Box-Pushing & Parking with the ROS Navigation Stack

Jordan Lane and Brad Ofrim

1. PURPOSE

The purpose of this competition is to develop a strategy to score the most points possible on a course given a time limit. A finite state machine was created and used to allow a TurtleBot robot to complete various tasks and earn points.

Video of the tasks described here (aside from box pushing) can be found at this link: https://drive.google.com/drive/u/1/folders/1QauypVfodiY3RVKXmjgKpZewQg1Gy8Nf

2. LAYOUT AND SCORING

The layout of the course consists of a white line, four stop lines, and three locations. The task at each location is optional, however, the Turtlebot will earn no points for any skipped locations. To complete the course, the Turtlebot must do a full loop of the course both starting and ending at the start line.

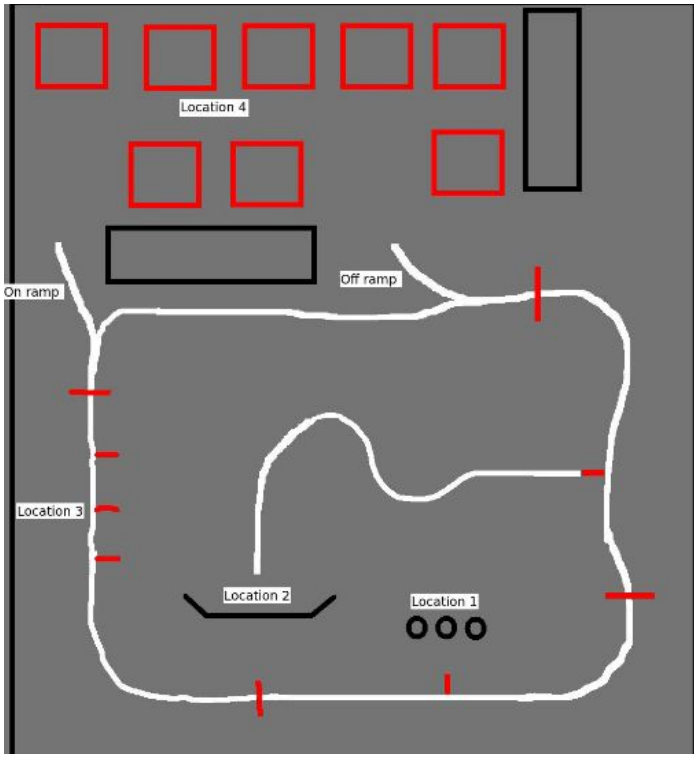

Figure 1: The Race Course

2.1 Location 1

Marked by a short red line, Location 1 is the first task on the course that the robot can complete. The first location is an object counting task, where the Turtlebot must count the number of objects 90° counterclockwise to the track. After counting, the Turtlebot must convey its results to the user with its onboard led lights and speakers.

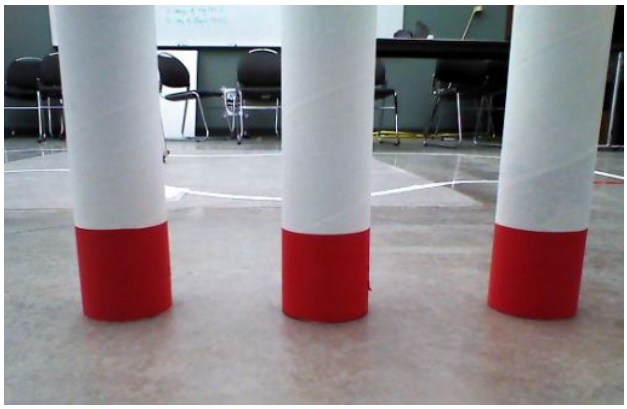

Figure 2: Objects for Location 1

2.2 Location 2

Location 2 is at the end of a small detour marked by another short red line. To get to Location 2, the Turtlebot must make a 90° turn at the small red-line, and follow a white line that branches off the original track.

Figure 2: Detour to Location 2

After following the detour line, the Turtlebot arrives at Location 2; a white display board with up to two red shapes and one green shape. Similar to Location 1, the Turtlebot must count the number of shapes and output the result to the user. Also, the Turtlebot must remember the shape of the green object if it plans on performing the task at Location 3. After completing the second task, the Turtlebot must make a 180° turn and follow the white line to return to the original track.

