Spring 2019 Research Report

Rachel Lim

May 2019

Contents

1 Introduction

This report describes the research I worked on in spring 2019 in Professor Tomizuka's lab, under Jessica Leu's guidance. My work included coding portions of the human prediction [1] and mobile manipulator MPC control [2] experiments, as well as starting the design of more complex base and manipulator tracking algorithms that take into account the kinematics and dynamics of the manipulator to ensure safe and accurate movement.

1.1 Hardware

The experiments were conducted on ROBOTIS's TurtleBot3 Waffle Pi [3] platform with the OpenMANIPULATOR [4]. The TurtleBot drivebase is a differential drive, and the manipulator is a 5 joint arm, both described in more detail in the relevant sections below.

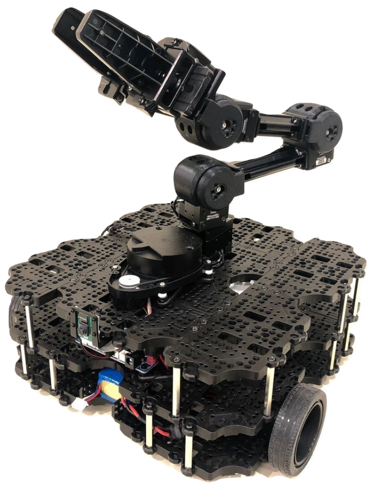


Figure 1: TurtleBot3 Waffle Pi with OpenMANIPULATOR [3].

1.2 Software

The TurtleBot platform is built work to work with the ROS (Robotics Operating System) framework. ROS is a software framework designed for robotic software development [5], and allows for highly modular development and the use of open source packages for common functionalities such as simulation and hardware drivers. ROBOTIS has built models of their robots for Gazebo, which were used extensively to test code before running it on the actual robot [6] [7].

Most code was written in Python, with some plotting scripts and the original MPC algorithms written in MATLAB. All code can be found in our github repo [8].

2 Background & Previous Work

2.1 MPC path planning

The human prediction and mobile manipulator control methods build upon the MPC algorithm developed by Jessica Leu last semester [9]. To be able to run the code directly on the TurtleBot, as well as to better interface it with the other ROS nodes, I had converted the code from MATLAB using quadprog into Python using CVXOPT. Both algorithms were functionally identical, finding the optimal path by solving a quadratic programming problem that penalized changes in the path from the old path, changes in velocity, and changes in acceleration, and that formed soft constraints around the obstacle with a certain margin.

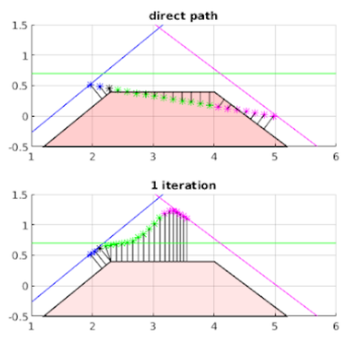
For the human prediction experiments, I modified the Python algorithm to work in more general cases, so that it can take in any robot location, obstacle locations and velocities, and desired margin and calculate the optimal path from there. For the mobile manipulator control, the addition of the manipulator into the optimization problem caused it to be large enough that we decided to leave it running in MATLAB, since quadprog was measurably faster than CVXOPT.

The optimization code can be found in `human_prediction/src/cfs_module.py`

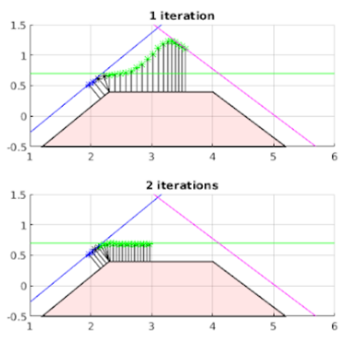
2.2 Low level base tracker

Since the optimization code publishes a series of waypoints, a low level controller is used to convert these into velocity commands and to ensure accurate tracking of that path. The main controller used was the one I wrote in the fall, which was based on the method described in this paper [10]. The low level controller also ran fast than the optimization node, which allowed it to be used for safety tracking, as described in the mobile manipulator section.

