmatcher.
- `<param name="lsigma" value="0.1"/>`
  The sigma of a beam used for likelihood computation.
- `<param name="ogain" value="3.0"/>`
  Gain to be used while evaluating the likelihood, for smoothing resampling effects.
- `<param name="lskip" value="0"/>`
  Number of beams to skip each scan.
- `<param name="minimumScore" value="200"/>`
  Minimum score for considering the outcome of the scan matching good. Can avoid jumping pose estimates in large open spaces when using laser scanners with limited range.
- `<param name="srr" value="0.01"/>`
  ```

```
<param name="srt" value="0.02"/>
<param name="str" value="0.01"/>
<param name="stt" value="0.1"/>
```
All of these are the odometry error based upon rotation or translation.

- `<param name="linearUpdate" value="0.5"/`
  `<param name="angularUpdate" value="0.436"/>`
  These values are how often a scan is processed based upon a distance translated (linear) or how much rotated (angular).
- `<param name="temporalUpdate" value="-1.0"/>`
  Process a scan if the last scan processed is older than this value. -1 means time based updates are turned off.
- `<param name="resampleThreshold" value="0.5"/>`
  The Neff based resampling threshold
- `<param name="particles" value="80"/>`
  The number of particle in the filter.
- `<param name="xmin" value="-1.0"/>`
  `<param name="ymin" value="-1.0"/>`
  `<param name="xmax" value="1.0"/>`
  `<param name="ymax" value="1.0"/>`
  These prameters create the initial size of the map.
- `<param name="delta" value="0.05"/>`
  This value changes the resolution of the map. Lower values means higher resolution but larger files.
- `<param name="llsamplerange" value="0.01"/>`
  `<param name="llsamplestep" value="0.01"/>`
  Translational sampling range and step for the likelihood.
- `<param name="lasamplerange" value="0.005"/>`
- `<param name="lasamplestep" value="0.005"/>`
  Angular sampling range and step for the likelihood
- `<remap from="scan" to="$(arg scan_topic)"/>`

```
</node>

</launch>
```

This block of code renames the outputted data to scan. Also closes ends the node and the file.

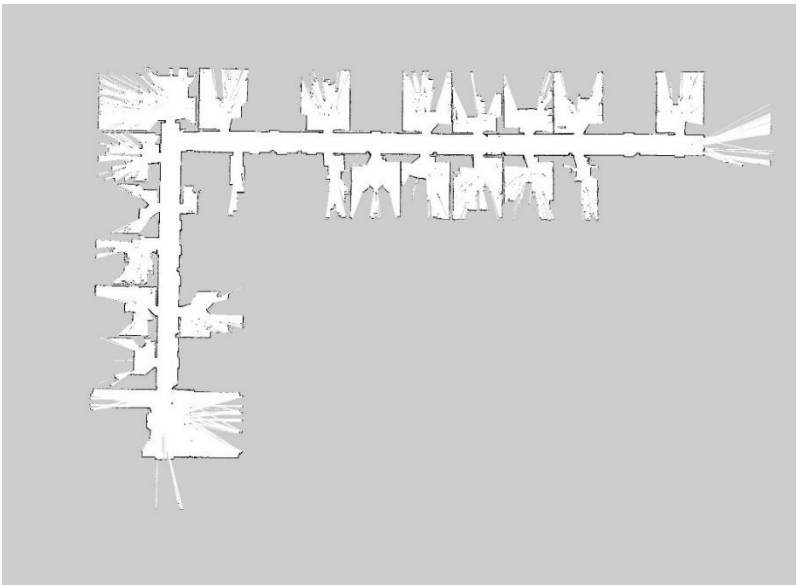

Figure 8 Map made by Gmapping and Tortugabot

What are the steps to run the Tortugabot (Movement and Gmapping)?

1. Plug in the USB cable from the USB hub into the netbook and plug in the battery to Tortugabot
2. Turn on the netbook and login to the desired account on it.
3. Turn on your own computer and ssh into the Tortugabot with the following command. Make sure you are on the same network as Tortugabot, should be Tortuganet.

   ```
   ssh –X [username]@Tortugabot[your Tortugabots number]
   ```

   The "-X" allows you to access gui elements from Tortugabot. It is not required. One will have to ssh into for each node/launch file that is run

4. Run the following command:

   ```
   roscore
   ```

   This starts the roscore on the Tortugabot.

5. Execute the following command:

   ```
   roslaunch tortugabot_bringup minimal_laser.launch
   ```

   This launch the nodes that allows the Tortugabot to be controlled with the game controller and starts the laser on Tortugabot.

6. Execute the following command:
   ```
   roslaunch navigate_map gmapping.launch
   ```
   This starts the gmapping nodes.
7. On the user's own computer, run the following commands
   ```
   export ROS_MASTER_URI=http://Tortugabot[your Tortugabots number]:11311
   ```
   This allows RVIZ to run successfully on the your computer
8. Run the following command on your computer
   ```
   rosrun rviz rviz
   ```
   This initiates the visualization of the robot. Add the displays need from topics

At this point Tortugabot bot has created the following nodes and is publishing on the following topics

Nodes

- “joystick”- the PS4 controller
- “turtlebot_teleop_joystick” – conversion from PS4 controller input to cmd_vel commands for the servos
- “Tortugabot_roboclaw” - Takes in data from turtlebot_teleop_joystick and uses on the servo control chip.
- “laser_driver” – controls the laser scanner and publishes its data.
- “laser_filter” – publishes a filtered verison of laser_driver data
- “laser_on_plate” – publishes the transformation of the laser scanner
- “slam_gmapping” – computes and publishes a map from scan data using Gmapping

Topics

- /cmd_vel – the published speed and rotation commands of Tortugabot
- /diagnostics- ros topic
- /joy – published button presses and joystick positions from PS4 controller
- /laser_driver/parameter_descriptions – where parameters of laserdriver are published
- /laser_driver/parameter_updates
- /map – where the created map from Gmapping is published
- /map_metadata
- /rosout
- /rosout_agg
- /scan – where the filtered laser scan data is published
- /scan_raw – unfiltered laser scan data
- /slam_gmapping/entropy
- /tf – transformations of Tortugabot and parts of Tortugabot
- /tf_static
- /Tortugabot_roboclaw/odom – location of Tortugabot at startup and calculations based upon movement.

What is rosbag?[11]

rosbag is a suite of tools that allows a user to record and replay ROS Topics. It records thise data onto a .bag file. With the command,