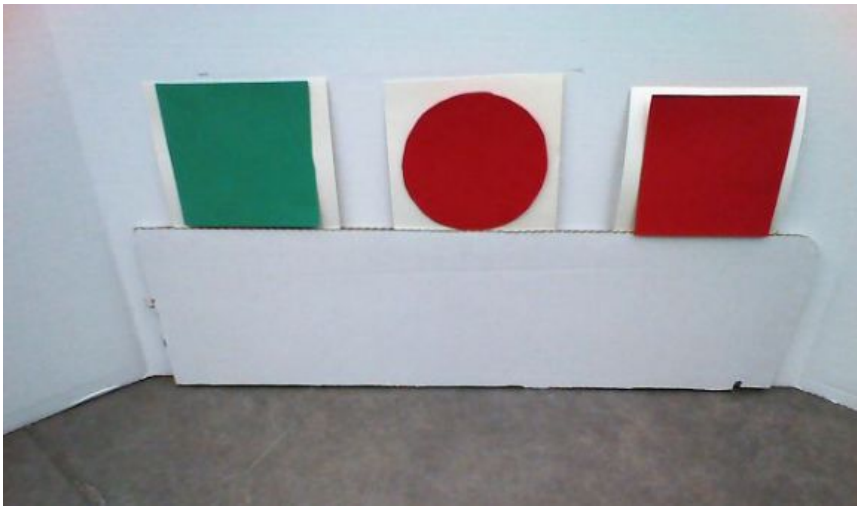

Figure 3: Location 2 Task

2.3 Location 4

Location 4 is located at the end of the white line off ramp. Unlike the other locations, Location 4 is an open area with no white line to guide the robot. The location consists of eight parking spots, which are red squares numbered one through eight.

The first task that the robot must perform once it has arrived at location four is to locate specific objects in the open area.

Figure 4: Location 4

An AR tag propped up on a small tripod is located in one of the squared marked 1 through 5. The location of this tag denotes the target parking location that will be used during the box pushing

component of this competition. The robot must be able to locate this AR tag and indicate that it has found it to earn points. An indication that the AR tag has been located consists of making a sound and turning a green LED on.

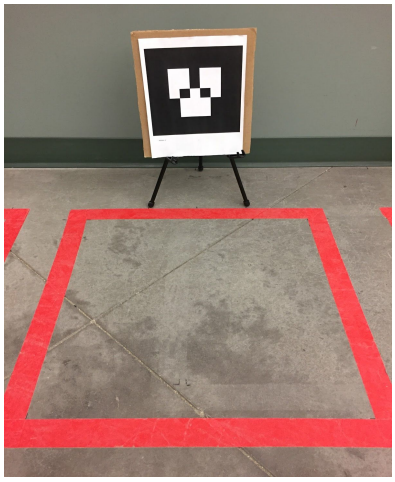

Figure 5: AR Tag Parking Spot

A box can also be found somewhere in location 4. Specifically, it will be found in one of the squares labeled 2, 3, or 4. This box has AR tags on all 5 visible sides. The robot must be able to locate the box. Once it locates the box, it must indicate that it has found it. To indicate this, the robot must turn a red LED on and make a sound.

After the robot has successfully located the target parking stall and the box, the second task in location 4 is box pushing. The robot must attempt to push the box from its starting location, into the parking stall that target AR tag is next to. Once the robot has pushed the box into the target stall, it must indicate that it has completed the box pushing portion of the course. To do this, it shall make a sound and turn it's LEDs red and green.

