ME 597 Lab #1:

Simulation and Turtlebot Introduction

Introduction:

The main goal of this lab is to familiarize your team with the development environment, tools, and the robotic platform used in this course. As this is your first time through the labs with this hardware and software interface, there may be a lot of exploration required to get things to work. This lab manual provides the general steps needed to complete the lab; however, the first hour lab sessions will be dedicated to a tutorial session which will walk you through a detailed overview of working with these tools and will walk you through many of the steps of the lab with you.

Each group must bring at least one good laptop computer with Linux installed with them to the lab. It is highly recommended that you perform Steps 1-3 of the lab before coming to your lab session as downloading and installing ROS can take a fair amount of time.

All suggestions and comments for improving documentation or template code are welcome.

Background:

This lab is a step by step guide to the various tools and programs you will need in order to complete the rest of the labs in this course. For this and all subsequent labs, you will be doing all of your coding in ROS (C/C++) in the Linux OS. ROS is a package of software tools which help in the development of robotic algorithms. In this lab you will be introduced to several of the most common tools including, the *Gazebo* simulator, the *RVIZ* visualization environment, and two common packages, *gmapping*, and the *navigation stack*. You will also be introduced to the *Turtlebot* robotic platform which you will be using for this course and the indoor positioning system in the SDC used to track turtlebot motion. An example ROS package (C++ code and makefiles) is available on the course website to get you started for this lab, called turtlebot_example_lab1.zip.

For the most part a large section of this, and subsequent labs, should be completed before coming to the lab. For those who have a background using ROS and do not wish to participate in the tutorial session a TA will be available to check your work and assign you a Turtlebot throughout the lab time if you have all of the prerequisite work completed. For those that are new to ROS please come to the ROS tutorial session during the first hour of your lab time slot to learn the fundamentals you will need to complete this and future labs. If possible, please attempt Step 1 of the lab instructions so that you can follow along with the presentation.

Lab Instructions:

1. **Installing ROS and required packages**: To get started each group should have a computer with Linux (Ubuntu 14.04.3 LTS is recommended http://www.ubuntu.com/download/desktop)

installed. If a group cannot meet these requirements please talk to the TA. The installation of ROS is detailed on the ROS Indigo installation webpage and will not be repeated here. We will use Gazebo 2 in this course. For this and all subsequent labs it is assumed that you are running the same version of ROS, namely ROS Indigo. It is recommended that you install the desktop-full version to ensure all the necessary packages are included. Ensure that all the environment variables are set correctly in your *.bashrc* file as described here. Also, we will use catkin as our build system, so make sure to set up your catkin workspace as described here. Next, the Turtlebot simulation software must be installed. To do this, open a terminal and type:

```
$sudo apt-get install ros-indigo-turtlebot*
```

This will install all the required packages to run the Turtlebot and the simulator. Finally download the example package from the course website and place it in your catkin workspace source folder (usually ~/catkin_ws/src), and compile it.

```
$cd catkin_ws
$catkin_make
```

Tip: Many ROS packages can be downloaded and installed directly using apt-get. The syntax is the same as above but replaces "turtlebot" with the desired package name. Other package may require the use of git or svn to obtain the source and compile manually. Feel free to explore the expansive collection of packages the ROS community has to offer.

2. **Complete the ROS Tutorials:** It is HIGHLY recommended that you complete all of the beginner and intermediate tutorials located here: 'http://wiki.ros.org/ROS/Tutorials '. It should take about 2-4 hours once everything is installed.

```
SERIOUSLY… DO THIS STEP…PARTICULARLY BEGINNER TUTORIALS 1-20
```

At any points where you have the option to choose between ROS distributions (such as fuerte, groovy, hydro, indigo, jade) select indigo. At any points where you have the option to choose between build systems (either catkin or rosbuild), select catkin.

3. **Running the simulator:** The simulating software, *Gazebo*, comes with the standard ROS installation and is used to simulate robot input and output. This makes it possible to test your code without the need of the physical hardware and will allow you to code your algorithms from home. A custom launch file has been included in the example package on the course website. To run the simulator enter the following into the terminal:

```
$roslaunch turtlebot_example turtlebot_gazebo.launch
```

Recall from the ROS tutorials above that roslaunch looks in the package folder *turtlebot_example's* launch directory for a launch file called *turtlebot_gazebo.launch* to launch all

the nodes with the associated parameters listed therein. This should bring up the Gazebo window and you should see a rendering of the turtlebot in a sample world.

