Path Planning for Manipulator Arm Mounted on Mobile Robot Base

Kyle Cantrell
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA
kjcantrell@wpi.edu

Craig Miller
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA
cdmiller@wpi.edu

Jordan Nelson
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA
jvnelson@wpi.edu

***Abstract*—Using an OpenManipulator and TurtleBot Waffle Pi in conjunction with ROS, Gazebo and Rviz a dynamic model of the system was generated and compared with a MATLAB-based model to track a determined trajectory, pick up an object and return to the initial arm pose.**

Keywords – Robotics; mobile robots; manipulator arm; path planning; pick and place; dynamic model; state space

I. Introduction

This project objective is to solve the problem of moving an object using a mobile robot with manipulator arm (TurtleBot Waffle Pi/OpenManipulator) after going from point A to point B in a 3D space. The performance of this task is simulated using Gazebo Robotic Simulation, ROS (Robotic Operating System) and RViz (a visualization tool for ROS nodes and data) and compared with experimental data and similar MATLAB models.

The application of mobile robots with manipulator arms have shown great promise in industrial use. Most notably through functions where robots are prescribed to go into environments where human interaction is not desired i.e. bomb defusal, dangerous, hazardous work zones, time frames where humans aren't present and repeated motions that humans would be unprofitable performing. Popular robots of this type resemble the flagship product of Endeavor Robotics' "First Look" product line. Understanding these specific systems is greatly beneficial to robotics engineers due to the practical application of robotics in a critical environment.

The analysis and modeling of a mobile robot/manipulator platform is particularly difficult. The OpenManipulator robot arm we considered has 4 degrees of freedom which in itself is difficult to model; with multiple elbow up/down considerations when calculating the forward and inverse kinematics, and articulation of the gripper so as to properly grasp/release objects. The TurtleBot Waffle Pi mobile robot it attaches to also has two actuators and that require a control scheme to correctly translate the assembly to the proper coordinates in planar space. A naïve approach to this problem will not accurately communicate the desired objective of the operator to the robot.

This problem has been solved before in slightly different situations but in this examination we are looking at comparing results between MATLAB based dynamic models and experimental data gathered from ROS models and similar real-world mobile robots (TurtleBot Burger). This plan should give us a better understanding of errors between the two systems and how to model them in the future.

The key components are ROS, RViz, and Gazebo modeling to complete the actual visualization/data driven representation of what we are trying to accomplish and the MATLAB dynamic models to compare results. Part of this comparison would be the typical measurement parameters that have been used in numerous assignments in our robotics studies so far such as joint angles vs time, control input as a function of time, end effector position vs time, and various errors in the system as a function of time. The culmination of our work is expected to produce a running simulation in Gazebo's physics engine environment able to navigate a simulated environment with lidar mapping to grip and move a virtual beverage can. The following sections will detail the steps we took in order to reach this objective and all pertinent information in our research.

II. Robot Models

A. TurtleBot Waffle Pi

The TurtleBot Waffle Pi is a simple differential-drive mobile robot. Reference [5] introduces a complete kinematic and dynamic model for the differential-drive mobile robot. Fig. 1 and (1) are taken directly from [5] without modification. Using the methods described in [5] along with the technical specifications of the actual robot, we were able to establish simple forward and inverse kinematic transformations between the world coordinate frame and mobile robot frame. Reference [5] also describes how it is possible to calculate the

wheel torques in terms of wheel position, wheel velocity, and rotational angle.

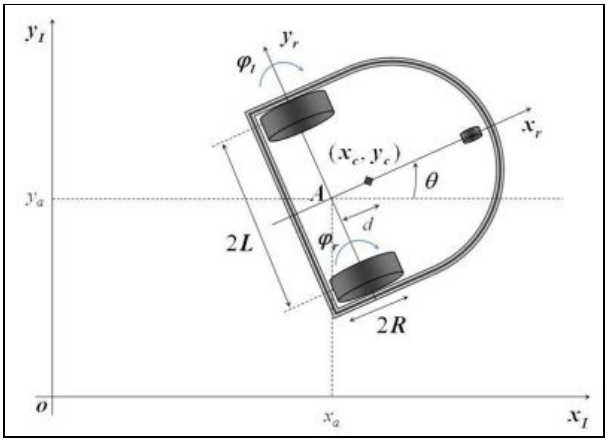

Fig. 1. Differential Drive Mobile Robot

$$\overline{M}(q)\dot{\eta}+\overline{V}(q,\dot{q})\eta=\overline{B}(q)\tau$$

Where

$$\overline{M}(q)=\begin{bmatrix} I_w+\frac{R^2}{4L^2}\left(mL^2+I\right) & \frac{R^2}{4L^2}\left(mL^2-I\right) \\ \frac{R^2}{4L^2}\left(mL^2-I\right) & I_w+\frac{R^2}{4L^2}\left(mL^2+I\right) \end{bmatrix}$$