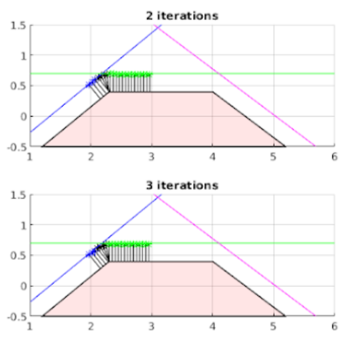
The tracking code can be found in `mobile_manipulator/src/pathmodule.py`



(a) Direct path from current location to target path, and result after one iteration.



(b) Result after second iteration.



(c) Result after third iteration.

Figure 2: MPC algorithm for path planning around obstacle

.

3 Human Prediction

For the first part of the semester, I worked on code for the human prediction and mobile manipulator control to be able to test the algorithms on the TurtleBot. The focus of the human prediction paper was being able to have humans and robots working safely together in factories, by further optimizing the path that robots can take while maintaining a safe distance from the human based on the predicted human behavior. The method is described in more detail in the following paper [1].

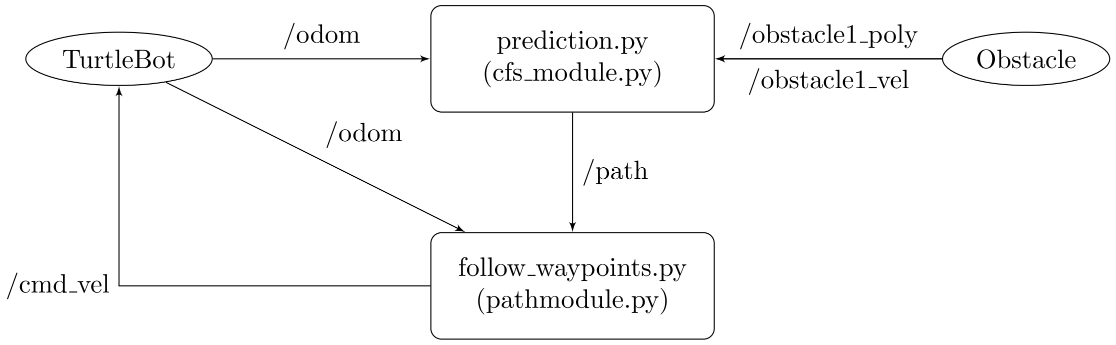


Figure 3: ROS node structure for the human prediction algorithm.

3.1 Working vs moving mode

In this method, human actions in a factory setting are characterized into two groups, a ”working mode,” where a human moves slightly but doesn’t deviate significantly from their original location, and a ”moving mode,” where the human moves with a clear direction with a faster velocity. The mode is detected based on the detected velocity and previous locations of the human. If the human is detected to be working, the human obstacle is set to have a zero velocity, and the margin around the human is set based on the variation in previous positions of the human. If the human is in moving mode, the margin is set to a preset value, and the velocity of the human is used to plan a path around the human. The algorithm predicting which mode the human was developed by Jesssica Leu in [1] and is included below. I implemented it in the python node and interfaced it with the optimization module.

The prediction code can be found in `human_prediction/src/prediction.py`

Algorithm 1 Prediction algorithm

```
1: procedure MovePred($\mathbf{h}^k_{obs}, p_{go}(k-1), p_{stop}(k-1)$)
2: Acc $\leftarrow getAcc(\mathbf{h}^k_{obs}$
3: endVel $\leftarrow PredictEndVel(Acc)$
4: if endVel $\leq$ threshold then
5:     $w_{go} \leftarrow r_1 p_{go}(k-1)$
6:     $w_{stop} \leftarrow r_2 p_{stop}(k-1) + b$
7: else
8:     $w_{go} \leftarrow r_2 p_{go}(k-1) + b$
9:     $w_{stop} \leftarrow r_1 p_{stop}(k-1)$
10: end if
11: $p_{go}(k), p_{stop}(k) \leftarrow normalize(w_{go}, w_{stop})$
12: return $p_{go}(k), p_{stop}(k)$
```

3.2 Experimental results

The experiments were run on the TurtleBot but with a "fake" obstacle, a node that published vertices of a polygon and velocities of that object. In the experiment, the obstacle began by moving quickly in the +y direction (upwards in the figures below), as can be seen in how the TurtleBot initially detects that it can converge quickly to the original path as the obstacle would have left by then. However, the obstacle switches into working mode, and the new path is planned around this new location (middle image). Finally, the obstacle moves away to the left, allowing the robot to quickly converge back to the desired path.