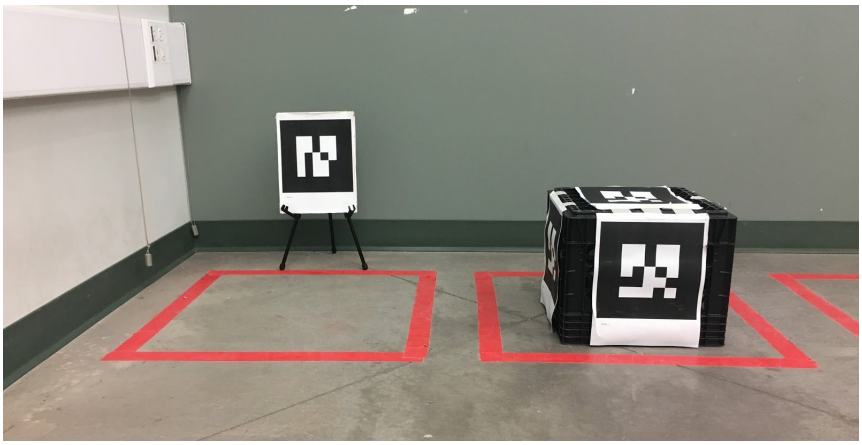

Figure 6: Box Pushing

The third task of Location 4 is an object recognition task that requires the Turtlebot to park at a parking spot with the same shape identified at Location 2. Once the robot has successfully found the parking shape containing this shape it must indicate this by making a sound and turning on an orange LED. After this indication the robot must move into the parking stall and indicate that it has parked. To do indicate this, it must make a sound again, as well as turn LEDs orange and green.

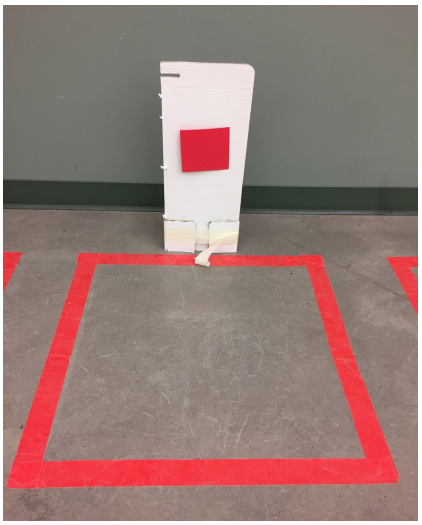

Figure 7: Shape Parking Spot

Points for Location 4 are given based on the accuracy demonstrated while parking in the correct spot, the successfulness of locating the target and the box, and the accuracy of the box push. After completing the tasks for Location 4, the Turtlebot must navigate to the on-ramp and continue the course.

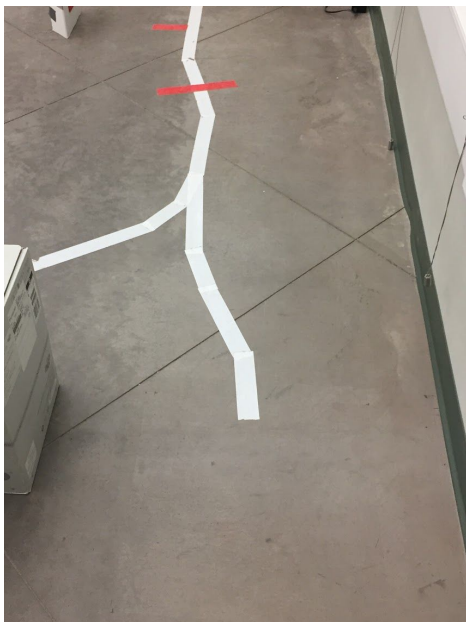

Figure 8: On Ramp

2.4 Location 3

The final location of the track is another object recognition task. This final task requires the Turtlebot to have previously completed the task at Location 2. Location 3 consists of three small red lines, with a red shape placed 90° counterclockwise to each red line. To complete the task, the Turtlebot must scan the object at each red-line, and indicate if it is the same shape as the green object at Location 2.

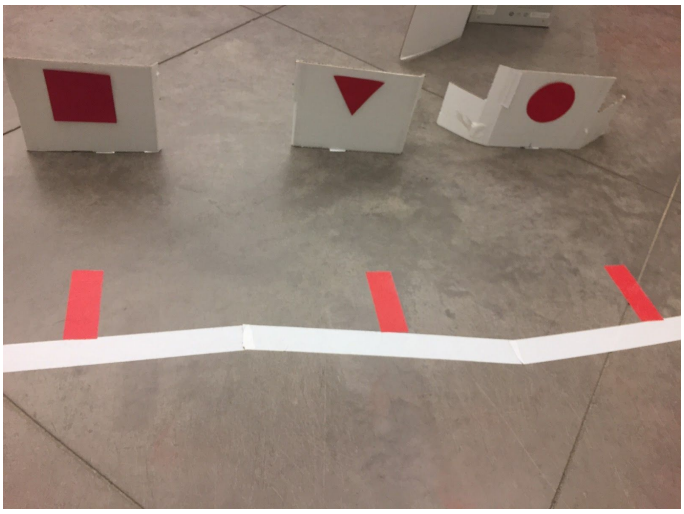

Figure 9: Location 3 Task

Once all tasks have been completed, the robot can start over if it still has time. After finishing the last task, the robot can follow the white line to arrive back at the start of the course.