$$\overline{V}(q,\dot{q})=\begin{bmatrix} 0 & \frac{R^2}{2L}m_c d\dot{\theta} \\ -\frac{R^2}{2L}m_c d\dot{\theta} & 0 \end{bmatrix}, \quad \overline{B}(q)=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Equation 1. Dynamic Model of Differential Drive Mobile Robot

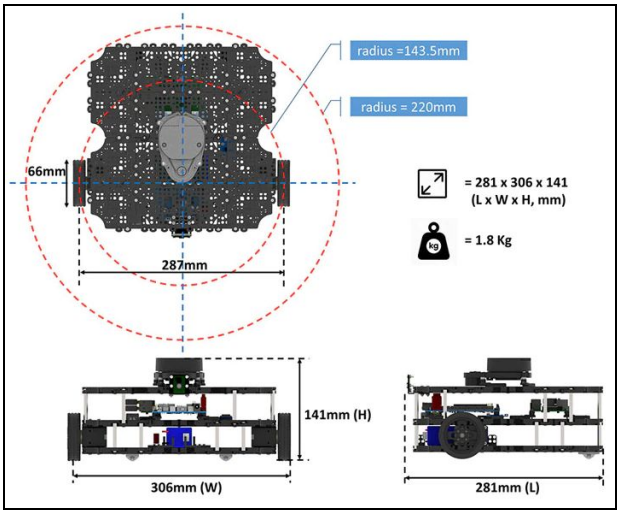

Fig. 2. TurtleBot WafflePi Specifications

B. OpenManipulator

Homogeneous transformation matrices were derived for the OpenManipulator arm using methods described in [3] in conjunction with the well-documented physical parameters of the robot. A dynamical model was derived using Lagrange's Method. The forward and inverse kinematics were transcribed similar to the 4-DOF robot research article referenced [8], where a pose specific to the end effector was needed to keep the end effector in desired orientation.

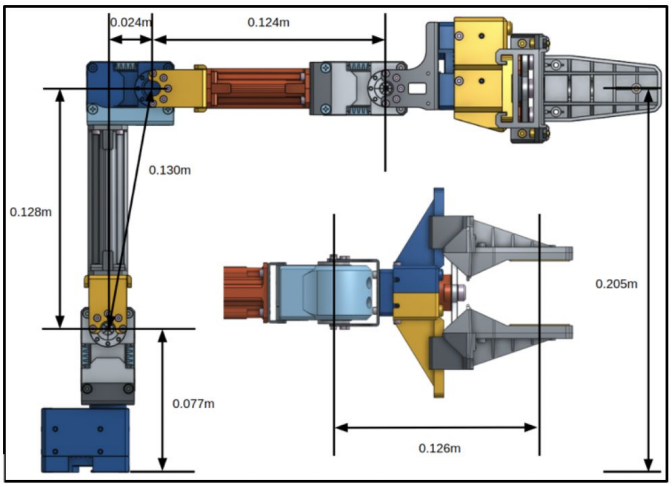

Fig. 3. OpenManipulator Robot Arm

C. OpenManipulator with TurtleBot Waffle Pi

Based on the mounting position of the OpenManipulator arm, additional transformation matrices were developed to transform positions from the mobile robot to the end-effector coordinate frame. Since the OpenManipulator arm remained stationary while the mobile robot was actually moving, we safely assumed that the center of gravity remained constant through this part of the trajectory.

