MIE443H1S: Contest 1

Where am I? Autonomous Robot Search of an Environment

1.1 Objective:

To use the simulated TurtleBot to autonomously drive around an unknown environment while using the ROS gmapping package to dynamically create a map using sensory information from the Kinect sensor and other sensors available to you on the TurtleBot. Your team must develop an algorithm for robot exploration that can autonomously navigate a gazebo environment. 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 and the detection of key obstacles within the time limit.

1.2 Requirements:

The contest requirements are:

- The simulated TurtleBot has a maximum time limit of **15 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 limit 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 15-minute exploration time is completed (or if your robot is done sooner), the robot must stop moving.
- The contest environment will be contained within a 6x6 m^2 3-dimensional simulated environment with static objects. An example environment is shown in Figure 1 (not the layout for the contest day). **NOTE:** The red brick wall cannot be seen by the robot.

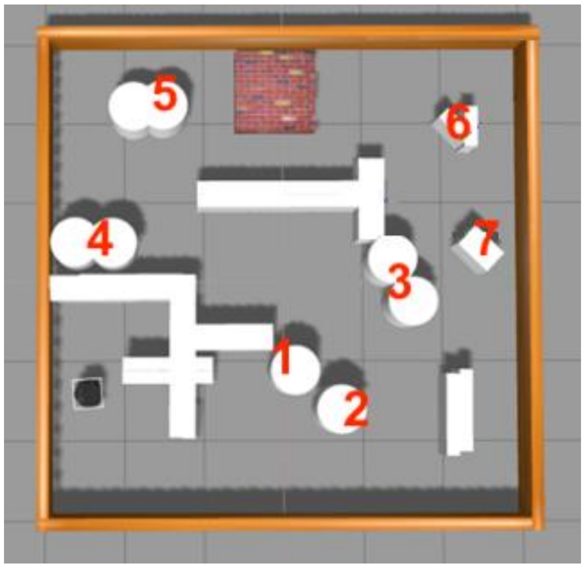

Figure 1: Example Environment

- 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 (15%):

- Your team will be given a total of 2 trials. Each trial will have a maximum duration of 15 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 a number 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, as an example, at the end of the contest, for Figure 1. In the case of this map the key obstacles are the cylinders and boxes distributed in the scene, which can be seen at points 1 - 7.

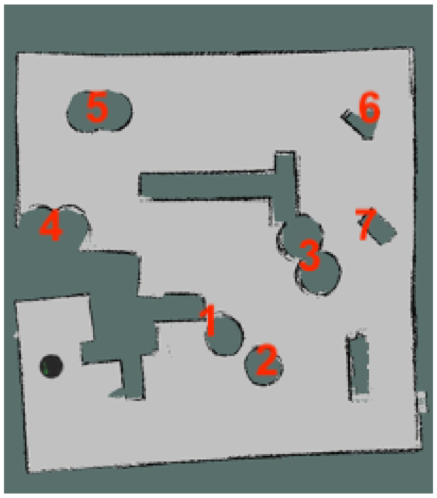

Figure 2: Example Map Generated of the Environment in Figure 1

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.****

- Robot Simulation in Gazebo – You will use Gazebo to implement and test your contest code. 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:=practice
```

      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 use the ROS gmapping module in simulation use the following command:

```
roslaunch mie443_contest1 gmapping.launch
```

      All other commands to view and save the map are the same as in Tutorial 2.
    - Finally, to run your code in simulation:

```
rosrun mie443_contest1 contest1
```

    - 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:

```
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.
- Refer to the contest1.cpp file located in the code package at the following directory ../mie443_contest1/src. Most development for this contest will be done in the contest1.cpp.
- Code Breakdown – contest1.cpp
    - These are the header files that are needed in order to use the required libraries for this contest:

```
#include <ros/console.h>
#include "ros/ros.h"
#include <geometry_msgs/Twist.h>
#include <kobuki_msgs/BumperEvent.h>
#include <sensor_msgs/LaserScan.h>
#include <stdio.h>
#include <cmath>
#include <chrono>
```

- Sensor Implementation
    - These are the callback functions that are run whenever the subscribers that are declared in the main block of code receive a message. The variable msg is an object that points to the data pertaining to each callback:

```
void bumperCallback(const kobuki_msgs::BumperEvent::ConstPtr&
msg)
{
      //fill with your code
}
void laserCallback(const sensor_msgs::LaserScan::ConstPtr& msg)
{
      //fill with your code
}
```

    - 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 simulated robot base and (2) the simulated laser scan information. 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.

```
float angular = 0.0;
float 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 900 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 simulated TurtleBot base to drive the robot.

```
while(ros::ok() && secondsElapsed <= 900) {
        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 a 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.

1.5 Code Submission:

- In the README.md file, please make sure you include the following:
  - Group Number and all names of students in your group.
  - The list of commands that we need to run in order to execute your group's code (as needed in consecutive order).
  - Any computer specific *file path string* that we need to modify for your code to run.
  - Where exactly the output file will be stored.

1.6 Contest 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)

            - All Sensors used, motivation for using them, and how are they used to meet the requirements for the contest 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 <u>do not</u> 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 <u>contest package folder</u> containing all your code, README file, etc. that will also be submitted alongside the softcopy (<u>PDF version</u>) of your report. Check and make sure your code runs before submitting (Towards the Contest and Code Marks Above)

Your report should be a maximum of **20 pages**, single spaced, font size 12 (not including appendix).

1.7 Individual Report (One per group member): Each group member will also submit an individual report answering a set of questions on Contest 1 development to be completed on your own. These questions will count towards your individual Participation Marks (10%).

1.8 Reference Materials for the Contest:

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

- A.1.1
- A.1.3