2.5 Strategy

The point distribution for completing the tasks rewards more points for the most complicated tasks, and fewer for simpler tasks. However, some lower value tasks (such as object detection in location 2) are required in order to score points later on in the course (parking at the stall with the shape in location 4). We also determined that some of the lowest value tasks such as stopping at a red line could be completed quick enough that attempting to do these tasks would not jeopardize our ability to score points in the higher value tasks.

With this information, we decided to try to attempt all possible tasks. To make sure that we would be able to gain as many points as possible from the high value tasks, we increased the speed of our robot and reduced stalled time as much as possible. To account for this increase in speed, we spent time adding in control systems and tuning parameters to ensure that the Turtlebot stayed on the course at all times.

3. SETUP & PREREQUISITES

3.1 Setting Up Your Robot

In order to complete this competition, you will need to get the following hardware:

- Turtlebot 2
- Asus Xtion Pro RGB-D Camera
- USB Camera
- Logitech Controller
- Laptop with 4 USB 2.0/3.0 Ports. (Warning: a USB hub may be used however we have found restrictions when using both the RGB-D camera and USB-cam on the same hub)
- A foam bumper
- A small block that is a few centimeters tall

Once you have acquired the hardware, you will need to configure your robot so that it will be able to operate properly with our code. Here are the steps for building your robot:

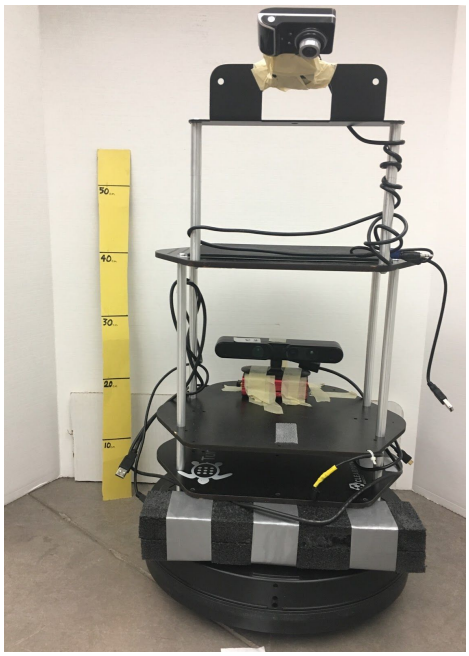

Figure 10: The Properly Configured Robot

1. Attach four short rods into the base and mount the first platform on top. Ensure that the locations of the four short rods will align with the holes drilled into the platforms. This platform will be used to hold the foam bumper.
2. A second platform will be needed to mount the Asus Xtion Pro RGB-D Camera. Secure the first platform in place by inserting four more short rods on top of the rods you inserted in step 1. Mount the second platform on top of these rods.
3. Wedge the foam bumper between the robot base and the first platform. The bumper should cover as much of the robot's bumper sensor as possible. The bumper will be

turned off for the competition, however we want to avoid any error created by the curved bumper during box pushing.

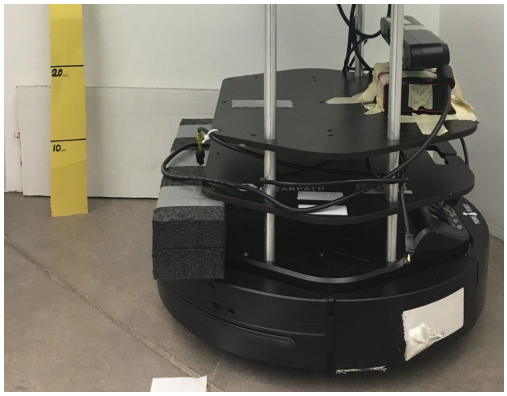

Figure 11: A Side view of the robot showing its lower platforms and foam bumper

4. Mount the third platform used to hold the control laptop. To do this, we will secure the second platform in place by attaching the four long rods on top of the short rods that the second platform rests on.
5. Fix the third platform in place by inserting screws through the platform and the top of the long rods.
6. Mount the USB camera by attaching two medium length rods near the front of the third platform and bridge them with the metal camera bracket.