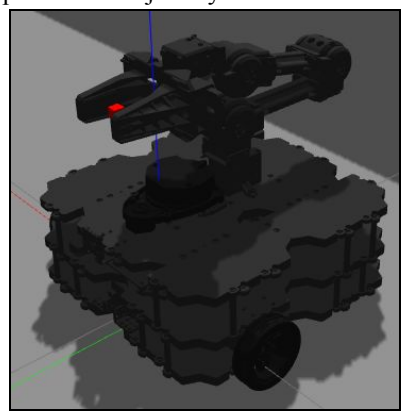

Fig. 4. OpenManipulator with TurtleBot Waffle Pi

III. Control Methods

A. TurtleBot Waffle Pi

The TurtleBot Waffle Pi is an advanced robotic platform leveraging the massive, community supported ROS (Robotic

Operating System) and a bevy of packages to support it. The main interest in regards to this article are the navigation and SLAM (Simultaneous Localization and Mapping) packages as it relates to mobile robotics.

Starting with the navigation package, we were able to use the highly abstracted interface to specify the expected parameters of desired X, Y, Z coordinates and a pose in Euler angles. This equates to specifying where on a cartesian plane the robot will be targeting and the orientation of the mobile robot at the end of the maneuver.

Once the destination is entered, the robot can begin it's path planning analysis and update. In ROS, this is handled by the DWA (Dynamic Window Approach) Planner node.

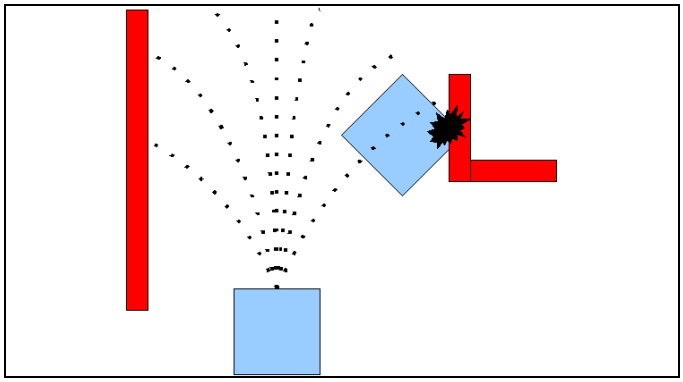

Fig. 5. DWA Planning and Scoring

Fig. 5 shows the gist of how the system works at a high-level. In essence, the system uses SLAM data, in our case it is a LIDAR sensor, to forward-simulate a series of maneuver choices. The simulations update a cost-map containing all routes to the desired position and the familiar Dijkstra's method is used to select the next waypoint in the "local" maneuver enroute to the "global" destination. In effect, this method uses several topics discussed in RBE 502 including trajectory generation in terms of angular and linear velocity. The important part of the control is located in the "DWA Planner" source code. The planner collects the the LIDAR point cloud data and feeds it into a subroutine that determines the best, immediate, local course of action. This is clearly the high level planning task. The output of the task is sent to the ROS Master node whom then distributes the high-level command of velocities into a "Joint Effort Controller". In the JointEffortController source code, we see familiar control strategies such as PID and PD, Position Control and motion control. In our case there are three different control systems collaboratively working together to achieve a single goal. Fig. 6 shows the flow of information in the TurtleBot system and ROS. Notice also the domain boxes illustrating where the Turtlebot and ROS begin and end. This decoupling demonstrates the complexity of infrastructure required to make a complicated task such as our chosen task achievable with minimal effort. For instance, in our simulation, the point-to-point motion and trajectories were explicitly defined. In our experimental setup, the system is only given a high level notion of our desired tasks which is remarkable.

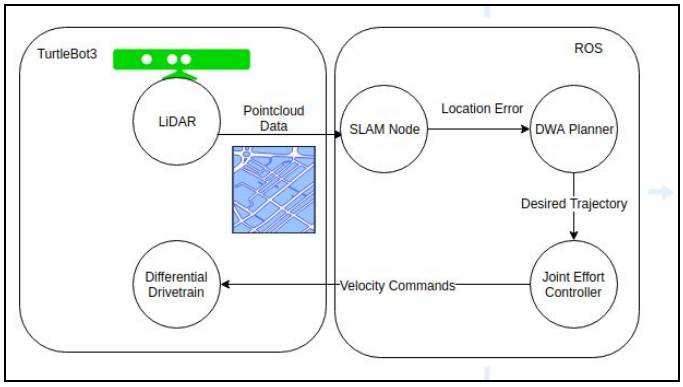

Fig. 6. Control flow of Turtlebot and ROS as it relates to control methodologies

The mobile robot control encountered in this project exemplifies just how much work has been accomplished in our area of study and how much there is to learn. Alternatively, it is a positive experience in that we are able to identify techniques we have been taught in RBE502 as a result of our research in ROS.