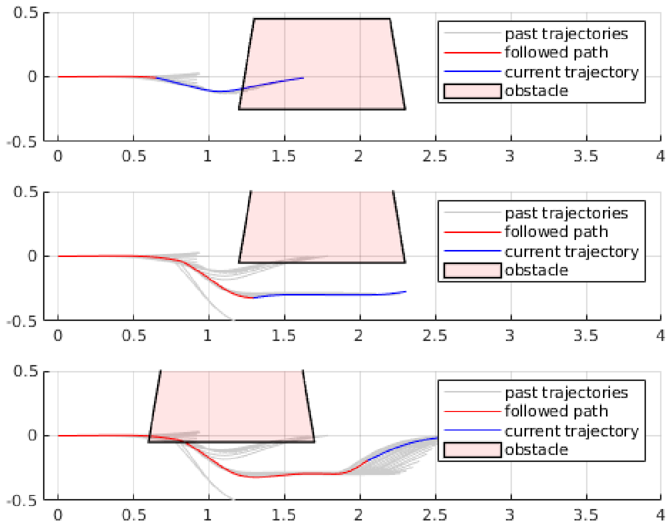


Figure 4: Experimental results of human prediction algorithm.

4 Mobile Manipulator MPC control

The mobile manipulator MPC was designed by Jessica Leu in [2]. I worked on making sure it ran on the TurtleBot by designing the tracking controllers for both the base and manipulator, and created a safety controller for the base that could react faster than the MPC algorithm.

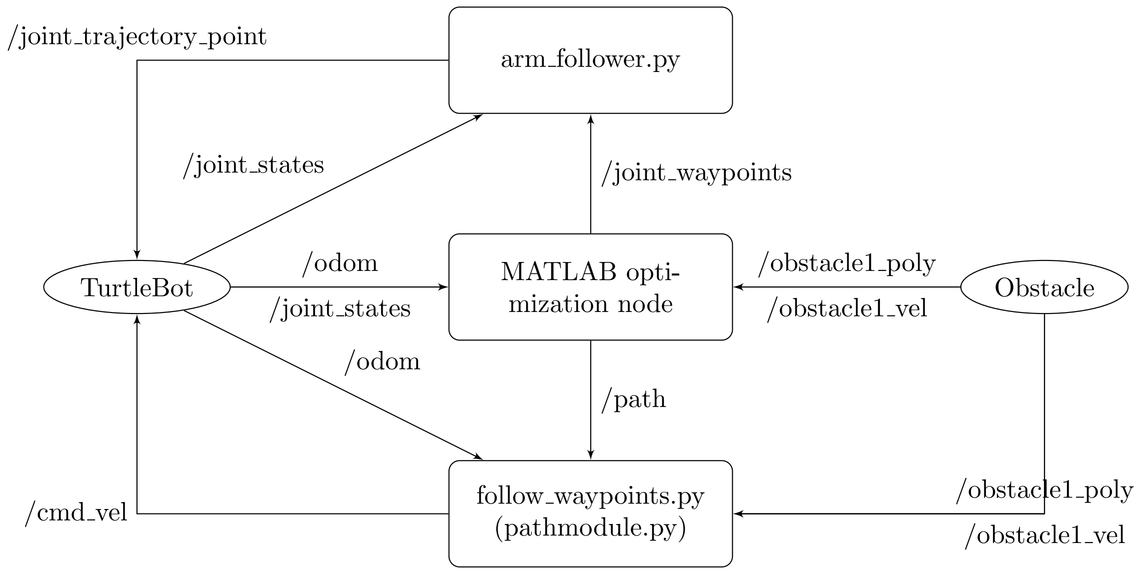


Figure 5: ROS node structure for mobile manipulator control.

4.1 Low level manipulator tracker

The low level tracker simply takes the waypoints published by the MPC node, which runs at 2 Hz, and often slower due to the size of the optimization, and interpolates them for the low level controller which runs at 10 Hz. For safety, the velocity of each joint to 1 rad/s, so waypoints more than half a radian from the last were further interpolaated. In the last section, more safety features were added, but in these experiments the MPC algorithm was used to ensure the arm would never try to reach an invalid position.