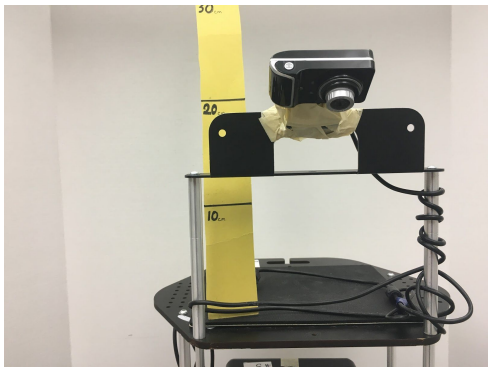

Figure 12: The upper platform with USB camera attached.

7. To affix the Asus Xtion Pro RGB-D Camera, fix a block near the rear of the second platform. Attach the camera onto the top of this block. Angle the camera down by exactly -4 degrees (using a phone with a digital level is quite handy when it comes to aligning this camera). It is recommended to fix this camera angle somehow in order to avoid difficult

bugs later on (for example if the camera is angled slightly up, it will have difficulty seeing the box with a laser scan).

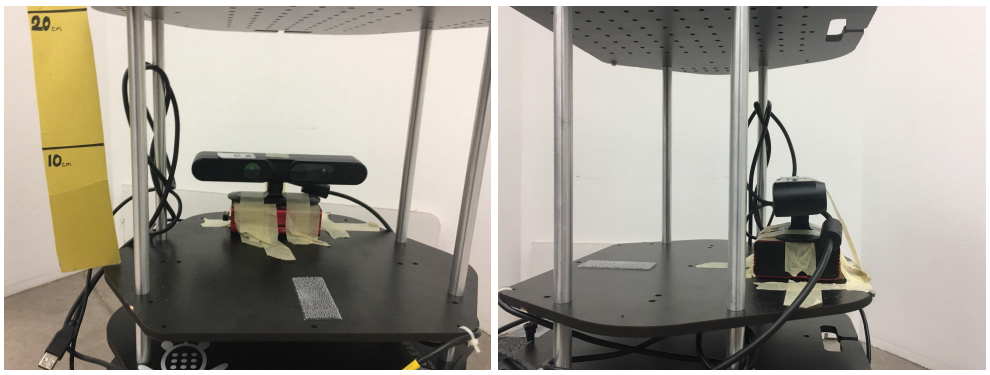

Figure: Front and side view of the Asus Xtion Pro RGB-D camera.

8. Lastly, attach the usb camera to the top of the cross beam that was attached to the very top of the robot. Angle the camera down such that the bottom of the cameras view can almost see the front of the robot base. We found that this camera does not have to be aligned as precisely as the Asus Xtion Pro RGB-D Camera, and an angle of approximately 50-70 degrees downward should work.

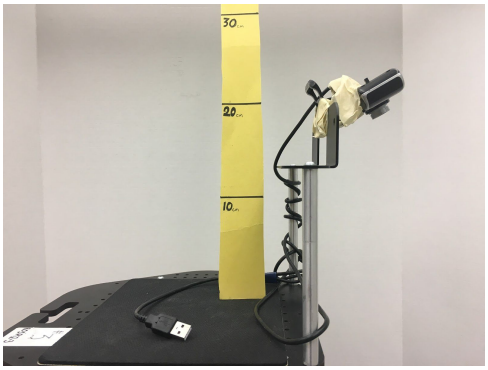

Figure: Alignment angle for the USB camera

3.2 Prerequisites

This project is built using python 2.7, ros-kinetic, ros-kinetic kobuki, and turtlebot libraries for Ubuntu Xenial 16.04. If these are not installed, please refer to their installation pages on the official ROS wiki or the official python installation page.

