MIE443H1S: Contest 1

Where am I? Autonomous Robot Search of an Environment

1.1 Objective:

To use the TurtleBot to autonomously drive around an unknown environment while using the ROS gmapping libraries to dynamically create a map using sensory information from the Kinect sensor. Your team will be in charge of developing an algorithm for robot exploration that can autonomously navigate an environment using sensor data without human intervention. The robot will need to navigate and explore the environment within a time limit. Team scores will be determined based on the percentage of the environment mapped within the time limit.

1.2 Requirements:

The contest requirements are:

- The TurtleBot has a time limit of 8 minutes to both explore and map the environment.
- The robot must perform the overall task in autonomous mode. There will be NO human intervention with the robot.
- You must use sensory feedback on the TurtleBot to navigate the environment. The robot cannot use a fixed sequence of movements without the help of sensors.
- There must be a speed limitation on the robot that will not allow it to go any faster than **0.25m/**s when it is navigating the environment and **0.1m/s** when it is close to obstacles such as walls. This is to ensure consistency in mapping. Note that if the robot moves any faster, there will be more errors in mapping.
- Once the 8 minute exploration time is completed (or if your robot is done sooner), you will save the map your TurtleBot has generated and provide this map to the Instructor/TAs.
- The contest environment will be contained within a 4.87x4.87 m^2 area with static objects in the environment. An example environment is shown in Figure 1 (not the layout on contest day).

Figure 1: Example Environment

- The exact layout of the environment will not be known to your team in advance of the contest. Therefore, please ensure that your exploration algorithm is robust to unknown environments with static obstacles.

1.3 Scoring:

- Your team will be given a total of 2 trials. Each trial will have a maximum duration of 8 minutes. The best trial will be counted towards your final score.
- Scoring is based on the percentage of the environment mapped (10 marks) and the mapping of key obstacles (5 marks) within the environment during exploration. Namely, the map generated by your robot will be compared to a ground truth map of the environment. Figure 2 shows an example of a generated map that would be submitted at the end of the contest. In the case of this map the key obstacles are the garbage cans which can be seen at points 1 and 2.

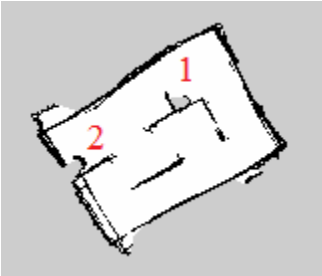

Figure 2: Example Map Generated of the Environment

1.4 Procedure:

Code Development - The following steps should be followed to complete the work needed for this contest:

- Download the mie443_contest1.zip from Quercus. Then extract the contest 1 package (right click > Extract Here) and then move the mie443_contest1 folder to the catkin_ws/src folder located in your home directory.
- In terminal, change your current directory to ~/catkin_ws using the cd command. Then run catkin_make to compile the code to make sure everything works properly. ****Please note if you do not run this command in the correct directory a new executable of your program will not be created.****
- Refer to the contest1.cpp file located in the code package at the following address ../mie443_contest1/src. The majority of development for this contest will be done in the contest1.cpp.
- Code Breakdown – contest1.cpp
    - These are the header declarations that allow you to use the required libraries in this contest:

```
#include <ros/console.h>
#include "ros/ros.h"
#include <geometry_msgs/Twist.h>
#include <kobuki_msgs/BumperEvent.h>
#include <eStop.h>

#include <stdio.h>
#include <cmath>

using namespace std;
```

- Sensor Implementation
  - These are the callback functions that are run whenever the subscribers that are declared in the main block of code are triggered. The variable msg is an object that contains the data pertaining to the instance of each respective callback press:

```
void bumperCallback(const kobuki_msgs::BumperEvent::ConstPtr
msg){
	//implement your code here
}

void laserCallback(const sensor_msgs::LaserScan::ConstPtr
msg){
	//implement your code here
}
```

  - Main block and ROS initialization functions are provided below:

```
int main(int argc, char **argv)
{
	ros::init(argc, argv, "image_listener");
	ros::NodeHandle nh;
```

  - The first two lines in the following block are the declarations of the subscribers that are used to return sensor data from (1) the bumper on the robot base and (2) the laser scan information from the Kinect sensor. The third line is the publisher that you will use to advertise wheel speed commands to the ROS framework.

```
	ros::Subscriber bumper_sub =
nh.subscribe("mobile_base/events/bumper", 10, &bumperCallback);
	ros::Subscriber laser_sub =
nh.subscribe("/scan", 10, &laserCallback);


	ros::Publisher vel_pub =
nh.advertise<geometry_msgs::Twist>("cmd_vel_mux/input/teleop",1
);
```

- Declaration of speed variables and the Twist Class. Twist is a type of descriptor that defines the velocities of a robot in 6 Degrees-of-Freedom. Three degrees for linear vectors XYZ and 3 degrees for vectors for Yaw, Pitch, and Roll.

```
double angular = 0.0;
double linear = 0.0;
geometry_msgs::Twist vel;
```

