Computational Robotics: Neato Operations

Nick Steelman and Siena Okuno

September 21, 2018

1 Code Structure

Our code base is bisected into ROS interfacing programs and action controlling programs, as shown Figure 1. The program interface.py is entirely comprised of classes to get data from ROS to Python and vice versa, i.e. SendSpeed, SendMarker, ReceiveLidar, ReceiveBump, ReceiveAccel, and ReceiveOdom. Each class will initiate a subscriber or publisher (depending on the type) and have a function to return the last message or publish a new message respectively.

All other controllers will import interface.py and create instances of these classes to send their data. Some also import from teleop.py, which handles manual control of the Neato, in order to move the robot into place.

This structure enables greater code re-usability as multiple controllers can import the same classes, and easier code writing and debugging with only a couple types of libraries in use at a time.

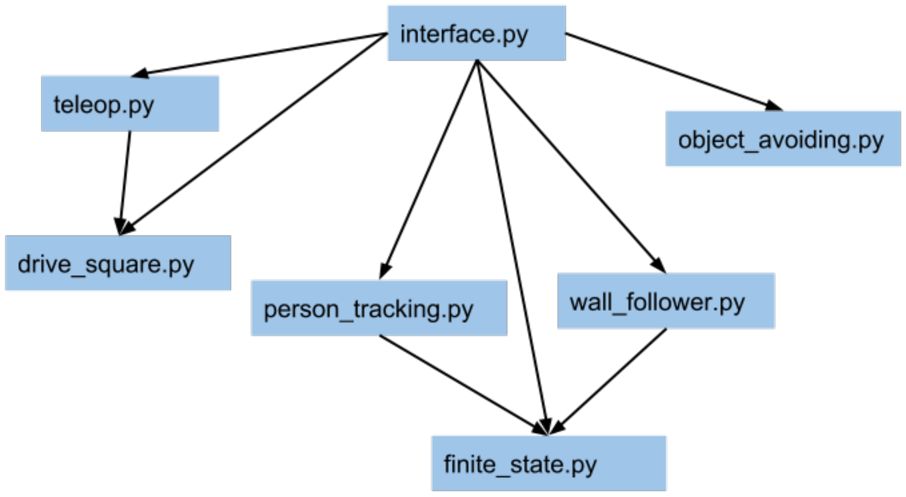

Figure 1: Map of interaction between files....I'm gonna make this cleaner

2 Teleop and Drive Square

2.1 High Level

The initial problem was building a control interface to maneuver the Neato. Given the function to read inputs from the keyboard, it was relatively easy to write a while loop to detect WASD keypresses and then send the corresponding signal to the Neato using interface.py.

The second and simplest problem is driving the Neato in a square. As we were still getting used to the Neato's topics and messages, we structured the code to fit the simplicity of the problem; we hard-coded it. Through trial and error, we were able to make the Neato drive in a relatively accurate 1m x 1m square. Follow this link[1] for a video of our drive square.

2.2 Potential Improvement

For the square script, an obvious improvement is to use the odometer on the Neato. This would allow us to accurately move set distances and turn at specified angles. Besides a better square as the outcome, it would also allow for the square to be adjusted should the desired size change.

3 Wall Follower

3.1 High Level

For the next problem, the goal was to have the Neato follow a wall at a specified distance. For this problem, we decided to use a robust method of wall detection by comparing observed data to an expectation of what the laser scan should look like for a wall. Given that the closest point in the laser scan is a candidate point for the wall, we can determine the likelihood of the point by computing how much the surrounding points form a plane tangent to the Neato. If we transform the plane into spherical coordinates we can compute the expected shape with the equation $\frac{d}{cos(\theta)}$, which is shown in Figure 2.