The manipulator controller code can be found in `mobile_manipulator/src/arm_follower.py`

4.2 LQR controller

In addition to the low level base tracker, as described in Section 2.2, I adapted the LQR controller from [9], and wrote a Python version of it. It's written to be easily swapped with the original controller. A two even simpler controllers were also written for debugging purposes.

All four controllers are implemented in `mobile_manipulator/src/pathmodule.py`

The LQR controller is written in `mobile_manipulator/src/lqr_module.py`

4.3 Safety controller

Since the MPC algorithm runs slower than the low level controller, changes in the direction or velocity of the obstacle are first detected in the controller. To avoid hitting these obstacles, I wrote a faster collision detection algorithm in the low level controller, to detect if the TurtleBot's current trajectory would cross into the obstacle's margin of safety. If a collision was detected, the path would be adjusted accordingly. In our first method, the waypoints were directly adjusted to turn the robot away from the obstacle. In the second, it called the optimization node again, but ran the optimization with simply the base and obstacle, and with a shorter horizon in order to speed it up. This method in general gave better performances, and made it easier for the MPC to take back over again. However, the first method was a guaranteed way to ensure the path was adjusted immediately regardless of the situation.

The safety controller was implemented in `mobile_manipulator/src/pathmodule.py`

5 Low Level Mobile Manipulator Kinematic and Dynamic Control

My final project for this semester involved extending the low level controllers for the base and mobile manipulator to ensure the manipulator always moved safely and that the base continued to track the desired path accurately. I began by finding the kinematic model of the manipulator and the weights and inertias of each joint, which allowed me to ensure the arm never tried to reach positions where it would hit the base or ground, as well as to calculate the overall center of mass and total inertia of the robot and see how that affects the base. In the future, the dynamic model will be added so the low level controller can compensate for the torques generated by the arm.

5.1 Coordinate frame convention

The following frames were set up:

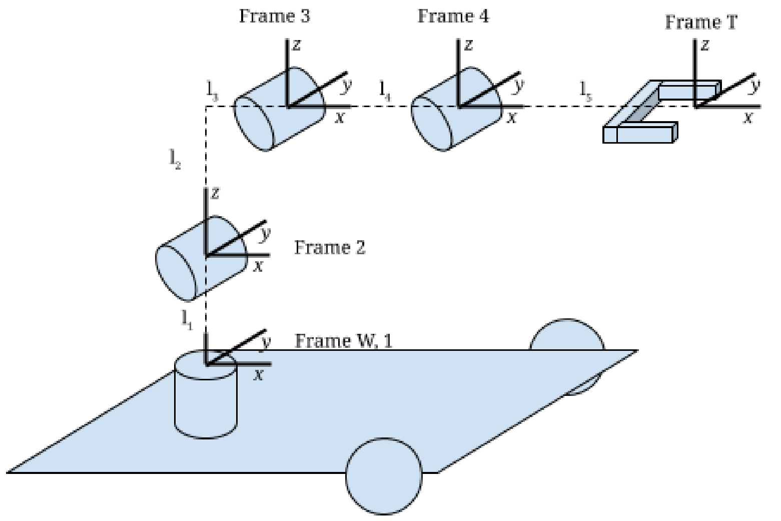


Figure 6: Coordinate frames for arm.

Frame W: world frame, fixed at joint 1
Frame 1: frame that rotates with joint 1
Frame 2: frame that rotates with joint 2
Frame 3: frame that rotates with joint 3
Frame 4: frame that rotates with joint 4
Frame T: frame attached to tip of end effector
They have the following transforms:

$$g_{W1} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$g_{12} = \begin{bmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_2) & 0 & \cos(\theta_2) & \ell_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$g_{23} = \begin{bmatrix} \cos(\theta_3) & 0 & \sin(\theta_3) & \ell_3 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_3) & 0 & \cos(\theta_3) & \ell_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$g_{34} = \begin{bmatrix} \cos(\theta_4) & 0 & \sin(\theta_4) & \ell_4 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_4) & 0 & \cos(\theta_4) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$g_{4T} = \begin{bmatrix} 1 & 0 & 0 & \ell_5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Where ℓ_1, ℓ_2, ℓ_3, ℓ_4, and ℓ_5 are constants and given by:

$$\ell_1 = 0.038 \text{ m} \tag{6}$$
$$\ell_2 = 0.128 \text{ m} \tag{7}$$
$$\ell_3 = 0.024 \text{ m} \tag{8}$$
$$\ell_4 = 0.124 \text{ m} \tag{9}$$
$$\ell_5 = 0.141 \text{ m} \tag{10}$$