- http://wiki.ros.org/kobuki/Tutorials/Installation/kinetic
- http://wiki.ros.org/kinetic/Installation
- https://www.python.org/downloads/

Other ROS libraries required for this competition include:

- http://wiki.ros.org/amcl
- http://wiki.ros.org/usb_cam
- http://wiki.ros.org/ar_track_alvar

The source code for our project can be found at https://github.com/bofrim/CMPUT_412

Create or navigate to an existing catkin workspace and clone our repository. Our repository consists of multiple catkin packages so you will need to copy the files (or just the packages you want; in this case, “comp4”) from our repository into your catkin workspace.

The repo README also details important information regarding CMPUT 412, ROS, and this competition.

4. EXECUTION

Open a console and source the catkin workspace that the package was placed. For example, if your workspace was called catkin_ws:

```
$ cd catkin_ws/
$ source ./devel/setup.bash
```

Next, build the project:

```
$ catkin_make
```

4.1 Turtlebot Racer

Before starting the Turtlebot, ensure that it is placed on the track behind the red start line.

To begin running the Turtlebot we start by running the general comp4.launch file. The comp4 launch file turns on the core components of our rover including the Turtlebot, RGB-D Camera, USB Cam, and navigation stack.

```
$ cd catkin_ws/src
$ roslaunch comp4 comp4.launch [video_device:=] [map_path:=]
```

Parameters:

- video_device: File path of the usb camera that will be used for white and red line navigation. (Eg: /dev/video1)
- map_path: (Optional) Filepath of a specific map. By default the launch file will use the map file located in {comp4}/data/map.yaml

After running the core launch file, we can start the competition state machine. Launching the ultra_sm launch file will start the Turtlebot for competition 5.

```
$ roslaunch comp4 ultra_sm.launch [initial_line:=]
```

Parameters:

- initial_line: (Optional) This parameter specifies how many red-lines the Turtlebot has already reached. This allows for more verbose testing and flexibility with our starting point. The parameter is fully optional and takes a default value of 0.

5. CONCEPTS AND CODE

The Turtlebot racer consists of a state machine, an incoming data path, and an outgoing data path. The state machine consists of general driving nodes, a red-line decision node, and specific nodes for each location task. These nodes are assisted by image recognition nodes that are used to detect the white and red line centroids, as well as general nodes that control the leds, sounds, and motors of the Turtlebot.