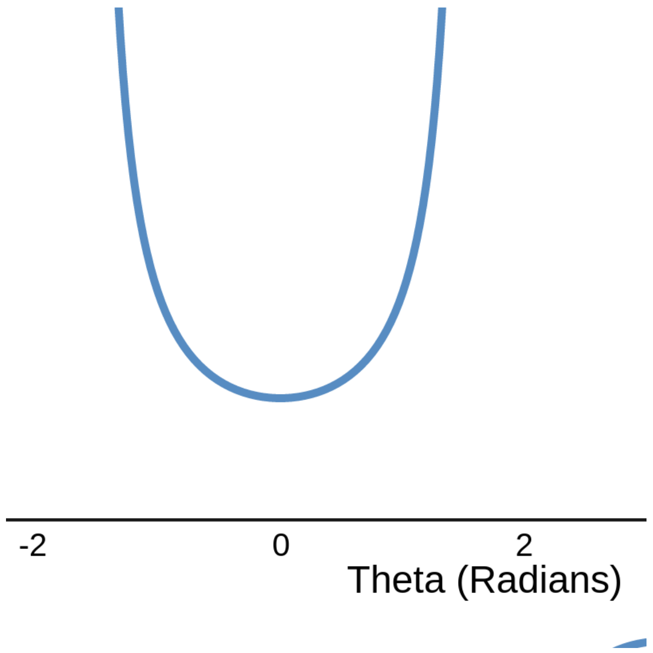

Figure 2: Graph of expected distance given degree

Rather than worry about approaching the correct distance and correct angle in conjunction, we decided to approach the problems discretely. So, if the Neato determines a point to be a satisfactory candidate for a wall but it is too far away or too close to the point, it will first correct the distance by heading in the appropriate direction, either turn towards or away from the wall (respectively). After reaching the correct distance it will align itself with the wall and travel along it, veering back on course should it get too close or far away. This can all be done with only the laser scan data and velocity control. Follow this link[2] for a video of our Neato following a wall.

