MIE443H1S: Contest 2

Finding Objects of Interest in an Environment

1.1 Objective:

The goal of this contest is for the TurtleBot to navigate an environment to find and identify five objects placed at different locations in the environment, and return back to its starting position when the robot is finished. For identification purposes, each object will have a feature tag placed on it. The coordinates of each object and the map of the environment will be provided to your team in advance. Your team will be in charge of programming the TurtleBot to navigate itself to locations in the environment where it can view and identify each object, and then return to its starting location, all within a specified time limit.

1.2 Requirements:

The contest requirements are:

- The contest environment will be 4.87×4.87 m^2. For simplicity, there will be no additional objects in the environment other than the five objects to be examined as shown in Figure 1. Each object is represented by a cardboard box of dimensions 50x16x40 cm^3 (l x w x h). The coordinates of these objects, measured with respect to the world coordinate frame of the map, will be provided by the Instructor/TAs on contest day. A test set of object coordinates will be provided for development purposes during the lab sessions.

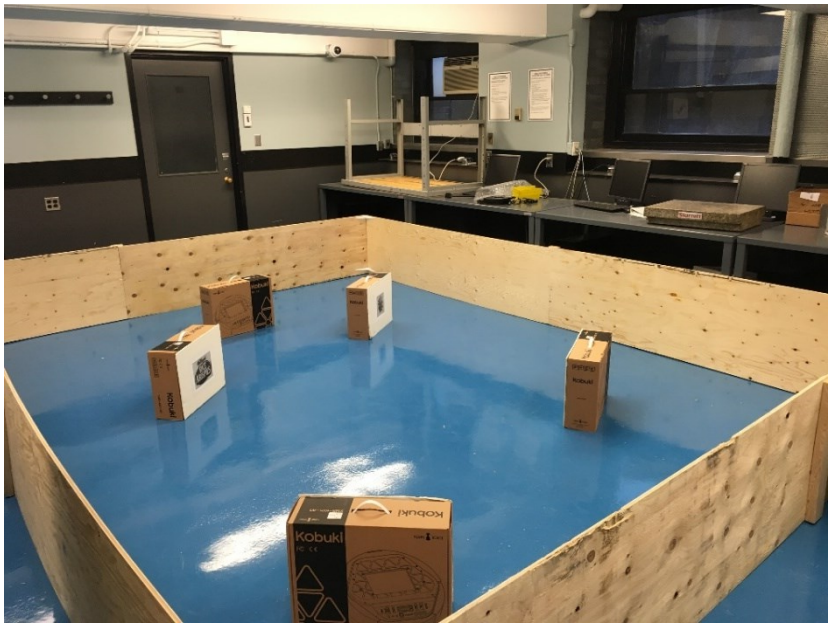

Figure 1: Example Environment Layout with 5 Objects

- Your team will be provided with: 1) a 2D map of the contest environment generated with gmapping, 2) test locations for the 5 objects within that environment, and 3) the image tags that will be used during the contest.
- An object's location in the environment map is defined by the coordinates of its centre and its orientation (x,y,φ), where φ is about the z-axis, with respect to the origin of the given 2D map. The origin of the world coordinate frame is determined by the TurtleBot's starting location when the map was created, see Figure 2. The origin is a native property of the map and all distances within the map coordinate frame are measured with respect to it. For more information on the world coordinate frame for the map please refer to the reference material in section A.1.2 in the appendix.
- The TurtleBot does not have to start at the origin of the world coordinate frame when the map is loaded; however, its pose within the map, just like the objects, will be measured with respect to this origin.