Explore the options in the Gazebo simulator, add some additional models (standard blocks and more interesting models like dumpsters). Save your world and open the turtlebot_gazebo.launch file, and modify it to load your new world model (currently sample_world.world).

4. **Controlling the Turtlebot:** The Turtlebot is controlled using a *ROS topic* called *cmd_vel*. This topic is a *Twist* message. Mathematically a twist represents a 6 degree of freedom velocity vector, 3 linear velocities and 3 angular velocities, [x,y,z,θ,ϕ,ψ]. Since the Turtlebot has only 2 degrees of freedom only one linear (x, forward velocity) and one angular (ψ, yaw rate) value are used, all in the body frame of the robot. Programmatically, the geometry_msgs/Twist definition can be found online in the geometry_msgs package.

   The cmd_vel topic can be set in many different ways, e.g. through teleoperation, or by a planner. As a result, the turtlebot has a cmd_vel_mux topic interface that receives commands from different sources simultaneously and selects which one to execute. Commands can be sent directly from the command line by defining the teleop inputs to execute:

```
$ rostopic pub -r 10 /cmd_vel_mux/input/teleop geometry_msgs/Twist
'{linear:  {x: 0.2, y:  0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: -0.2}}'
```

   This command publishes the Twist message topic cmd_vel_mux/input/teleop at a rate of 10 Hz as a constant value with forward velocity 0.2 m/s and yaw rate of -0.2 rad/s. The robot in the simulation window should move according to the commands you send. It should therefore drive in a circle every 31.4 seconds with a circumference of 6.28 m. Since the robot uses differential drive it is possible to rotate on the spot by setting the linear speed to zero and rotation to a non-zero value.

   The Tutlebot code also comes with a tele-operation mode that let you control the robot using the keyboard. To do so run the following command from a terminal:

```
$roslaunch turtlebot_teleop keyboard_teleop.launch
```

   The output in the terminal indicated the control scheme, but note, you can only drive when the terminal that you launched the teleop from is active (it must receive your keyboard inputs). Drive the robot around and crash into some blocks at speed. What happens when you run both control methods at the same time?

5. **Visualizing sensor data:** Another important tool included in ROS is *RVIZ*. This program is used to visualize various topics which are being published by currently running nodes. To run default configuration of *RVIZ* enter:

```
$rosrun rviz rviz
```

This opens the rviz window with the default configuration you can add elements to the visualization using the add button and selecting the type of topic to visualize. Try adding a *PointCloud2* topic to view the Kinect sensor output (*/camera/depth/points*) and an *Image* topic to view the depth image (/camera/depth/image_raw). Save this configuration, then restart rviz and load it automatically (see rviz User Guide if needed).

Tips: Each topic contains a reference frame identifier. These identifiers help RVIZ to align the various visualizations correctly using predefined transforms (tf). If your data is not being visualized correctly or is not being shown at all make sure that you have selected the correct frame.

Also, you may see an error "Fixed frame [map] does not exist." This error occurs because rviz uses [map] as a default frame and this frame is not yet being published as you are not running any mapping code. Either run gmapping (see below) or select another global tf in rviz other than map (such as base).

6. **Map building:** ROS has many built in libraries that perform many of the fundamental tasks for robotics that you will be learning in this course. The first that we will explore is mapping. In this lab we will use a prebuild node called *Gmapping* to map the area. More information on *Gmapping* can be found here. To do this run the follow:

```
$roslaunch turtlebot_example gmapping_sim_demo.launch
```

The robot is now running the *Gmapping* node and will map the simulated environment in 2D. If the above step fails with errors relating to symmetry of input scan, you may need to upgrade your version of gmapping manually to version 1.3.8. To do so, download the latest, compile and install.

```
$ wget http://packages.ros.org/ros/ubuntu/pool/main/r/ros-indigo-
gmapping/ros-indigo-gmapping_1.3.8.orig.tar.gz
$ cd ~/catkin_ws/src
$ tar xvf ../../ros-indigo-gmapping_1.3.8.orig.tar.gz
$ mv ros-indigo-gmapping-1.3.8/ gmapping
$ catkin_make
$ sudo cp devel/lib/gmapping/slam_gmapping* /opt/ros/indigo/lib/gmapping/
```

To view the map that is being generated an existing turtlebot *RVIZ* setup can be used. Therefore close any open *RVIZ* windows and run:

```
$roslaunch turtlebot_rviz_launchers view_navigation.launch
```

This avoids the need for you to setup the RVIZ environment manually and sets up the necessary visualizations.