5.2 Arm safety controller

From testing in gazebo, the joints were found to have physical rotational constraints as follows:
Joint 1: -2.8 to 2.8 rad
Joint 2: -$\frac{\pi}{2}$ to $\frac{\pi}{2}$ rad
Joint 3: -0.9 to 1.3 rad
Joint 4: -$\frac{\pi}{2}$ to $\frac{\pi}{2}$ rad
Any joint waypoints received that were beyond these values were mapped to these minimum/maximum values for better tracking and controlling. Additionally, the kinematic model of the TurtleBot arm was used to ensure the arm did not try to reach a position were it would hit itself.

In the world frame, the base dimensions are: -0.07 to 0.20 along the x-axis, -0.13 to 0.13 along the y-axis, and -0.035 to -0.15 along the z-axis. Additionally, the LDIAR is at 0 to 0.135 along the x-axis, -0.035 to 0.035 along y-axis, and -0.035 to 0.01 along the z-axis. A 2 cm margin was added around everything, and the following equations used to find the positions of joint 4 and the manipulator tip, which were then checked against the previously listed dimensions.

$$\bar{q}_W = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{11}$$

$$\bar{q}_4 = g_{W1} g_{12} g_{23} g_{34} \bar{q}_W \tag{12}$$

$$\bar{q}_T = g_{W1} g_{12} g_{23} g_{34} g_{4T} \bar{q}_W \tag{13}$$

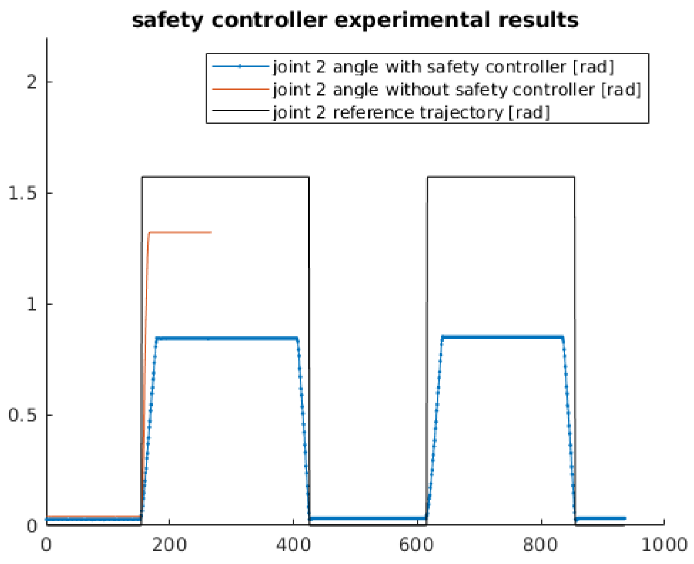


Figure 7: Experimental results for joint movement with and without safety controller.

If the arm waypoints were detected to describe an invalid position, each joint angle was halved until the arm reached a valid position. The original velocity safety controllers were also left in place.

The plot below shows the result of the safety controller, when a joint angle that causes the arm to hit the ground is sent. The orange line shows the movement before the safety controller: the arm moves to reach that angle very quickly, and then hits the ground and dies. The blue line shows that the arm realized the desired angle is unachievable, and thus only tries to reach half of that. It also shows the slower, more controlled movement to reach the desired angle.

These results can also be seen in the following videos, which show the arm movement before and after the safety controller.

Before: https://drive.google.com/file/d/1nHmA248WhNcNuikgzM9mTozglxpTr4mb/view?usp=sharing

After: https://drive.google.com/file/d/1JIB8k6J_yu2VpHdKAHYxq1EDVGvyhbZR/view?usp=sharing

This code was implemented in `mobile_manipulator/src/arm_safety_follower`

5.3 Center of Mass and Inertia

Additionally, the kinematic model was used to find the center of mass of the entire base + arm system based on the current arm position. The components of the robot were weighed in the sections listed below in the table. These measurements, and the provided CAD model of the robot, were loaded into SolidWorks where the moment of inertia calculations were done. These were used to obtain the center of mass and overall inertia of the robot.