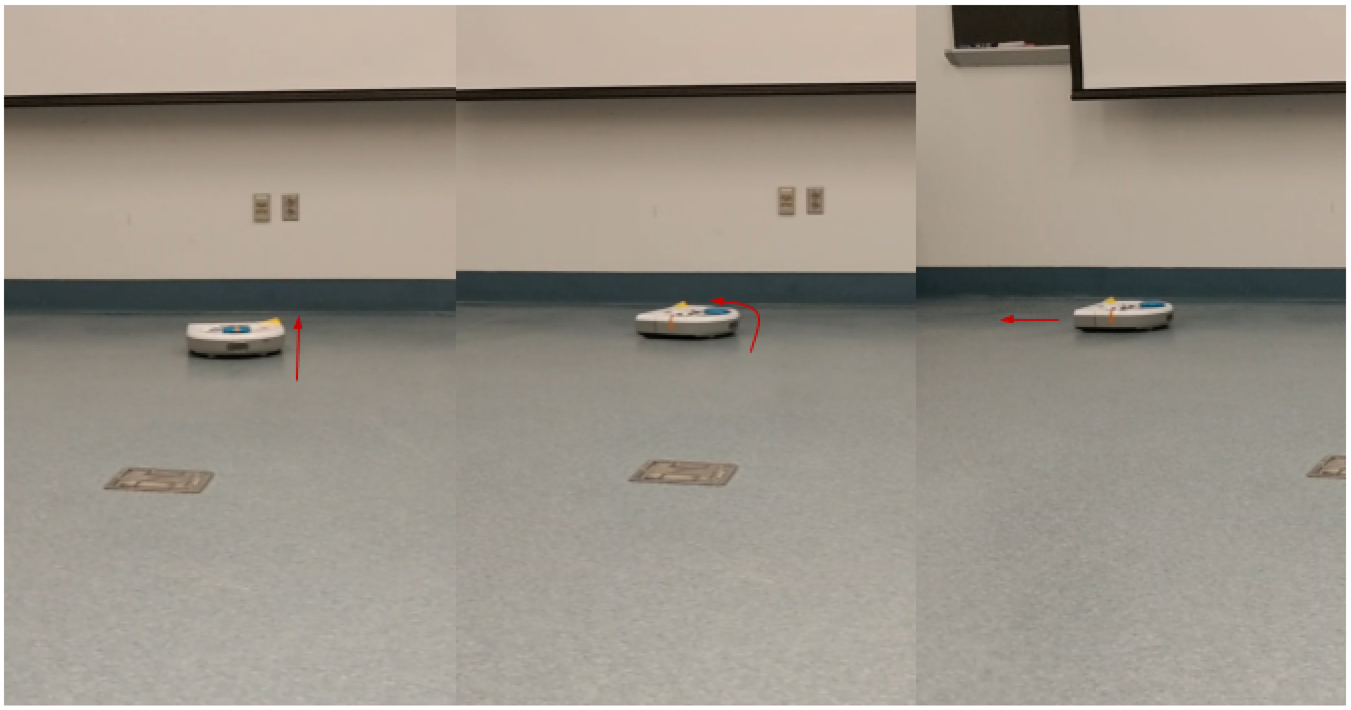

Figure 3: Images of our wall follower approaching the wall, turning, and following the wall

3.2 Challenges

The main challenge we faced was determining the appropriate logic around the Neato's approach of the wall given the angle of the wall and distance. In particular, the turning direction during the approaching and following states was a bit tricky. After substantial blackboard drawings, we were able to arrive at the correct numbers.

3.3 Potential Improvement

Potential improvements include programming the Neato to seamlessly and gradually transition between going to the wall to along the wall, adding safeguards for a noisy environment, and designing a method that is robust while being less computationally costly.

4 Person Tracking

4.1 High Level

For the person tracking problem, we had the Neato store the position of a person in its coordinate frame. From here, it filtered out noise and small variations and followed the average path of the person. Follow this link[3] for the final product.

4.2 Challenges

This problem broke down in a several interesting challenges.

1. The first interesting challenge was to locate a person, aka a patch of moving points. For this we examined the points immediately in front of the Neato, specifically within a 90 degree field of view from the front of the robot and a distance of 2m. We wanted to set up the Neato such that this area was not too dense with points, making movement in the environment easier to track. To do this, we took the average of the points in front of the Neato and looked for significant changes. When a change was detected, we used the midpoint as the center of the person and begin tracking from there.

2. Once we had the location of the individual we began tracking their location. With so many data points every second, it can hard to parse what was human leg and what was table leg. Therefore we decided on a mix of techniques to get the relevant points. First, we did not consider more than 8 points as that was more than enough for an accurate mean. Second, we did not consider points more than a meter away as we assumed that the person would not move such a large distance between readings. Finally, we looked at the differences between point differences and filtered out points with large jumps because these are likely other objects.

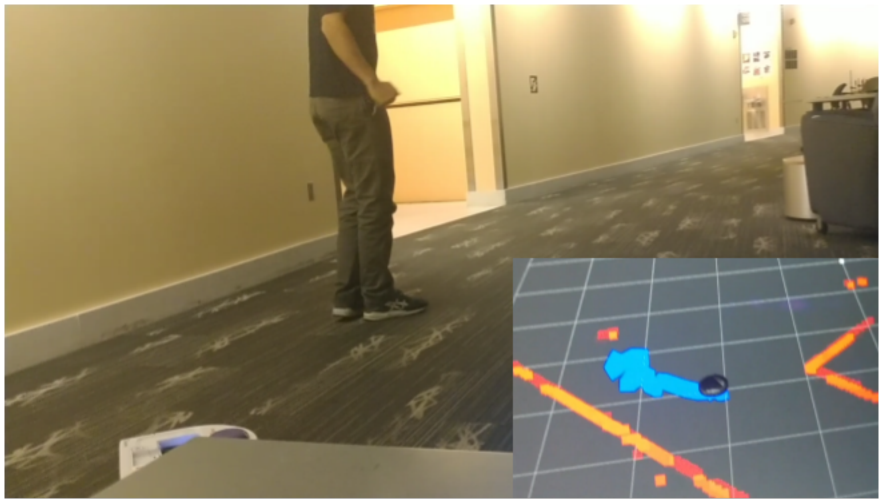

Figure 4: Comparison of reality and what the Neato sees

3. Finally, the data produced from averaging choppy laser scans of moving legs is rarely clean. The mean changes, jumping from leg to leg rather than following a smooth curve along the persons path. Our solution to this had 3 steps. First, we moved the Neato forward slowly on turns and modulated its speed proportionally to how much it had to turn, allowing it to turn more smoothly for linear points. Secondly, we added a threshold such that the Neato will ignore upcoming points it is already close to, choosing to navigate to the farther points and reducing any short, sharp movements. Lastly, we took groups of 8 points and averaged them, sacrificing some accuracy of the path tracing for overall speed and fluidity.