- Controller Design and Implementation
  - The following starts the timer that keeps track of the total amount of time the robot has spent exploring the environment.

```
// contest count down timer
std::chrono::time_point<std::chrono::system_clock> start;
start = std::chrono::system_clock::now();
uint64_t secondsElapsed = 0;
```

  - The following is the main while loop. The program remains within this loop until a termination signal is received (sigterm, sigkill, etc.) or 480 seconds have elapsed. The *ros::spinOnce()* checks the ROS framework to see if new sensor values have been published to the running topics and whether the registered callback functions need to be called. The angular and linear values are used to set the respective speeds in the Twist class declared in the previous block. The Twist class is then published to the ROS framework and is used by the TurtleBot base to drive the robot.

```
while(ros::ok() && secondsElapsed <= 480) {
      ros::spinOnce();
      //fill with your code
      vel.angular.z = angular;
      vel.linear.x = linear;
      vel_pub.publish(vel);
      // The last thing to do is to update the timer.
      secondsElapsed =
      std::chrono::duration_cast<std::chrono::seconds>(std::chron
      o::system_clock::now()-start).count();
      loop_rate.sleep();
}
```

- Every time any file in your project is changed, you must compile it from terminal in the catkin_ws directory using the catkin_make command. If not, the executable will not contain your modifications.
- Robot Simulation in Gazebo – You can use Gazebo to implement and test your TurtleBot code prior to integrating it onto the physical robot. The following steps should be followed in order to do this:
  - Begin by launching the simulated world through the following command:

```
roslaunch mie443_contest1 turtlebot_world.launch world:=1
```

If there are other environments you want to simulate in the ../worlds folder, you can change the world argument number in the command to load the corresponding environment.

- To perform the ROS gmapping algorithm in simulation instead of on the physical TurtleBot use the following command:

```
roslaunch turtlebot_gazebo gmapping_demo.launch
```

  All other commands to view and save the map are the same as in the gmapping tutorial.

- To visualize robot information (i.e., position, sensor reading), run the following command in a separate terminal window:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

- Finally, to run your code in simulation it is the same as running it on the physical TurtleBot:

```
rosrun mie443_contest1 contest1
```

- Robot Execution - When you are ready to run your program on the physical TurtleBot, the following steps should be followed:
  - Connect the two USB cables on the TurtleBot to the laptop. One USB cable is for the Kobuki base, and the other USB cable is for the Kinect sensor.
  - With the Kobuki base and laptop turned on, run the following commands in separate terminals:
  - The below command starts the TurtleBot Kobuki base and starts publishing topics:

```
roslaunch turtlebot_bringup minimal.launch
```

  - When all processes are running properly and you are ready to begin mapping, run the following command in a new terminal:

```
roslaunch turtlebot_navigation gmapping_demo.launch
```

  - This command opens a plugin that visualizes the data that the robot is publishing and subscribing to.

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

  - Finally, to execute your code, run the following line in a new terminal:

```
rosrun mie443_contest1 contest1
```

  - If all is working correctly, your robot will begin navigating the environment and collecting map data.

- At the end of the allotted contest time or if you are finished before the contest time, **save the map** you have created by calling the following command in a new terminal (please type out the command instead of copy/paste to avoid formatting issues):

```
rosrun map_server map_saver -f /home/turtlebot/
```

- When you are done, cancel all processes by pressing Ctrl-C in their respective terminals, prior to closing them.

1.5 Report:

- The report for each contest is worth half the marks and should provide detailed development and implementation information to meet contest objectives (15 marks).
- Marking Scheme:
  - The problem definition/objective of the contest. (1 mark)
    - Requirements and constraints
  - Strategy used to motivate the choice of design and winning/completing the contest within the requirements given. (2 marks)
  - Detailed Robot Design and Implementation including:
    - Sensory Design (4 marks)
      - Sensors used, motivation for using them, and how are they used to meet the requirements including functionality.
    - Controller Design, both low-level and high-level control (including architecture and all algorithms developed) (5 marks)
      - Architecture type and design of high-level controller used (adapted from concepts in lectures)
      - Low-level controller
      - All algorithms developed and/or used
      - Changes and additions to the existing code for functionality
  - Future Recommendations (1 mark)
    - What would you do if you had more time?
    - Would you use different methods or approaches based on the insight you now have?
  - Full C++ ROS code (in an appendix). Please do not put full code in the text of your report. (2 marks)
  - Contribution of each team member with respect to the tasks of the particular contest (robot design and report). (Towards Participation Marks)
  - The contest package folder containing your code will also be submitted alongside the softcopy (PDF version) of your report. (Towards the Code Marks Above)
- Your report should be a maximum of 20 pages, single spaced, font size 12 (not including appendix).

1.6 Reference Materials for the Contest:

The following materials in Appendix A will be useful for Contest 1:

- A.1.1
- A.1.2
- A.1.3
- A.2.1
- A.2.2