Drive the simulated Turtlebot around using the tele-op node from Part 3 to fill in the map. Once the map is filled in save the map to a file using:

```
$rosrun map_server map_saver -f file_location
```

Once the file is saved, turn off the gmapping node (*Ctrl-C*). The map will be used in the next step.

Note: file_location does not need a file extension.

Note: The location at which the robot started is considered the origin of the map with the forward direction being the x axis.

Tips: The Gmapping algorithm builds the map by attempting to align the scans from the Kinect with the map. You will notice that if you are not looking at an area with sufficient features the map can align scans poorly and lead to divergence of the map. If necessary, add elements to the environment to improve mapping performance.

7. **Navigation and Planning:** The second fundamental task for a robot is navigation and planning. To perform this task we will again use a built in node which implements Adaptive Monte Carlo Localization (AMCL), and Trajectory Rollout planner (base_local_planner), you can follow the links to learn more about these fascinating algorithms. To run this node use:

```
$roslaunch turtlebot_example amcl_demo.launch map_file:=file_location.yaml
```

Note: The file_location.yaml specified above must be an absolute path (i.e.: /home/user_name/folder/filename.yaml)

When this node is initially started, the robot does not know where it is. You must first give the robot an initial estimate of its starting location in the map. To do this, in the RVIZ window

a. Click the "2D Pose Estimate" button
b. Click on the map where the TurtleBot approximately is and drag in the direction the TurtleBot is pointing.

The laser scan should now line up approximately with the walls of the map. Often, a small amount of motion may help the amcl node to correct its position. If not, repeat the position initialization step.

Once the Turtlebot is localized you can now give it way points to travel to. To do this:

a. Click the "*2D Nav Goal*" button
b. Click on the map where you want the TurtleBot to drive and drag in the direction the TurtleBot should be pointing at the end.

Try navigating the Turtlebot around the environment using waypoints.

Note: The turtlebot will not move if you still have a teleop node running.

8. **Changing node parameters:** All of the above nodes run based on a set of parameters which determine the operation of the node. These parameters can be changes in the launch file, using the *dynamic reconfigure node*, or using the command line. We will use the dynamic reconfiguration node:

```
$rosrun rqt_reconfigure rqt_reconfigure
```

From this interface you will have a list of all nodes which have configurable parameters. In your case the two main nodes of interest are the *AMCL* node and the *move_base* node which govern the localization and planner respectively. Try changing the *local_costmap* parameters such as so you can see the changes in the robot path. A description of all the amcl and move_base parameters are found on the ROS wiki.

9. **Compiling you own code:** In this step you will create your own custom node to control the Turtlebot. This node will be the basis for your future labs. In this node you will program the robot to drive in a square by reading in position estimates and publishing *cmd_vel* messages. The example package provided contains the skeleton code for what you will need. You should only need to modify the source file located at *./turtlebot_example/src/turtlebot_example_node.cpp.* The comments contained in the example file will direct you in how to setup your code.

Start by compiling the example code and confirming that it drives the turtlebot in a circle by running the node. To do this, you will need the simulator and the amcl_demo running with the map you computed, and you will need to initialize the robot pose per Step 7. Look through the code and decipher where it is getting its position information, and where it is sending its commands.

```
$rosrun turtlebot_example turtlebot_example_node
```

Next, create a new node to implement your square driving algorithm (be sure to add a new executable to your CMakeLists.txt). Decompose the problem and create a pseudocode plan before you start coding. What steps will you functionalize, how will you pass data from the callback to the main function? For debugging, use ROS_INFO or ROS_DEBUG messages to see the state of variables in your code during execution.

Run your package as you did the example code, using your pre-generated map and localization node. When complete, your robot should move in approximately a square pattern, limited by the quality and update rate of the amcl localization solution.