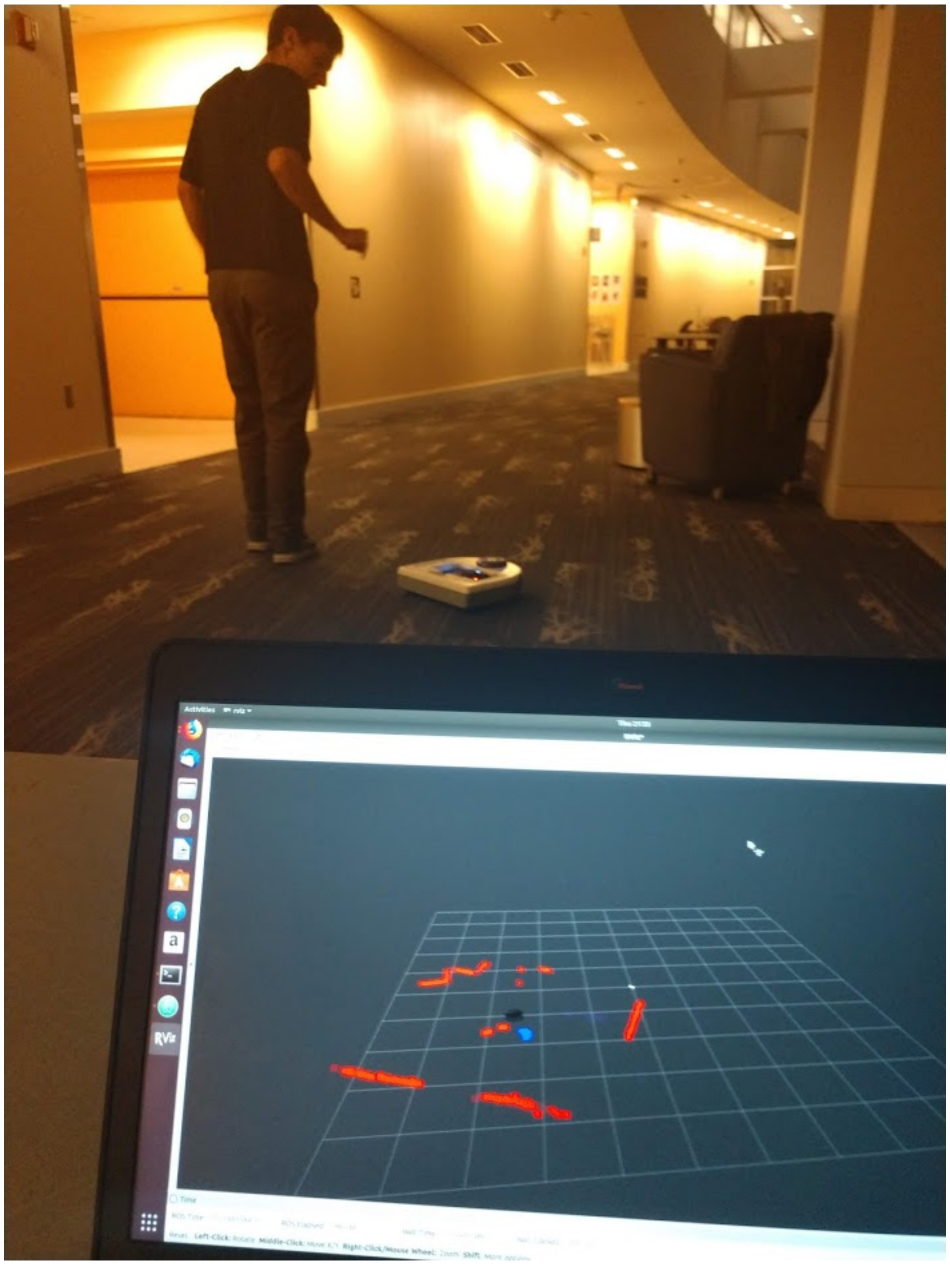

Figure 5: Image of our Neato in action

4.3 Potential Improvement

Despite our efforts, the Neato's actions are still choppy to an extent, so there is room for improvement in that space. The Neato also moves very slowly so increasing the fluidity would also help in his regard. A final touch would be implementing a distance threshold where the Neato would stop before running into people's feet.

5 Obstacle Avoiding

5.1 High Level

For the obstacle avoiding challenge, our Neato drove in a line until it encountered an object and then turned, waiting until the object was no longer in its way to turn back again and continue. This could be explained by two states: Normal and Waiting. During the normal state, the Neato continued in a straight line and looked ahead in a 90 degree cone. If any object was detected (using "get object" in interface.py), then the Neato entered the Waiting state. Basically, "get object" was