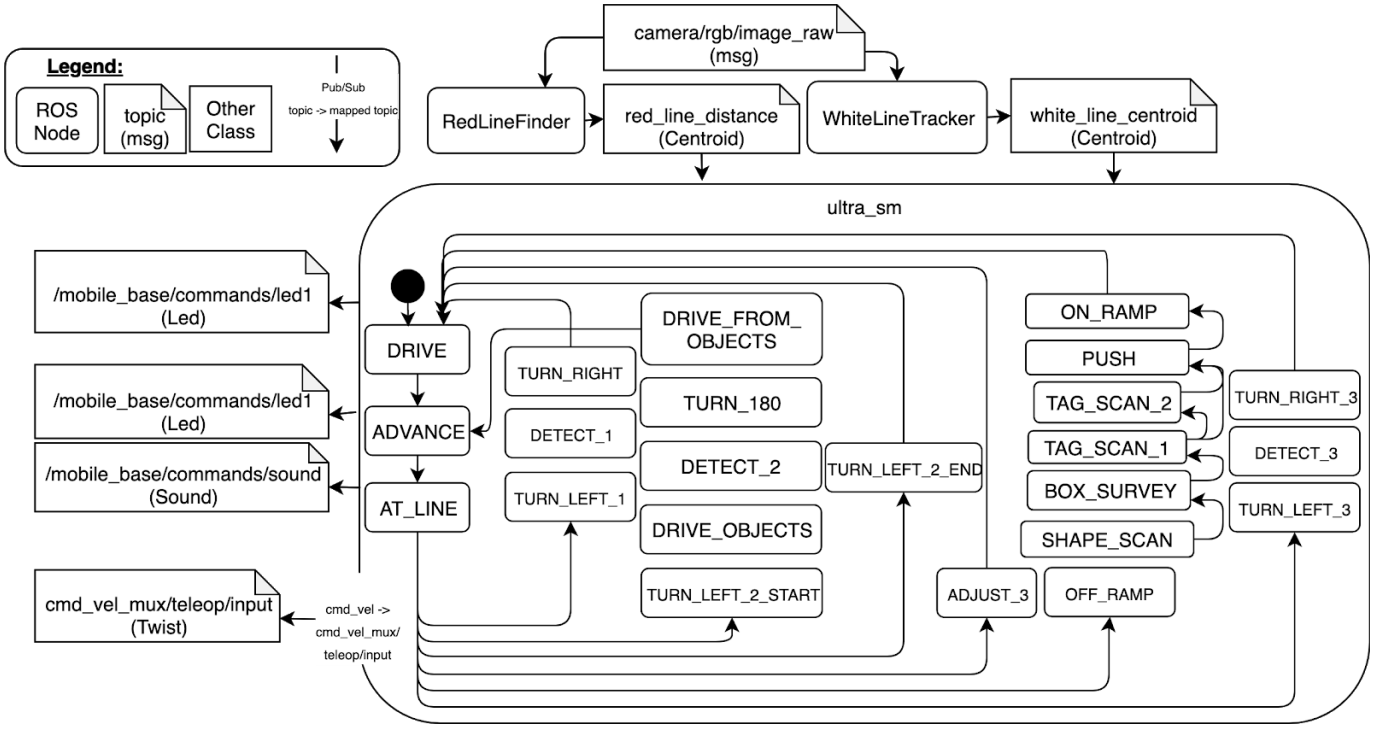

Figure 10: UltraSM Nodes

5.1 Course Navigation States

The general driving states consist of the Drive and Advance nodes. These states subscribe to topics published by the RedLineFinder and WhiteLineFinder nodes. This data is used to follow the white line and advance onto the various red lines of the course.

At each red line, the Turtlebot enters the key decision making state: the AtLine node. The AtLine state keeps track of how many red lines have been encountered, and decides if the Turtlebot needs to stop, or perform a specific task. Once the robot reaches the end of the loop the state machine will start over from the beginning allowing the robot to do continuous loops.

Our state machine contains all the states for all locations, but the specific logic for each location can be found in a location specific file. For example if you are interested in the location 4 code, you should take a look at Location4.py.

5.2 Location 1 States

Location 1 consists of three states: TurnLeft1, Detect1, and TurnRight1. The TurnLeft1 and TurnRight1 states are used to turn the robot towards the objects and back to the track.

Detect1 uses OpenCV to detect the number of red objects within the image and output the result using the Turtlebot's LEDs and speaker.

5.3 Location 2 States

Location 2 consists of six unique states: TurnLeft2Start, DriveObjects, Detect2, Turn180, DriveFromObjects, and TurnLeft2End.

The detour of Location 2 is unique from the rest of the track due to the white line ending with no red line. The detour line was also the most common location for the Turtlebot to lose the line due to glare or other various factors. Therefore Location2 has two unique drive functions: DriveObjects, and DriveFromObjects. These two drive states are similar to the general drive states, however, DriveObjects will follow the white line until it detects multiple shapes within the camera feed. DriveFromObjects is similar to the general drive state, however, it corrects for the vertical red line and corrects the Turtlebot if it loses the white line.

TurnLeft2Start, Turn180, and TurnLeft2End are generic turning nodes that are used to enter, leave, and navigate Location 2's detour.

Detect2 is the main image detection state used to detect the various shapes at Location 2. When the Turtlebot arrives at the whiteboard, it processes images to count the total number of shapes and attempt to identify the green shape. In order to detect the green shape with reasonable confidence, the system processes images until it repeatedly detects the same shape for a few consecutive images and records this shape for later. If it does not settle on a decision

after a maximum allowed number of images were processed, it chooses the shape that it thought it saw the most and records this shape for later.