Comparatively, in the simulation space, simple trajectory control was employed using the error dynamics learned in RBE502 Week 10. The method used was to generate a trajectory of velocities based on time and via points as opposed to continuous, time-driven trajectory tracking. In cases of definitive motion not requiring smooth motion, this method presents an easy implementation.

For the MATLAB analysis, we implemented an Error Dynamics Trajectory Tracking solution as described in Section 11 of [1]. As we will discover in later sections, the error dynamics model produces idealized position results and discontinuous velocities which would be impossible to execute on a real-world robot

B. OpenManipulator

The OpenManipulator is another robotic platform, specifically a 4 Degree-of-Freedom arm manipulator. It is a product also from Robotis, the same company as the TurtleBot. Control of the arm is more straightforward than the mobile base but much of the control is abstracted from the user as is the case with the mobile control. In this instance, an entire motion planning framework dubbed "Moveit" is used to accomplish these tasks.

As was the case with the mobile robot, these packages allow us to specify simple parameters for the manipulator to accomplish with its own decision making. Some of these methods included in the library are:

- STOMP (Stochastic Trajectory Optimization for Motion Planning)
- Search Based methods
- CHOMP (Covariant Hamiltonian Optimization for Motion Planning)

In MATLAB, initial testing was attempted with PD Control plus Feedforward. For the practical implementation of controllers in robot manipulators [1] the reduction in computation time for can lead to higher processing frequency which was considered helpful in this 4 DOF application.

$$\tau = K_p\tilde{q} + K_v\tilde{q}dot + M(q_d)\ddot{q}_d + C(q_d, \dot{q}_d)\dot{q}_d + g(q_d)$$

Equation 2. PD Plus Feedforward Control Law

However after a few iterations and tests the overall desired trajectories and positions weren't being achieved in acceptable time frames so a PID controller was implemented. In cases such as this where the robot model contains gravitational torque vectors simple PD control laws won't support a desired outcome.

Using PID control and position/velocity/integral gains the overall performance of the robot manipulator was greatly improved.

$$\tau = K_p\tilde{q} + K_v\tilde{q}dot + K_i\int_0^t \tilde{q}(\sigma)d\sigma$$

Equation 3. PID Control Law

IV. Simulation

A. Overview

In both simulation methods, the robot is controlled to reach the desired positions and joint angles a shown in Table 1 and Table 2.

ID	Timeline of Events		
	Event	***Time (s)***	***Pose (m)***
1	Start	0	0,0,0
2	Mobile Robot at Via Point A	13.8	-2,1,0
3	Mobile Robot at Can	29.9	-1.5,3.75,0
4	Arm Started Reaching for Can	30.0	-1.5,3.75,0
5	Arm Grabbed Can	42.0	-1.5,3.75,0
6	Arm Returned to Initial Pose	48.0	-1.5,3.75,0

Table 1. Timeline of Events

ID	Timeline of Joint Angles			
	Joint 1 (rad)	**Joint 2 (rad)**	**Joint 3 (rad)**	**Joint 4 (rad)**
1	0	-1.57	-1.37	0.23
2	0	-1.57	-1.37	0.23
3	0	-1.57	-1.37	0.23
4	0	-1.57	-1.37	0.23
5	0	0.43	-0.1	-0.31
6	0	-1.57	-1.37	0.23

Table 2. Timeline of Joint Angles

B. MATLAB

Using the analytical models identified in the previous section, we wrote two MATLAB programs - one for the TurtleBot Waffle Pi and one for the OpenManipulator - that independently solve for joint positions, joint velocities, global positions, and global velocities over the desired trajectory. The MATLAB simulations were executed using the iterative ODE ordinary differential equation solvers. In both programs, we see quick convergence to the desired positions.