again used but shifted by 90 degrees to compensate for the turning. This allowed the Neato to detect whether the original obstruction that caused it to turn was out of the way. From here, the Neato turned again and resumed its straight path. Follow this link[4] for a video of the Neato avoiding various obstacles.

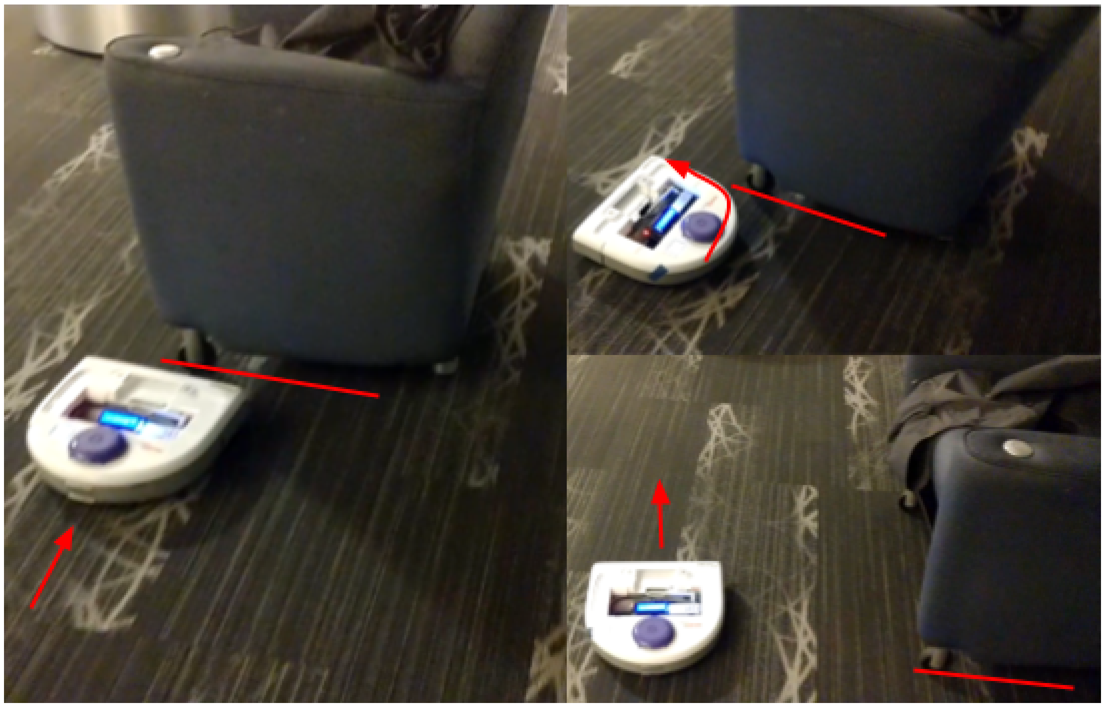
Figure 6: Our Neato detecting an obstacle, turning to avoid it, then continuing on when it gets clear

5.2 Challenges

The main challenge that came with avoiding obstacles was figuring out the appropriate logic behind the different states. There were a lot of "if" statements that became confusing before we decided to create more helper functions and some of the behavior was hard to conceptualize. As for coding, the most difficult part was coding the Neato to look for the obstacle it turned to avoid. We ended up writing a new function (the "get object" function) to handle this and afterwards it worked well.

