BLG456E, Robotics, Fall 2014

Assignment #3:

Exploration

Lect. Dr. Damien Jade Duff

Res. Asst. Çağatay Koç

Due: Nov 23 (Sunday) 11:59pm

Note: Your assignment mark will be the best two of three marks, so you may choose not to attempt this assignment if you have already collected sufficient marks from the first two assignments and you have not the time.

Assignment summary:

In this assignment you need to to design and implement a robot controller node that is able to explore to discover a map of the environment. While discovering the environment, the robot should not collide with any obstacle and should be directed to undiscovered areas of the environment. It should also not leave a prescribed search area (to be decided by the student but should include all obstacles in the simulation). The same robot (Turtlebot) and simulator (ROS/Gazebo) will be used as in the previous assignments.

You will use the output of the *slam_gmapping* node which takes in *sensor_msgs/LaserScan* messages and builds a map of type *nav_msgs/OccupancyGrid*.

Submission details: An archive (ZIP) file of the source code. The archive will contain the following files (replacing **{student ID}** with your ID):

```
a3_{student ID}/src/a3_{student ID}_node.cpp
a3_{student ID}/CMakeLists.txt
a3_{student ID}/package.xml
```

NOTE: do not create the ZIP file from within the `a3_{student ID}` directory – the archive file should contain the directory.

It should be possible to compile the package by unzipping the ZIP file into a catkin workspace and running the catkin compilation.

Step-by-step

Basic knowledge

You can start the turtlebot simulation as before:

```
roslaunch turtlebot_gazebo gmapping_demo.launch
```

You can manually drive the robot as before:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

You can start the mapping node like this:

```
roslaunch turtlebot_gazebo gmapping_demo.launch
```

You can use rviz to visualise the map like this:

```
roslaunch turtlebot_gazebo turtlebot_playground.launch
```

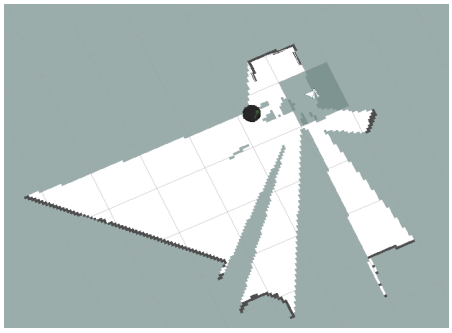

Figure 1: Example rviz visualisation of map being created by slam_gmapping

See Figure 1 for an example of how this visualisation should look.

Create your package

Your catkin package should have the following dependencies:

`std_msgs nav_msgs roscpp gmapping geometry_msgs sensor_msgs tf`

Don't forget to setup `CMakeLists.txt` and the `add_executable` and `target_link_libraries` lines so that your source file gets built.

Acquire the map

You can acquire laser scans and send movement commands as you did in the first assignment.

You will need to process laser scans as you did in the first assignment in order to avoid obstacles. Additionally, you will need to receive and process map messages.

You will need to subscribe to the map callback (here `n` is your node handle object):

```
ros::Subscriber map_reader = n.subscribe("/map",3,mapCallback);
```

The map callback function will have the signature:

```
void mapCallback(const nav_msgs::OccupancyGrid::ConstPtr& msg)
```

Understand the map

Information about how to use the occupancy map message can be found here:

http://docs.ros.org/indigo/api/nav_msgs/html/msg/OccupancyGrid.html

In order to access map information you can use the following expressions:

Height in pixels: `msg.info.height`
Width in pixels: `msg.info.width`
Value (at X,Y): `msg.data [ X*msg.info.width + Y ]`
Meaning of value:
`0` = *fully free space.*
`100` = *fully occupied.*
`-1` = *unknown.*
Coordinates of cell 0,0 in `/map` frame:
`msg.info.origin`
The size of each grid cell:
`msg.info.resolution`

Some sample code for making use of that occupancy grid message follows:

```
vector<signed char > map =  msg->data;
int unknownArea=0;
int allArea=0;
int occupiedArea=0;
for(size_t i=0; i< map.size(); ++i ){
```

```
        int map_x = i % msg.info.width;
        int map_y = i / msg.info.height;
        allArea++;
        if(map[i]>50) // 0 – 100 represents how occupied
              occupiedArea++;
        if(map[i]<0) // -1 means unknown
              unknownArea++;
        }
}
ROS_WARN("Discovered obstacle area: %d",occupiedArea); //Success Measure
ROS_WARN("Unknown area: %d",unknownArea);
ROS_WARN("All area: %d",allArea);
ROS_WARN("Pose: x=%f,y=%f,theta=%f",msg->info.origin.position.x,msg-
>info.origin.position.y,2*asin(msg->info.origin.orientation.z)*(msg-
>info.origin.orientation.w<0?-1:1));
```

Additional information about how slam_gmapping works can be found here:

http://wiki.ros.org/slam_gmapping

You will also need to work in the `/map` frame, and you can transform between the `/map` and robot (`/base_link`) frames using the teqhniques you learnt in assignment 2 (*tf library*). The map message contains the origin of the map `msg.info.origin` which corresponds to map entry `map[0][0]` in the `/map` frame of reference, and each map entry (cell) is `map.info.resolution` in width and height.

Marking criteria

- Accessing map data correctly and using it.
- Mapping between map and robot frames.
- Avoiding obstacles.
- Moving intelligently.
- Moving intelligently accounting for the map.
- Exploration accounting for the map.
- Intelligent exploration.
- Clear code, lack of redundant or incorrect expressions.
- Lack of errors.

Bonuses avaiable for:

- Novelty.
- Speed / exhaustiveness.
- Theoretical basis.
- Extra features.

Assignments not submitted according to requirements will not be evaluated.

Class competition

There will be a ranking of assignments in the class according to how much obstacle area can be discovered in 1 minute and 10 minutes. There may nor may not be prizes of chocolate, carrots, or similar.