ME 597 Lab #2:
Mapping and Localization

Introduction:

In this lab you will be creating two ROS nodes which will perform mapping and state estimation separately using the Turtlebot's onboard sensors and the vision system, with the option to try simultaneous localization and mapping (SLAM) as a bonus. SLAM is a keystone component for any autonomous system working in unknown environments and allows the robot to explore areas not previously seen or mapped. SLAM requires mapping and state estimation methods to be combined into a common algorithm. To keep the lab tractable, we will separately code mapping and particle filter state estimation on the Turtlebots. If either package is coded poorly, your code will run quite slowly on the netbooks and will drastically degrade your results. These types of limitations are common in robotics applications, so this should be good experience for future robotics projects.

This lab requires a short lab report to be submitted. Section 5 details what is required in the report. **It is highly recommended that you complete the simulated portion of the lab prior to coming to your lab times** as you will want to use your lab time to work with the live robot. In Lab 2, there will be more testing and modification on the robot, to get things to work properly with the true sensors.

Description:

The main components of this lab are the implementation of the occupancy grid mapping algorithm and particle filter localization algorithms as described in class. Lecture notes are available which discuss the implementation of these methods. There will be no predefined map given for this lab. You may create your own environments to use for this lab. It will be your task to explore and map the area you create. Instructions on reading the various sensors of the Turtlebot are discussed in the lab setup section below. Note that only linear algebra related 3rd party packages are allowed (e.g. eigen) to be used in this lab, and you may use any code you have created in the previous labs. Position information will be given for both localization and mapping.

1) Mapping: Using accurate IPS position measurements and Kinect data, construct a 2D occupancy grid of a large portion of the field of view of the IPS system. Perform the log odds update, using Bresenham's line drawing algorithm for the Kinect scan points. You may downsample the measurements or aggregate them, and you do not need to use all 30 Hz worth of measurements. Select the appropriate modifications so that your code can run on the netbooks in close to real time and produce a good map of the surroundings.

2) State estimation: Using degraded IPS information and wheel odometry, perform particle filter based state estimation. Compare results using the true vision data and 1 Hz position updates plus an additional position measurement noise with a standard deviation of 10 cm. Identify the number of particles needed to maintain a good motion estimate.

3) SLAM: (Bonus, worth 2% on final grade) – Combine occupancy grid mapping with particle filter localization to perform SLAM on the Turtlebots. Without using the IPS, use ICP to estimate the position of the robot. This is a big challenge!

Lab Setup:

This section describes various useful procedures for the lab. Please be sure to read this section carefully as it will make the lab run significantly more smoothly for you.

Simulation Position and Environment (Simulation Only)

As there is no IPS system in the simulation you will use the actual position provided by the simulation environment. This is in some respects considered "cheating" as the simulation position will be perfectly accurate since it is the position Gazebo is using to generate the simulation results. As such you should remember that when switching to the actual robot your position information may contain noise not present in the simulation. The topic used to subscribe to the position in the simulation is */gazebo/model_states*; however this is a wrapper of all the model positions so it is slightly non-intuitive. An example of subscribing to this topic is included in the new example package on learn. This topic will be available once Gazebo is running, no additional nodes are needed.

IPS setup (Live Only)

The same IPS system used in the previous lab will again be used for this lab. See Lab 1 for instructions on connecting to the lab's wireless network and for use the multiple machine configuration to control the turtlebot from your personal machine. Although multiple robots will be able to function on the course at the same time, since you will be attempting to map the course please be considerate and attempt to not interfere with each other's work.

Reading Kinect Laser Scans

As you saw in the first lab, the Kinect sensor produces dense 3D point clouds as its output; however, for the purposes of this lab we will treat the Kinect as a 2D laser scanner in order to simplify the implementation.

To enable the Kinect run the following command after you have initialized the turtlebot (only needed on live robot).

```
$roslaunch turtlebot_bringup 3dsensor.launch
```

This node will publish the topic */scan* which is of the type *sensor_msgs::LaserScan*. This message has an array of *ranges* which correspond to angles (relative to heading of the robot) ranging from *angle_min* to *angle_max* in radians. The same topic is available in the Gazebo simulation.

Reading Wheel Odometry

The *minimal.launch* script provides odometry information which will be used in your particle filter solution. The Turtlebot publishes the message */odom* of type *nav_msgs::Odometry*. The *twist* field of this message contains the estimated velocity of the robot at the given time step. The *pose* field contains the position of the robot relative to the starting position obtained by integrating the velocity messages since the robot was first turned on. Note that when using the pose message you most likely want to take relative transforms not the total integrated pose. You may use either the *pose* or *twist* message in your implementation.