For the model of the OpenManipulator arm the first step involved estimating/recording all the physical characteristics of the robot and the associated weights, joint lengths, centers of masses and inertias. Then under a move arm function the gains, desired and initial conditions were set under inverse kinematics calculator using textbook references and basic trigonometric calculations [3]. The joint and end effector positions were mapped to compile figures in joint space to get better representations of the motions being performed. In solving for the planar arm ode initially ODE45 was attempted but due to the iteration/calculation time it was not solving in an acceptable amount of time so a stiff ODE (ODE15s) was implemented which produced worthy results. M, C and G matrices were used with a separate differential equation program and substituted. PID control was then implemented [1].

C. Gazebo and Robot Operating System (ROS)

By running Gazebo, ROS, and RViz on an Ubuntu machine, we were able to perform a comprehensive simulation of the integrated robot. Following the Pick and Place example detailed in [6], an accurate model of the robot was loaded into a virtual environment. Within the environment we were able to visualize simulated sensor data, view real-time joint torques, and build a 2D map using SLAM. In order to extract meaningful data from the simulation, we wrote a listener node that records time-stamped parameters in a .CSV file.

D. Performance Comparison

As we see in Fig. 7 below, all three control methods successfully reached the via point and end destination. The Error Dynamics modeled calculated in MATLAB is clearly the most idealized trajectory, but it unrealistic to achieve in a

real-world application. Both the ROS simulation and true experimental results display much smoother trajectories.

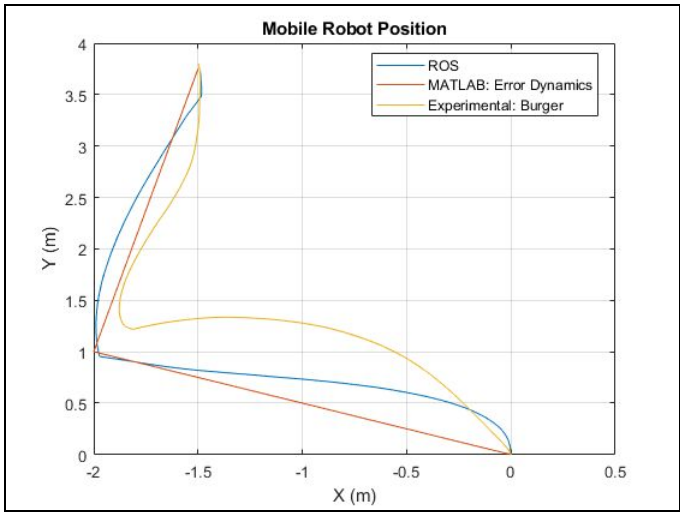

Fig. 7. Mobile Robot Position, Overhead View

By normalizing the timescale of each control method, we can compare the position and velocity performance over time. We observe that the positions in the ROS and Experimental results are offset with respect to time, but they perform very similarly.