```
rosbag record –a
```

All the published topic data will be recorded into a bag file which will be named [the-time-the-bag-file-was-created].bag, eg. `2014-12-10-20-08-34.bag`. To examine the bag file use the following command,

```
rosbag info [your bagfile]
```

This will tell the user information about the bag file such as types of messages, list of topics and the duration of the bag file. To replay the bag file use the following command

```
rosbag play [your bag file]
```

This will start publishing messages to its recorded topics. These messages on the topics can be used by running nodes. This is highly useful as a user could create bag file of a run of Tortugabot and then replay these files, along with Gmapping, to create maps. Bag files can also be used for troubleshooting and as a way to manipulate data without having to constantly use Tortugabot. Also of interest is the following command,

```
rosbag play –r 2 [your bag file]
```

which changes the playback rate of a bag file, in this case by 2, so bag files can be replayed both slower and faster.

Tips and other advice

- The following ROS commands are important and useful
    - `rostopics` this command set of commands that are used to look at the topics in the command shell
        - `rostopics echo [topic-name]` : This command puts the what a topic is broadcasting on the command shell
        - rostopics list : this command lists all of the current topic running on Tortugabot
- Tips for while Running Tortugabot
    - There is a known bug where the Tortugabot will stop responding to commands from the gamepad when Tortugabot in a single direction for an extended period of time. The users will have to disconnect the battery from Tortugabot. It is then recommended to just restart all of the nodes on the Tortugabot
    - Try to keep cords away from the front of Tortugabot as they will be picked up by the laser scanner and cause bad mapping.
    - Moving in a straight line produces better maps than many turns

[11] http://wiki.ros.org/rosbag

Project with Tortugabot

As part of my independent study with Tortugabot, I was given the task to accurately map out the second floor of the IAI using Tortugabot. This map can be seen in the section "What does a Gmapping launch file look like?" I drove Tortugabot up and down the hallways of IAI and recorded the produced topics into a bag file. I then ran the bag file through the Gmapping node and tweaked both the Gmapping parameters and the play back speed of the bag file. The changes that produced a more accurate map were a higher delta, a large decrease in the map interval value, and a slower play back. The higher delta produced a more precise map with much more detail than the default delta but produced a much larger file (Default delta produced around a 1 MB file while my delta of .005 produced a 31 MB file). The decrease map interval value and slower play back time go hand in hand. A decrease interval (I used .005) means more scans of the map are processed per second which leads to a better map but at normal playback speeds the computer could not keep up with the workload. When the map was played back at a slower speed, half or quarter speed, the computer could keep up with the processing and the maps produced where much better than the default configuration.

In order to test the accuracy of Tortugabot, I got measurements of the IAI office and compared them against the map I made. The results are as follows

Measurement	Measured Distance (m)	Tortugabot measured distance	% difference
Door Frame of Room 58	.81	.8	1.2
East-West Hall	43.93	47.4	7.9
North-South Hall	21.27	23.9	12.4
Door of Room 57 to Room 58	2.4	2.5	4

The Tortugabot seems to have a small ~10% margin of error to big. This could stem from many factors, including a poor recording session to make the map, imperfect physical measured differences or problems with Gmapping or its parameters.