Mini Hack 1 - Writeup

Adolfo Roquero, Leticia Schettino, Kobi Green

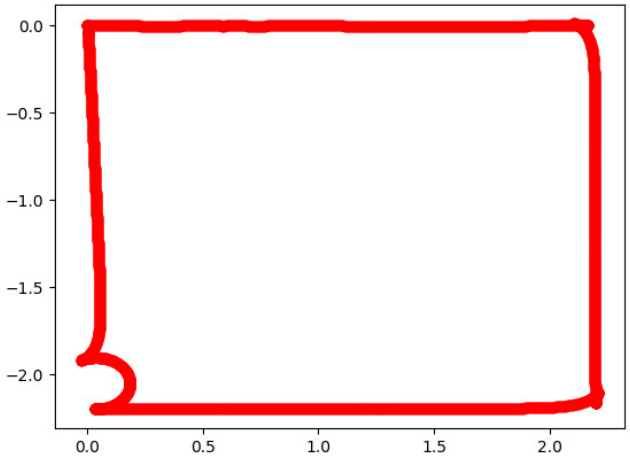
February 2022

1 Task 1

The code we have for Task 1 was based on the skeleton code for Mini Hack 1 as well as modifications taken from the 'turtlebot3_pointop_key.py'. One modification we made that is worth pointing out was that we changed the waypoint coordinates in the skeleton code to $([2.2, 0, 0], [2.2, -2.2, -90], [0, -2.2, -180], [0, 0, 90])$, so that we could start the square trajectory from the position we were starting from during the Mini Hack day. Once we had the subscriber to the position topic implemented and the while loop that controlled linear movement correctly modified, we used the following command to run our code for task 1:

```
$ python mini_hack_1.py --position_topic /odom --velocity_topic
    /cmd_vel --scan_topic /scan
```

Figure 1: Plot showing robot's trajectory using on-board odometer



As the figure above shows, the robot successfully completed the square trajectory autonomously, by moving towards each of the waypoints. In our first try, the robot was completing the square however, it was performing unnecessary rotations at each of the corners. We managed to fix some of these rotations however, we were unable to correct the rotation in the third corner, where we can see the robot turns 270^{o} instead of turning 90^{o}. A more detailed discussion of this is included in the 'Final Plot' section of this pdf.

2 Task 2

To complete Task 2, there was no need for us to modify the code we used for Task 1, we simply ran the code using the following command instead:

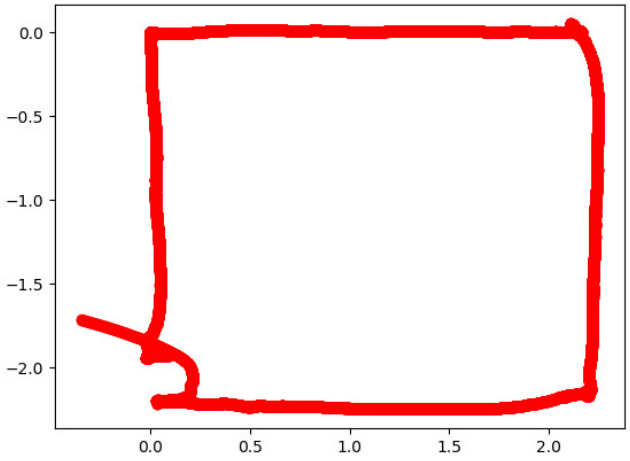
```
$ python mini_hack_1.py --position_topic /camera/odom/sample
    --velocity_topic /cmd_vel --scan_topic /scan
```

At first, we had a lot of difficulty getting the robot to successfully traverse the square using the camera topic for position sensoring. In trying to understand what was causing the error, we included print statements for the robot's x and y coordinates at each iteration of the while loop that controls the robot movement. In doing so, we realized that even though the robot was moving towards its first waypoint coordinate, the x, y coordinate of the robot was not changing (it continued perceiving as if it was still around $(0, 0)$ with very small fluctuations likely due to noise). Then, once the robot got really close to the wall the coordinate suddenly jumped to approximately $(60, 0)$, which was clearly incorrect and indicated the robot was not being able to correctly perceive its position from the camera odometer only. Following recommendations from the TF, we lined the walls of the testbed with bricks at the height of the camera to see if this would increase the visual stimulus. While this helped slightly, the coordinates were still really off and the robot was not being able to traverse the square correctly. At that point, we decided to switch wafflebots in case it was a hardware issue with the camera sensor in our wafflebot.

After switching robots we indeed saw a huge improvement in the coordinates the robot was estimating at each step and it was able to successfully traverse the square in the test bed. We have included below the plot of coordinates for task 2.

We can see from the plot above that while the robot is able to successfully traverse the square, there is more variation in the coordinate estimates and the square does not look as clean due to more noise in the data collected by the visual odometer compared to the on-board odometer. From the video recorded for task 2, we can see that the robot trajectory is actually almost as clean as the trajectory for task 1 and the spike in the plot (seen around coordinate $(-0.5, -.1.8)$ did not actually occur but rather is shown in the plot because of where the robot perceived its position to be given the noisy sensory data.