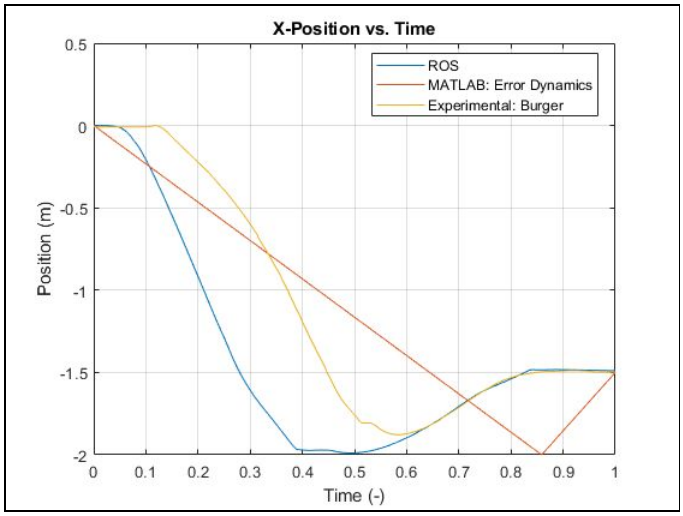

Fig. 8. Mobile Robot X-Position vs. Time

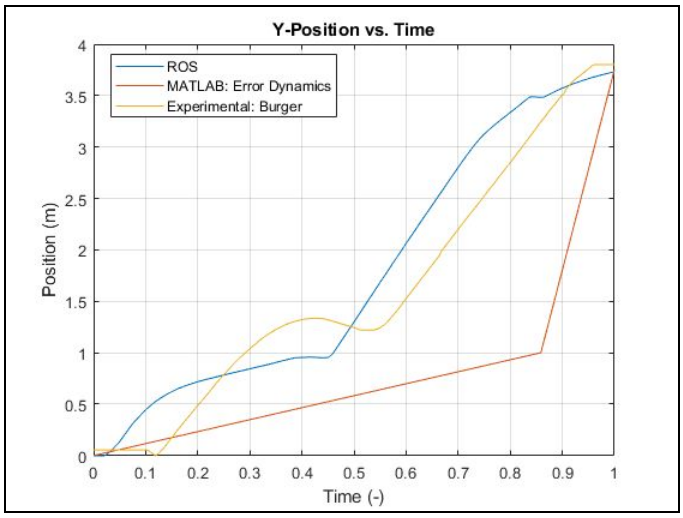

Fig. 9. Mobile Robot Y-Position vs. Time

In the Error Dynamics orientation shown in Fig. 10, we see a discontinuity. Obviously, this is impossible in a real-world scenario. Since the Error Dynamics code dictates a perfectly linear desired path, the corresponding velocity terms are constant and result in inaccurate orientation behavior.

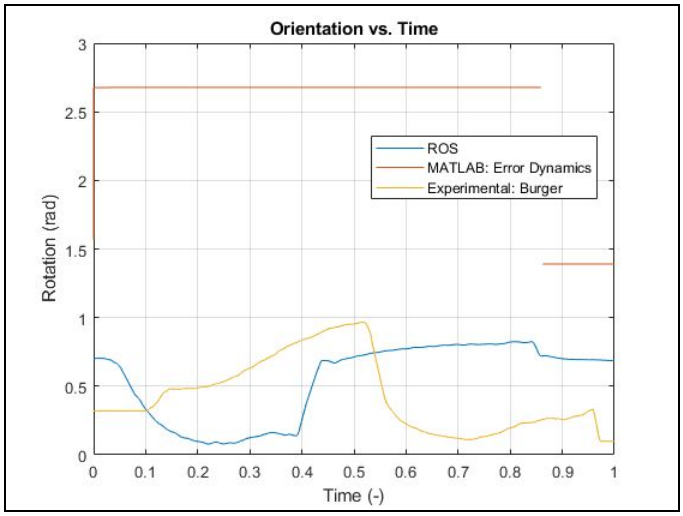

Fig. 10. Mobile Robot Orientation vs. Time

Similar to the Error Dynamics behavior shown in Fig. 10, we can see that the linear and angular velocity terms are also discontinuous. Again, we observe that the ROS and Experimental results show similar performance with a slight phase shift.