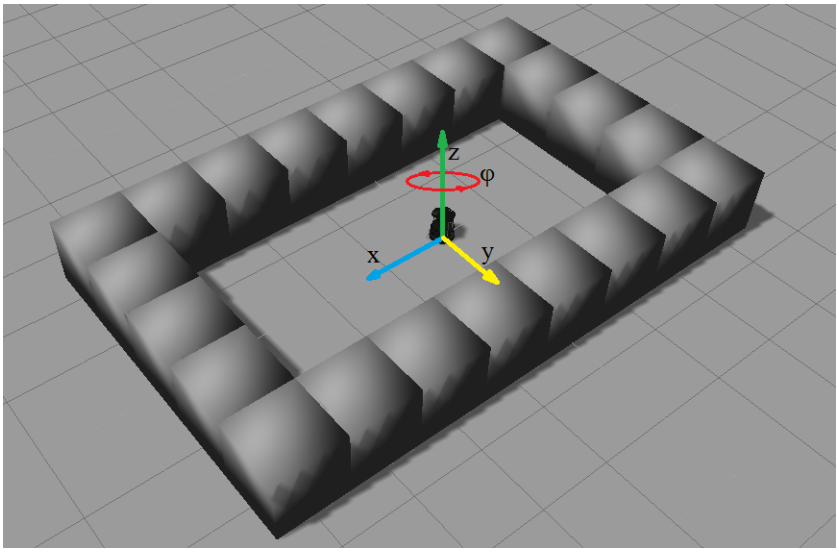

Figure 2: Example World Coordinate Frame

- The object locations are defined by two vectors: 1) a coordinate vector which defines each object's location in x and y, and 2) an orientation vector which contains the object rotation about the z axis. These locations are measured from the object's local frame (Figure 3) with respect to the world coordinate frame (Figure 2) at the origin of the map.
- The image tags are high contrast images with many unique features. An example is shown in Figure 4. These tags will be placed in the centre of one of the long faces of the objects, Figure 3.

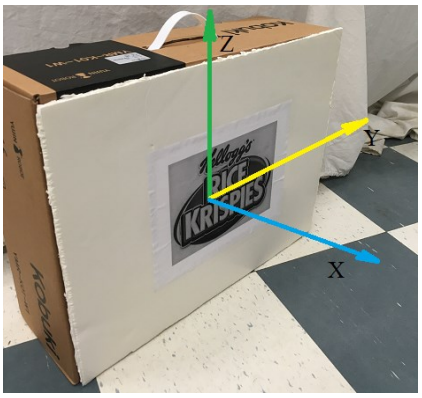

Figure 3. Object Coordinate Frame

Figure 4. Image Tag Example

- On contest day, your team will receive a new set of object locations and map to utilize for the contest. Please ensure that your code is robust enough to handle such dynamic changes to the locations of the objects. There will be no additional objects in the environment.
- During the contest, the TurtleBot will start at a location in the map of the Instructor/TAs choosing. The robot will have a maximum time limit of 5 minutes to traverse to and identify all the objects provided, before returning to its starting location and indicating that it is done.
- Your TurtleBot must utilize the provided navigation library to drive the robot base safely and use the RGB camera on the Kinect sensor to perform SURF feature detection in order to find and identify the image tag on each object.
- For this contest, there will be three objects in the environment with unique tags, one object with a duplicate tag, and one object without a tag, giving us a total of 5 objects.
- Once the TurtleBot has traversed to all the objects and returned to its starting location, it must output to a file (http://www.cplusplus.com/doc/tutorial/files/) all the tags it has found and at which object location (i.e., tag+location). This information will be used for scoring.
- In order to reduce run time, your team is strongly encouraged to optimize the robot's path to find the objects.

1.3 Scoring (15%):

- Your team will be given a total of 2 trials. The best trial will be counted towards your final score.
- The scores for each trial are as follows. One mark will be provided for the navigation to each object location (5 marks in total). Two marks will be given if your TurtleBot is able to correctly identify the image tag on an object at each location (including determining duplicate tags) or determine that there is no tag on the object (10 marks in total).

1.4 Procedure:

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

- Download the mie443_contest2.zip from Quercus. Then extract the contest 2 package (right click > Extract Here) and then move the mie443_contest2 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.****
- It is recommended that robot navigation for this contest be handled by the native TurtleBot library called *move_base*. This library takes coordinates and orientations in the global map of the environment and drives the robot to a specified location while handling robot localization and obstacle avoidance. For more information please refer to the reference material in section A.1.3 in the appendix.
- For tag feature detection, it is recommended to use the OpenCV feature finding framework. Please refer to the reference material in section A.2.4 in the appendix for more information and tutorials with regards to this library. Please keep in mind that the OpenCV feature finding tutorial is only programmed to handle a single image at a time, and therefore, it is important to understand how it works so that it can be implemented for the purpose of this contest with as few repetitive variable declarations as possible.
- Refer to the contest2.cpp and imagePipeline.cpp file located in the mie443_contest2/src/ folder. For readability and ease of troubleshooting it is important that your team develops the navigation code and the image processing code in these respective files. Contest2.cpp will handle the robot navigation code and then will call a function from imagePipeline.cpp that handles the image processing.
- **PLEASE DO NOT** make changes to any other files.