5.3 Potential Improvement

With more time, we could make a more robust avoidance code. As of right now, our code can be delicate, especially if there are too many obstacles in the environment. This is especially true in the Waiting phase which does not detect additional obstacles when trying to avoid the original. We also do not have a way to keep track of direction besides the two different states so this could be problematic. Another major improvement would be the use of odom to set a beginning and end goal, therefore allowing the robot to figure out its own path while avoiding obstacles without adding odd complications.

6 Finite State Controller

6.1 High Level

The final problem required combining the activities of two finished behaviors into one system. We chose to link person following and wall following into one system that will follow a nearby wall until signaled to follow a person to another wall. For this design, the states would be either wall-following or person-following with transitions indicated by the bump sensors. This is shown graphically in Figure 8 and in video form here[5].

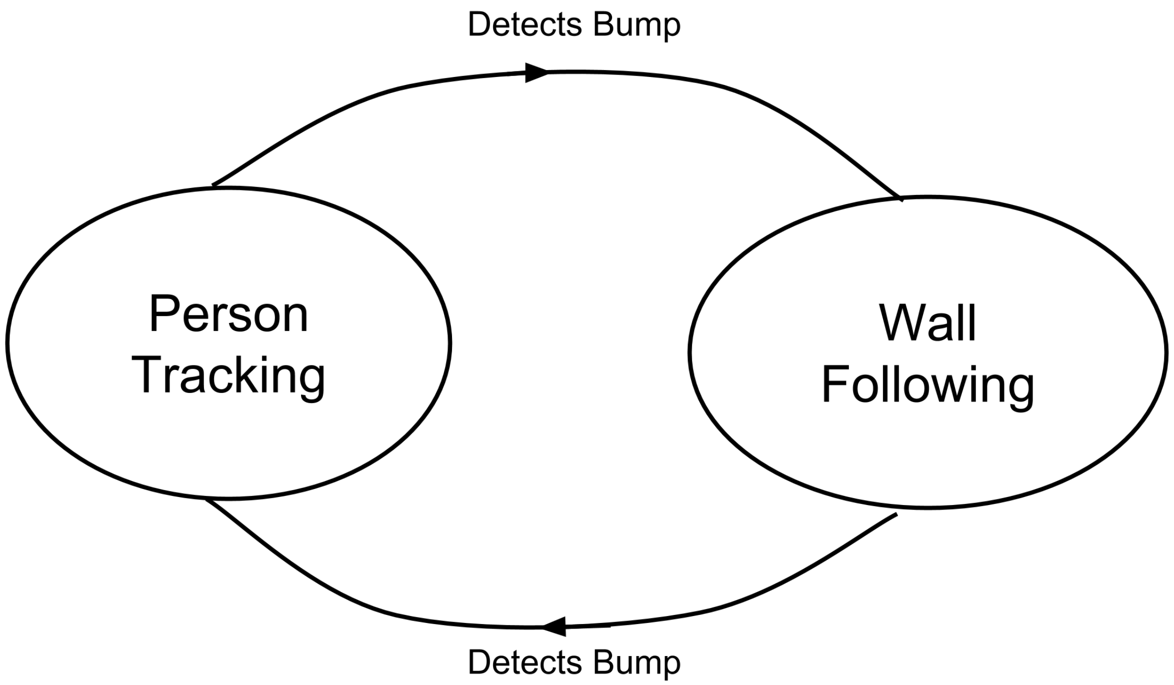

Figure 7: The states and transitions of our finite state control

6.2 Challenges

A significant hurdle for this problem was the transitions. Since we spent extra time debugging and streamlining the wall and person following applications, those were smoother but also over-fit to their respective applications. Specifically, for person following we were unable to use our usual function to look for a person in front since it detects changes in surroundings and therefor relies on the robot being stationary. Consequentially, we decided to rely on the bump sensors to change states. However, this presented its own difficulties, i.e. the process running the callback for bump sensors could get pushed back by the computation of the main processes. To avoid this we limited the execution times of the respective loops with a rospy.Rate() with inconsequential loss in speed.