Component	center of mass (x,y,z) [m]	mass [kg]	frame	moment of inertia [g*m^2]
Base + motor1	(0.08, 0, -0.115)	2.332	W	$\begin{bmatrix} 18.7 & 0.379 & 1.79 \\ 0.379 & 18.8 & 0.484 \\ 1.79 & 0.484 & 26.8 \end{bmatrix}$
Motor2	(0, 0, 0.03)	0.082	W	$\begin{bmatrix} 0.0132 & 0 & 0 \\ 0 & 0.0227 & 0 \\ 0 & 0 & 0.020 \end{bmatrix}$
Arm2	(0, 0, 0.0875)	0.042	2	$\begin{bmatrix} 0.0047 & 0 & 0 \\ 0 & 0.0267 & 0 \\ 0 & 0 & 0.0283 \end{bmatrix}$
Motor3	(0.0125, 0, 0.1275)	0.082	2	$\begin{bmatrix} 0.0132 & 0 & 0 \\ 0 & 0.0227 & 0 \\ 0 & 0 & 0.020 \end{bmatrix}$
Arm3	(0.055, 0, 0)	0.033	3	$\begin{bmatrix} 0.0038 & 0 & 0 \\ 0 & 0.0112 & 0 \\ 0 & 0 & 0.0125 \end{bmatrix}$
Motor4	(0.1125, 0, 0)	0.082	3	$\begin{bmatrix} 0.0132 & 0 & 0 \\ 0 & 0.0227 & 0 \\ 0 & 0 & 0.020 \end{bmatrix}$
Motor5	(0.045, 0, 0)	0.082	4	$\begin{bmatrix} 0.0132 & 0 & 0 \\ 0 & 0.0227 & 0 \\ 0 & 0 & 0.020 \end{bmatrix}$
Gripper	(0.065, 0, 0)	0.147	4	$\begin{bmatrix} 0.2034 & 0.0003 & -0.0137 \\ 0.0003 & 0.1180 & -0.0003 \\ -0.0137 & -0.0003 & 0.2511 \end{bmatrix}$

6 Conclusion

My work this semester built on my previous work on the MPC obstacle avoidance path planning, to include more complex low level tracking as well as manipulator control. This included adjusting how obstacles are treated if they're detected to be in working or moving mode, adding additional collision detection into the base tracking, and working on a kinematics model for the arm for a safety controller for the manipulator. I also began work on identifying how the arm movements affect the base tracking, in order to achieve more accurate movement.

7 Future work

In the future, I plan to finish the experiments to determine how exactly changes in the arm configuration affect the base. So far most of the work included restricting the motion to ensure safe movement or a relatively centered and low and center of mass, but knowing that relationship would allow for more flexibility in the arm motions by having the base compensate accordingly. This involves finishing the measurements on the wheel coefficient of friction, and how much that changes as the center of mass moves to one side of the robot.

Additionally, I will continue the work with the dynamic model, setting up the symbolic equations in Python and accounting for the arm torques in the base tracking. By knowing the torques generated, we can increase the velocity commands being sent to one of the wheels in order to counteract that motion, and help keep the robot on a steady path. Additionally, we can determine if some motions are simply too fast for the robot to handle safely, and slow those motions down even more if so.

References

[1] Leu, J. ”EUA-MPC: Motion planning for robots in industrial HRI system.” In submission. .

[2] Leu, J., Lim, R., Tomizuka, M. ”Safe and Coordinated Hierarchical Receding Horizon Control for Mobile Manipulators.” In submission.

[3] TurtleBot3 Overview. https://github.com/jessicaleu24/NSF-T3.

[4] TurtleBot3 Manipulation. http://emanual.robotis.com/docs/en/platform/turtlebot3/manipulation/ #manipulation.

[5] About ROS. http://www.ros.org/about-ros/.

[6] Gazebo. http://gazebosim.org/.

[7] TurtleBot3 Manipulation: Simulation. http://emanual.robotis.com/docs/en/platform/turtlebot3/ manipulation/#simulation.

[8] https://github.com/jessicaleu24/NSF-T3.

[9] Leu, J., Liu, C., and Tomizuka, M. “M-Analysis for non-convex motion planning using MPC framework.” In submission.

[10] Jadlovský, J., Kopčík, M. ”Distributed Control System for Mobile Robots With Differential Drive”.