10. **Pose tracking:** All turtlebots come equipped with a unique marker called AprilTag (http://april.eecs.umich.edu/pdfs/olson2010tags.pdf). This visual fiducial marker system has a specific encoding for each "tag" that allows for 3-DOF localization of the robot (position : x,y and yaw) with respect to the camera (mounted above the TV in the SDC bay area). Connect the camera to the dedicated ME597 laptop and launch execute the following:

```
$roslaunch firefly pgr firefly pgr.launch
```

followed by the following in another tab (ctrl+shift+T opens a new tab):

```
$rosrun me597_april_tag_cpp me597_april_tag_cpp
```

Connect the ME597 laptop to the wireless router via the Ethernet cable provided. You now need to configure your laptop to communicate with the laptop on the turtlebot over the WaveQR network where the laptop on the turtlebot acts as the ROS Master. For a tutorial on how to how to start a ROS system using two machines please visit: http://wiki.ros.org/ROS/Tutorials/MultipleMachines.

Each turtlebot laptop has a designated IP address on the WaveQR network of the form: **172.24.84.*X***, where **X** is the laptop's number (1-6). To connect your laptop to the turtlebot laptop, connect to the WaveQR network and append these 2 lines to you .bashrc file:

```
export ROS_IP=whatever.your.ip.is
export ROS_MASTER_URI=http://172.24.84.X:11311
```

Next, run the *turtlebotPosClient_node* on the turtlebot's laptop so that it can receive information from the ME597 laptop:

```
$rosrun turtlebotPosClient turtlebotPosClient_node
```

Once all set, you should be able to now view */indoor_pos* topic on the turtlebot.

Pose information received by the *turtlebotPosClient_node* is published as a ROS topic : */indoor_pos*. You can view the topic by typing the following command:

```
$rostopic echo /indoor_pos
```

11. **Show the TA your work:** You are now ready to move on to working with the real robot! Show the TA your working code and they will give you a Turtlebot to continue.

12. **Running the real robot:** You are now going to run you code you made in Part 9 on the robot hardware. All of the necessary packages should already be installed on the Turtlebot netbook computer. Simply transfer your code to the Turtlebot computer and perform steps 6-9 on the robot except instead of running the simulator run:

```
$roslaunch turtlebot_bringup minimal.launch
```

13. **Show the TA your work:** Congratulations, you have completed the first of the ME 597 labs. We encourage you to continue to experiment with ROS and the Turtlebot simulator as there are many other interesting packages available.

Note on future labs: This lab was meant to introduce you to ROS and the Turtlebot and walk you through the steps to get everything up and running. In the future it will be assumed that you are familiar with using ROS and the lab manual will not include such specific instructions.

Optional Extensions

For those who would like to explore some additional packages and functionality of the robot the following options may be of interest. Setting up these optional extensions can sometimes be challenging and will not be covered by the TAs or this lab manual.

A. **Turtlebot Follower:** The Turtlebot Follower package is a fun little program which makes the Turtlebot follow and object that is in front of it at a set distance. The documentation can be found here. This is a fun and easy extension to enable as it should work out of the box. If you have extra time in your lab we encourage you to try it!

B. **Using rosbag:** Rosbags or simply bags are a form of data collection and storage that ROS implements which records all or some of the topics that are being published at a given time and can be replayed exactly at will. Usage details can be found here.

C. **SLAM Extensions:** There are many different methods and version of SLAM algorithms. You can find several different ones in the ROS repositories. Two algorithms of interest that use the Kinect sensor are RGBD-SLAM and KinectFusion. These algorithms can perform full 3D dense mapping in real time.

Lab Report Instructions:

For this lab **no report is required**! You must simply show the TA all items on the checklist below have been completed and submit your single source code file from the turtlebot square demo to the dropbox on Learn for review by your fellow students.

Lab Marking Scheme:

The lab mark will be based on completion of all of the steps detailed above. TAs will check-off your group for achieving the following milestones:

Simulator:

- ☐ Simulator is running correctly
- ☐ Map has been generated
- ☐ Robot localized correctly
- ☐ Robot drives in a square pattern

On the Turtlebot:

- ☐ Robot mapped an area
- ☐ Robot pose detected with respect to the camera
- ☐ Robot drives in a square pattern