Generating and Outputting the Map

You will most likely store your map in a 2D array with known resolution and size as you did in the previous lab. In this lab, instead of reading the map however you will be publishing it. The message type used to publish the map is the *nav_msgs::OccupancyGrid*. When publishing the occupancy grid make sure you the fill in all of the fields in the message, including those in the *nav_msgs::MapMetaData* structure to ensure it visualizes correctly. Also ensure that the data in your array is formatted in the same order as in the OccupancyGrid message.

Outputting the Path

As in the previous lab you will want to output the path of your robot as it moves around the environment. This time the robots position will have been calculated by your estimation algorithm. As in the previous lab you may continue to use the *visualization_msgs::Marker;* however, since you will only need to plot a single path it may be easier to use the *nav_msgs::Path*, which is simply a vector of *geometry_msgs::PoseStamped* messages which you will most likely want to generate for visualization anyway.

Outputting the Particle Filter

To output the particles of the particle filter for visualization in RVIZ you may use the *visualization_msgs::Marker* message as seen in the previous lab. To output points simply use the *visualization_msgs::Marker::POINTS* type instead of a *LINE* type. An example tutorial for using points can be found here.

Lab Instructions:

1. First implement both your mapping and particle filter solutions in simulation. You may use one of the previously provided gazebo worlds, modify an existing world or create your own. Many parameters of the algorithms are left up to your discretion such as grid resolution, map update steps, occupancy thresholds, and others. It is suggested that you make these parameters which are easily tuneable in the launch file or through dynamic_reconfigure as they will be discussed in the report. To test your algorithm simply tele-op your robot around the environment and make sure it is mapping and estimating correctly.

   Note: The world which you create to explore is up to you; however, it should be at least 100m^2 is area and contain some interesting structure to map.

2. Once you have completed the algorithms in simulation you may now run them on the actual turtlebot. In both cases, you may simply tele-operate the Turtlebot to move it about. You may make an environment out of any available materials in the SDC.

3. Before you leave the lab ensure that you have collected all the necessary data, plots, and information you will need in order to complete the required lab report. This should include at least: the generated map created by your robot at 3 stages of completeness, the state estimates of the path traversed by your robot for state estimation, any additional information needed to properly describe the methods used and the results achieved, and finally an image of the environment you used. Please be sure to include any additional information you think will improve your report.

Lab Report Instructions:

The write up should be approximately 2-3 pages long not including appendices, with a 4 page maximum. Please submit a single PDF to the Dropbox on Learn. The report should have the following information, organized into sections however best fits your methods. There is no need to describe the mapping and filter algorithms in detail as they are in the course notes, but instead focus on the implementation details, parameter selections and decisions that you made in the lab, the reasoning behind why those decisions were made and the results as described below.

1. Theory: The goal of this section is to summarize how you did each section of the lab, so we can assess if you understood the underlying principles. Briefly present the main ideas of the lab, the measurement and motion models used.

2. Implementation: Describe any implementation decisions that your group made that are not a part of the two algorithms. This should include but is not limited to: decisions on parameter tuning, efficiency concerns, and utilization of different sensors. Include any other details which you used to improve your implementation.

3. Results and Discussion: For each section of the lab, present the simulation and experimental results, and discuss any of the major discrepancies between the two. Describe whether these results bore out in the implementation of the algorithms. Also describe any limitations that inhibited performance of the algorithms. What would you change about the current system, algorithm, or implementation to improve the performance?

4. Source Code: The source code for you implementation should be included as an appendix. Only source files and headers (.cpp, .c, .hpp, .h) are require. Please do not include any makefiles or configuration files.

This is perhaps a more vague description of the lab report and a different way of writing reports than you may be used to. The idea is to make you think like an academic, to present your ideas in a way that convinces everyone of their validity and does so quickly, without a lot of excessive detail. Since the work you are doing is

coming so close to the cutting edge, it only seems natural to have you write this way. Feedback and questions are welcome!

Lab Report Marking Scheme:

- Content & Results: 60% total (~20% each)
  - Background and Theory
  - Implementation Details
  - Results
- Discussion of simulation and experimental results, pros and cons of methods, issues, limitations, possible improvements : 20%
- Presentation: 20%
  - Nicely formatted, either single or double column, no need to double space
  - 4 page limit observed
  - Figures, tables labeled, no superfluous figures, no excessive displays of raw data
  - Appropriate references provided, if necessary
  - No table of contents needed