Figure 2: Plot showing robot's trajectory using T265 visual odometer

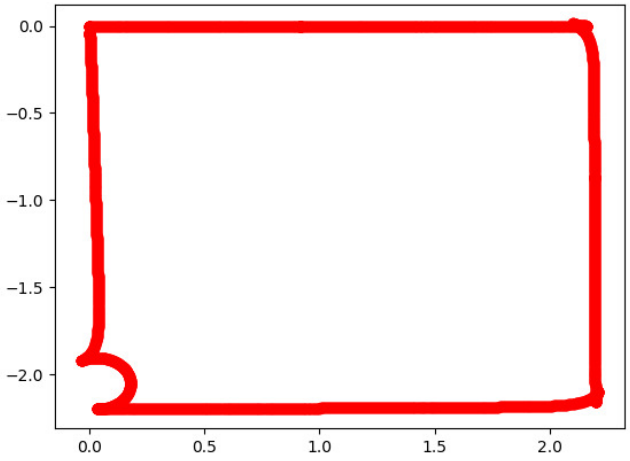


3 Task 3

The code used for task 1 and task 2 had to be modified to complete task 3 since. Since the robot is required to detect obstacles using the LIDAR sensor and stop its motion until the obstacle is removed, the two modification to the code from task 1/2 was to 1) implement the 'get_scan' function which collects data from the LIDAR and is based off the distribution code in 'turtlebot3_obstacle.py' and 2) modify the while loop that controls linear movement such that it had an if/else clause that verified whether the LIDAR was sensing obstacles in the specified minimum distance allowed. It is worth noting that we used a sample view for the LIDAR of 2^{o} (instead of the 180^{o} we experimented with during the MiniHack day) since the test bed contained objects in the center that we do not want to interfere with the trajectory. This means that the implementation we have for task 3 the robot will only stop for obstacles directly in front of the robot.

The plot above shows the trajectory of the robot when traversing the square using the on-board odometer and performing obstacle detection. Since in the presence of the obstacle the robot stops but it does not deviates from its path, the plot above seems very similar to the plot generated in task 1, which is the expected behavior. In the video and the bag file, we can clearly see the locations in which there was an obstacle detected and the robot's response of stopping in place until the obstacle has been removed.

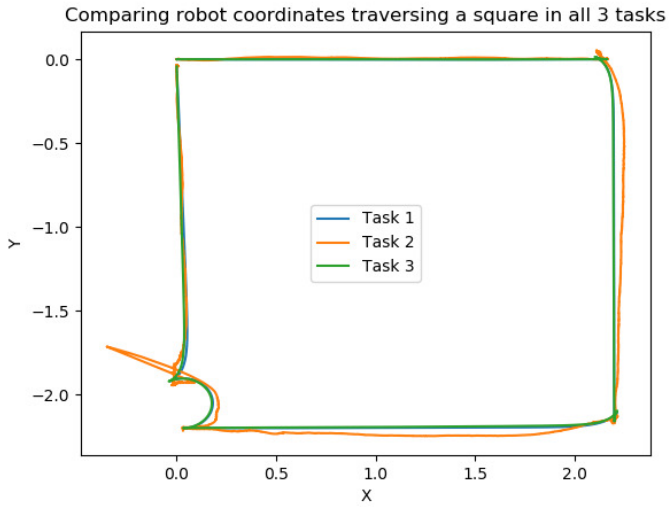
Figure 3: Plot showing robot's trajectory using on-board odometer and demonstrating obstacle avoidance



4 Final Plot

We have also included below the plot with the trajectories for all three tasks. We can see that in all 3 tasks the robot successfully traversed a square. However, in all of the cases we were unable to fix the rotation made by the robot in the second to last corner of the trajectory. Originally, the robot was performing a similar unnecessary rotation in the first and second corners as well but we were able to fix the first two rotations by changing the z coordinate of the waypoints to ensure that once the robot reached a waypoint, it would remain facing in the direction it was facing during its linear navigation. Unfortunately, for the final turn the robot has to change from facing -180º to 90º, which in the skeleton code results in a 270º turn instead of the shortest alternative of turning only 90º. This is due to the fact that only degree values between -180º and 180º are accepted and the accompanying logic in the code. Unfortunately, we couldn't test it with actual robots before the deadline due to the snow storm. But, we believe that if we removed the 'if' statement that caps the variable 'goal_z' to be on the [-180,180] range, we would be able to provide an angle of -270 in the last waypoint instead of 90 degrees. This way the robot would only need to make a -90 degrees turn from -180 to -270 instead of having to do a 270 degrees turn from -180 to 90 degrees.

Figure 4: Plot with trajectories for all tasks



5 Deliverables

Included in the zip file:

- Bag file with recordings for all sensors of robot completing Task 1, Task 2 and Task 3.
- Coordinate plots for each individual tasks as well as one plot overlaying all of the tasks.
- Video of the robot completing each of the tasks.
- mini_hack_1.py - Python file used to complete all 3 tasks, based on skeleton code.
- generate_plots.py - Python file used to read the bag files and generate the plot with coordinates for each task.