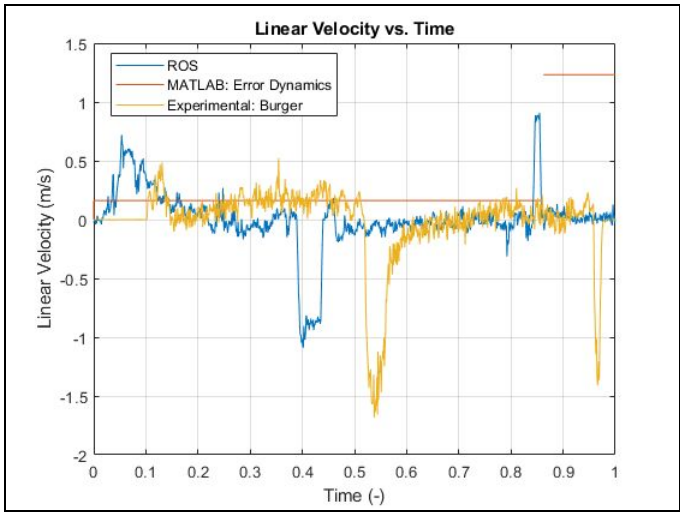

Fig. 11. Mobile Robot Linear Velocity vs. Time

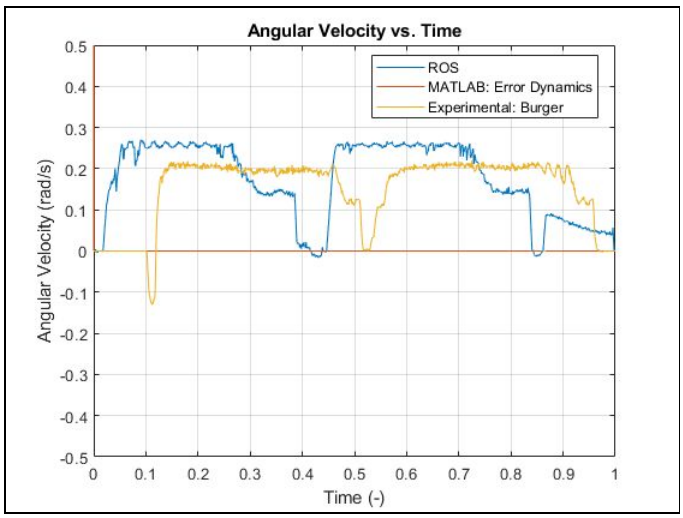

Fig. 12. Mobile Robot Angular Velocity vs. Time

For the OpenManipulator comparison the joint angles of the ROS model were applied to a SolidWorks sketch of the frame and from there the x,y,z positions of the joints were set as the desired positions for the MATLAB models, with two movement files resulting in the openmanipulator arm going from starting position to the grabbing position where the gripper actuates to grab the can and then return on the same trajectory. Based on elements of manipulator design a PID control was implemented and the data arm was actuated to and from the mentioned positions. Based on the MATLAB data and stick models the arm was able to achieve both position changes in the desired time frames however the theta/torque values differentiate themselves.

ID	Timeline of End-Effector Positions		
	X(m)	Y(m)	Z(m)
1	0.119	0	0.1206
2	0.1411	0	0.31729
3	0.119	0	0.1206

Table 3. Timeline of End-Effector Positions

Based on the center of the base of the manipulator arm on the robot, for the position plots these values were translated based on the mobile robot coordinate position. The MATLAB file manages to translate to the required positions for both movements and we can see that the overall position is correct from the stick model of the motion.