Code Breakdown – contest2.cpp

- Sensor Implementation - Odometry
    - This is the callback function that is run whenever the robot's pose (x, y, φ) in the map is published. This callback is found in the robot_pose.cpp file. The variable msg is an object that contains the data pertaining to the robot's current position and orientation.

```
void RobotPose::poseCallback(const geometry_msgs::PoseWithCovarianceStamped& msg) {
      phi = tf::getYaw(msg.pose.pose.orientation)
      x = msg.pose.pose.position.x
      y = msg.pose.pose.position.y
}
```

    - Main block and ROS initialization functions.

```
int main(int argc, char** argv){
      ros::init(argc, argv, "map_navigation_node");
      ros::NodeHandle n;
      // Robot pose object + subscriber
      RobotPose robotPose(0,0,0);
```

- Declaration of the subscriber amclSub that is used to return the global pose of the TurtleBot.

```
ros::Subscriber amclSub = n.subscribe("/amcl_pose", 1, &RobotPose::poseCallback,
                                      &robotPose);
```

- Robot Navigation
  - Initialization of the vector containing the object coordinates (x, y *and* φ), and the vector containing the image tags that the TurtleBot is searching for in the environment. If the function is not able to properly initialize the variables it will terminate the program and state what the problem was.

```
// Initialize box coordinates and templates
Boxes boxes;
if(!boxes.load_coords() || !boxes.load_templates()) {
      std::cout << "ERROR: could not load coords or templates" << std::endl;
      return -1;
}
```

"`boxes.coords`" is a two-dimensional vector, where the first index of the vector is used to specify which coordinate you would like to choose and the second index is used to select whether you would like to access the x location, y location or orientation of that coordinate, stored in that order. For example typing `boxes.coords[1][2]` would return the orientation of the second coordinate stored in the vector (vectors are zero-indexed data structures).

"`boxes.templates`" is a one-dimensional vector, containing the templates from the boxes_database.

- Sensor Implementation – RGB camera
  - The following block defines a class that is used to return an image from an image topic provided by the Kinect sensor.

```
ImagePipeline imagePipeline(n);
```

Code Breakdown – navigation.cpp

- The `Navigation::`*moveToGoal()* function is used to move the TurtleBot to a goal location. When destination coordinates (x, y and φ) are passed to the function, it allows the TurtleBot to navigate to the commanded location while performing obstacle avoidance. For more information on this function please refer to reference material in A.1.3 in the appendix.

```
bool Navigation::moveToGoal(float xGoal, float yGoal, float phiGoal){
```

Code Breakdown – imagePipeline.cpp

- The *getTemplateID()* function takes in object of class type *Boxes* containing the image templates *Boxes.templates*. The algorithm should search video feed that is being returned from the image callback for the matching image tags from the vector. If it finds a tag in the video, it should return the index of the found tag to the main function.

```
int getTemplateID(Boxes &boxes){
      int template_id = -1;
```

- The below checks to see if an image has properly been set to the video variable and then displays it to the screen before releasing the memory and returning an integer.

```
if(!isValid){
      std::cout << "ERROR: INVALID IMAGE!" << std::endl;
} else if(img.empty() || img.rows <= 0 || img.cols <= 0) {
      std::cout << "ERROR: VALID IMAGE, BUT STILL A PROBLEM EXISTS!" << std::endl;
      std::cout << "img.empty():" << img.empty() << std::endl;
      std::cout << "img.rows:" << img.rows << std::endl;
      std::cout << "img.cols:" << img.cols << std::endl;
} else {
      /***YOUR CODE HERE***/
      // Use: boxes.templates
      cv::imshow("view", img);
      cv::waitKey(10);
  }
  return template_id;
}
```

- Refer to the coords.xml file located in mie443_contest2/src/boxes_database folder. This file is used to store the coordinate locations that the robot will be traversing. During testing, these locations can be altered to test for robustness but **do not** make changes to the structure of the code for the contest here as a new coords.xml, contest2_env.pgm and contest2_env.yaml will be provided on contest day.
- Refer to the templates.xml file located in mie443_contest2/src/boxes_database folder. This file is used to store the image tag names. The tags can be found in the same folder.

Code Breakdown – coords.xml

- XML is a coding syntax that allows data to be ordered and stored in a very efficient structure for retrieval in code. Also, because it is XML and not C++ code, it does not need to be complied every time you make a change.
  - The headings in the following block of code contain the x, y and φ information of each coordinate the robot will navigate to with respect to the world coordinate frame of the map. The data in each heading is listed as x, y, φ in descending order.

```
	<coordinate1>
	1.857  <!—x -->
	0.153  <!—y -->
	0.9873 <!—z -->
	</coordinate1>

	<coordinate2>
	1.9744
	0.4719
	-0.1584
	</coordinate2>

	<coordinate3>
	0.4078
	0.9061
	2.9841
	</coordinate3>

	<coordinate4>
	0.4458
	-0.069
	1.32333
	</coordinate4>

	<coordinate5>
	0.8928
	-0.234
	-1.932
	</coordinate5>
```