6.3 Potential Improvement

Given more time, the first item to improve would be a more natural transition from wall to person following in the form of a function that detects people standing in front of the Neato. Additionally, we could always make the functions more robust to noise since as of now a nearby person could potentially be mistaken for a wall.

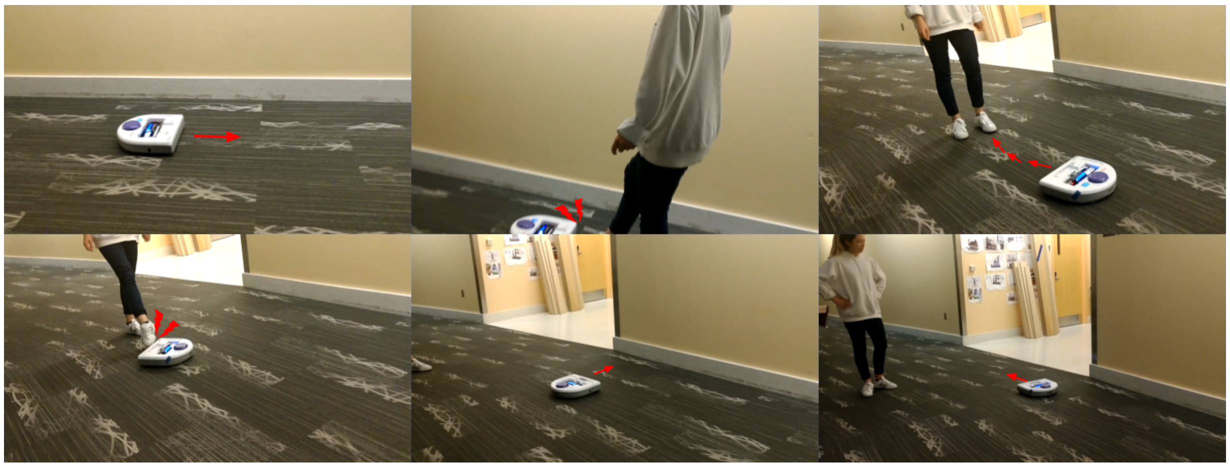

Figure 8: Storyboard of our Neato switching from wall following to person following and back.

7 Take Away

7.1 What worked

1. Separating the code between ROS commands in interface.py and other pure python scripts was **incredibly** helpful. It allowed for better compartmentalization, a reduction in number of imports in a given documents, and easy debugging.

2. ROS is a convenient and effective framework for Robot communication.

3. Drawing out the position and orientation of the Neato on a chalkboard is a powerful tool in solidifying the logic around a function as well as fact checking the implementation.

7.2 What we learned

1. Applications that require some form of position tracking require significantly more overhead and have greater potential to yield sketchy results. Loops that require just knowledge of immediate surroundings tend to be more robust, but also more limited.

2. We learned how to communicate between nodes through topics in ROS

3. We learned some powerful debugging tools such as rosbag or rviz that may take up slack that the python interpreter cannot.

4. ROS is a fickle mistress. You must treat her right if you want her love. Otherwise you will be inundated with error messages that have no equivalent online.

7.3 Links

Links to correct time in the video:

[1] - https://youtu.be/-ad1L5mY0yo
[2] - https://youtu.be/-ad1L5mY0yo?t=30
[3] - https://youtu.be/-ad1L5mY0yo?t=55
[4] - https://youtu.be/-ad1L5mY0yo?t=130
[5] - https://youtu.be/-ad1L5mY0yo?t=171