5.4 Location 4 States

Location 4 states include DriverRamp, ShapeScan, BoxSurvey, TagScan1, TagScan2, and Push. The DriverRamp state will allow the robot to smoothly navigate the off ramp. The ShapeScan state will analyze parking stalls 6, 7, and 8 to try to find the matching shape, park, and indicate that it has parked in the correct stall. In the BoxSurvey state, the robot will drive to a vantage point that will allow it to see the box and it will measure the position of the box. The TagScan1 and TagScan2 states are similar; the robot will attempt to find the AR tag target marker by looking at the potential tag locations. TagScan1 will check stalls 4 and 5, TagScan2 will check stalls 1, 2, and 3. In the Push state the Turtlebot aligns itself with the box based on the placement of the box and target ARTag. Once the robot is aligned, it will begin pushing the box until it reaches the target destination. After completing the box pushing task the Turtlebot will navigate to the On-Ramp and continue the course.

5.5 Location 3 States

Location 3 states include the TurnLeft3, Detect3, and TurnRight3. The TurnLeft3 and TurnRight3 states are used to turn towards each red shape and back to the white line of the course.

Detect3 uses OpenCV to detect the shape and compare it to the shape saved during the Location 2 task. The robot displays the results by using green LEDs for a successful match, or red LEDs for a failed match.

5.6 Image Processing

The navigation and tasks performed by the turtlebot rely on image processing to operate. Here are a few high-level pseudo code samples showcasing some of the more important aspects of the image processing:

5.6.1 Color masking

```
Given: An HSV image, maximum and minimum HSV thresholds, optional noise
removal size, optional noise filter fill size

    Mask the input image, removing pixels below the minimum threshold
    Mask the input image, removing pixels above the maximum threshold
    Combine the maximum and minimum masks
    Apply an openCV morphology open filter to the mask with the removal
```

```
size
    Apply an openCV morphology close filter to the mask with the fill size
    Return: the thresholded and filtered image
```

5.6.2 Finding the centroid of the closest red line

```
Given: A masked image

    Find all contours in the image
    Pick the contour with a mass above a certain threshold whose center is
the lowest on the screen
    Return: the centroid of that contour
```

5.6.3 Detecting shapes

```
Given: A masked image

    Get contours of all shapes in the mask with masses above a threshold
    Create an empty array to hold discovered shapes
    Approximate each contour as a polygon
    Count the Number of sides the shape has
    If it has 3 sides
        It is a triangle, append a triangle to the array of shapes
    If it has 4 sides
        It is a square, append a square to the array of shapes
    If it has 5 sides
        It is a pentagon, append a pentagon to the array of shapes
    If it has more than 6 sides
        It was probably a circle, append circle to the array of shapes
    If it has any other number of sides, ignore it.
    Return: The array of discovered shapes
```

5.6.4 Studying a scene to confidently detect an image

```
Given: A function for getting a masked image

    Initialize a map of shape -> counts
    Store triangle, square, and circle in the map
    For at most a predetermined maximum number of iterations
        Get an image using the given function
        Detect the shape in that mask (see 5.5.3)
        Add one to the count of the detected shape in the map
        If we have iterated more than a predetermined minimum of iterations
            Compute: count/number of iterations for all shapes in the map
            If the computed value is larger than a confidence factor
                Return the detected shape
```

```
    Since no shape was detected yet, return unknown
```

5.6.5 Counting Objects

```
Given: Mask of an image

    Find all the contours in the mask, return the number of contours that
are bigger than a certain size
```

6. Additional Information

If you set up a course that is not the exact same as the course we ran our robot on, you will need to update a few things. Here are a few things you should update:

1. Turn Angles

   Our robot turns based on hard coded values that were determined based on experimentation. Any “Turn” based state in the state machine will might need to be updated for your course. Try adjusting with the integer value passed into the Turn state constructor to tune your robot for your course. Positive values will turn the robot left, negative values will turn it right.

2. Map & Waypoints

   If your location 4 is different than ours you will need to remap it and regenerate the waypoints. To do this, follow the mapping tutorial found here. Save your map file and map metadata file to the CMPUT_412/comp4/data/ directory. If you make a new map, you will need to regenerate waypoints. To regenerate the waypoints, run the waypoint_recorder.launch launch file and waypoint_finder.py node included in our project. Use the Logitech controller to drive your robot to each waypoint that is listed in the config_globals.py file. When at a each waypoint, click the X button on your Logitech controller. When you are done, click the “B” button to finish. The waypoints will be printed to the screen and can also be found in a file that was saved to your disk.