Code Breakdown – templates.xml

- The following initializes the folder path to load the images to be identified during the contest. The folder paths to the pictures are defined with respect to the folder path of the ROS package location on your computer.

```
<templates>
	"tag1.jpg"
	"tag2.jpg"
	"tag3.jpg"
</templates>
```

Compiling

- Every time the code is changed you must compile it from terminal in the catkin_ws directory using the following command. If you do not do this, a new executable will not be created when you run it.

```
catkin_make
```

Robot Simulation- To simulate the TurtleBot and Kinect information before implementing your code on the physical TurtleBot, the following steps should be followed:

- Move_base and OpenCV in Gazebo
  - Begin by launching the simulated world through the following commands (you can change the world number if other world files are available in ../worlds folder):

```
roslaunch mie443_contest2 turtlebot_world.launch world:=1
```

- To run the amcl localization and obstacle avoidance algorithm in simulation instead of on the physical TurtleBot use the following command (make sure the map number corresponds to the world number):

```
roslaunch turtlebot_gazebo amcl_demo.launch
map_file:=/home/turtlebot/catkin_ws/src/mie443_contest2/maps/map_1.yaml
```

- 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
```

- All other commands to view and save the map are the same as in the amcl tutorial.
  - Before running your code, make sure the information contained in ../boxes_database/coords.xml is for the map which you launched in the previous step. If not, copy the correct information to ../boxes_database/coords.xml. For Gazebo, xml files corresponding to the maps are located in ../maps folder. Run the following command to start running your code in simulation:

```
rosrun mie443_contest2 contest2
```

- OpenCV on TurtleBot Laptop Webcam
  - To test your vision algorithm, you can use the laptop webcam. The webcam will work to create a proof-of-concept algorithm before it is transferred and finalized on the TurtleBot.

  - In order to publish the webcam images, you must first create the following lines of code in the imagePipeline.cpp file and recompile by running catkin_make:

```
#define IMAGE_TOPIC "camera/image" //Kinect:"camera/rgb/image_raw" webcam:"camera/image"
```

  - Next run the following node in a separate terminal (make sure contest2 node is running before you run the following command):

```
rosrun mie443_contest2 webcam_publisher 0
```

- This will allow your program to subscribe to the webcam images instead of the Kinect images. To change back to the Kinect, close the webcam node terminal, switch the commenting back to the original code and recompile.

Robot Execution - When you are ready to run your program on the TurtleBot, the following steps should be followed:

- If your team chooses to test in a different area than the contest environment, then the default map **will not work** and a new map must be made for the robot. To do this, please complete the following steps:
  - Follow the mapping tutorial in the reference material section A.1.2 in the appendix.
  - Copy the resulting my_map.yaml, and my_map.png files to the maps folder in your code package at the address: ../mie443_contest2/maps.
  - Change test destination coordinates in your coords.xml file to reflect the changes in the map.
  - Within the mie443_contest2_package you have been provided with a pdf of the tags being used for evaluation, feel free to print these off in order to make your own objects for testing purposes.
- Connect the laptop to the two USB ports on the TurtleBot. One USB port is for the Kobuki base, and the other USB port is for the Kinect sensor.
- With the Kobuki base and laptop turned on, run the following commands in their own terminals:

  - This starts the TurtleBot Kobuki base and starts publishing topics.

```
roslaunch turtlebot_bringup minimal.launch
```

  - This starts the Kinect sensor, loads the proper map and determines the robot's location in the map.

```
roslaunch turtlebot_navigation amcl_demo.launch map_file:=/<map_path>/my_map.yaml
```

  - This command opens a plugin that visualizes the data that the robot is publishing and subscribing to. It also allows the user to troubleshoot the navigation algorithm by providing an interface to publish navigation goals for the TurtleBot to pursue.

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

- Before running your code, make sure the information contained in ../boxes_database /coords.xml is for the map which you launched above. If not, copy the correct information to ../boxes_database /coords.xml. To execute your code, run the following line in its own terminal:

```
rosrun mie443_contest2 contest2
```

- If all is working correctly, your robot will begin navigating through the environment to find the objects at their locations, while identifying the image tags on each object.
- After all objects have been examined and the TurtleBot has returned to its starting location it ****MUST**** output what tags have been found as well as their respective coordinate index for evaluation.
- Cancel all processes by pressing Ctrl-C in their respective terminals when you are done.

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 of use for Contest 2:

- A.1.1
- A.1.2
- A.1.3
- A.2.1
- A.2.2
- A.2.4