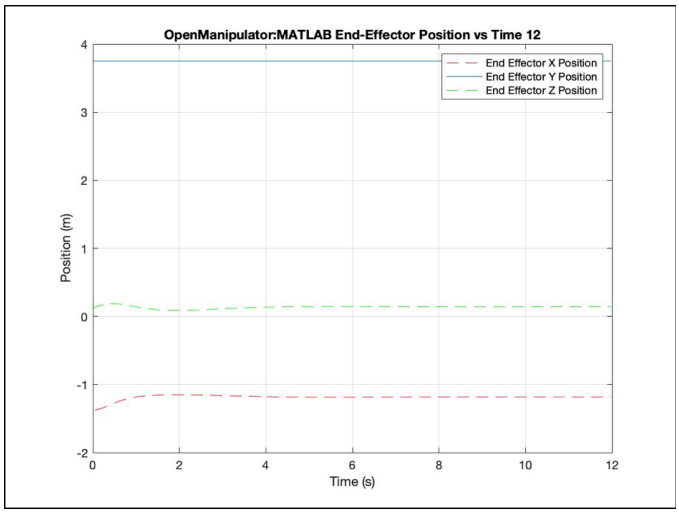

Fig. 13. End-Effector Position vs. Time 1-2

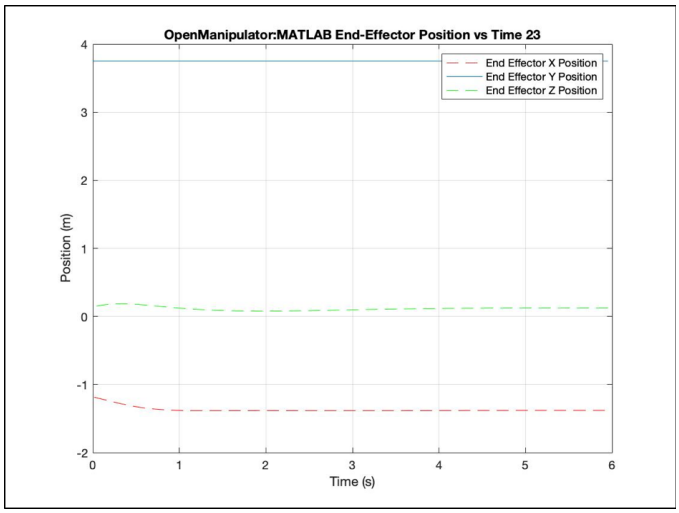

Fig. 14. End-Effector Position vs. Time 2-3

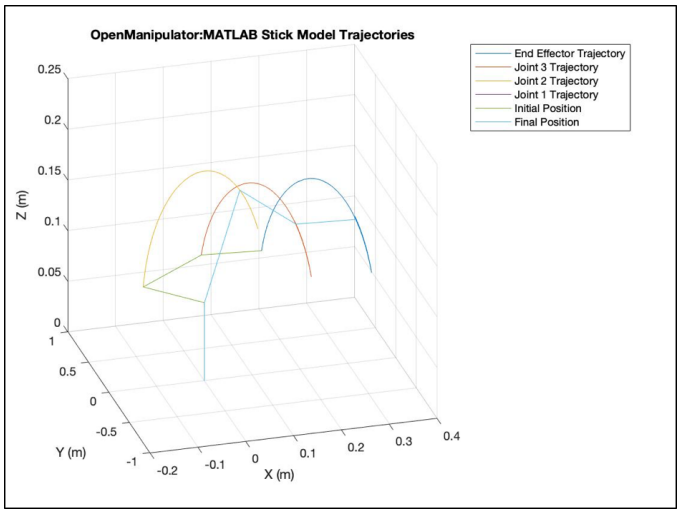

Fig. 15. OpenManipulator Positions in Joint Space

After examining joint torques and positions of arm lengths in both ROS and MATLAB it was discovered that the reference angles used were different which would cause different results in joint position/joint velocity and torques over time however the actual end effector manipulation based on coordinates remained correct.

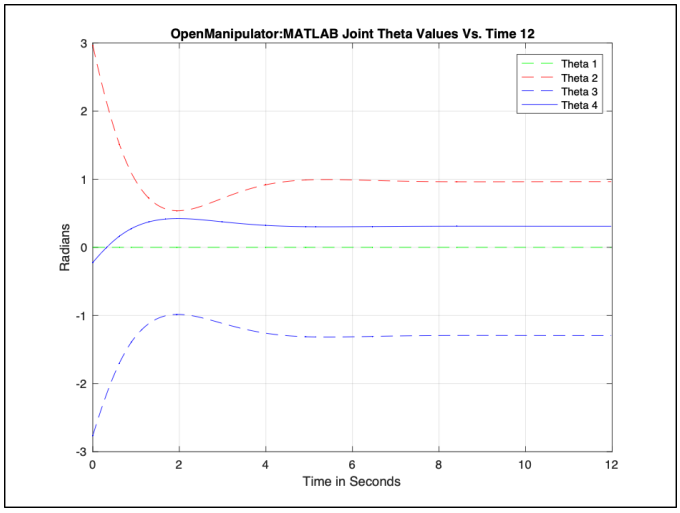

Fig. 16. OpenManipulator MATLAB Theta Values vs Time 1-2

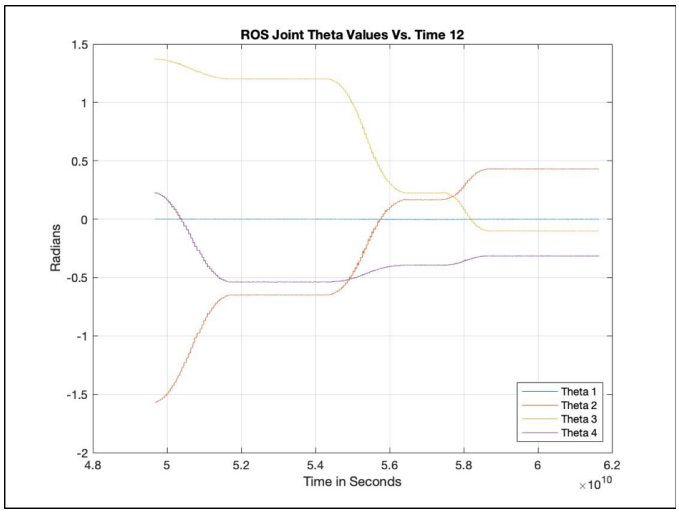

Fig. 17. OpenManipulator ROS Theta Values vs Time 1-2

Joint velocities in MATLAB have smooth trajectories to zero whereas the ROS plots available show significant noise and hard to decipher values. This is most likely due to the real world conditions applied to the ROS model which are causing noise events and disturbing the data being relayed back to the logger.

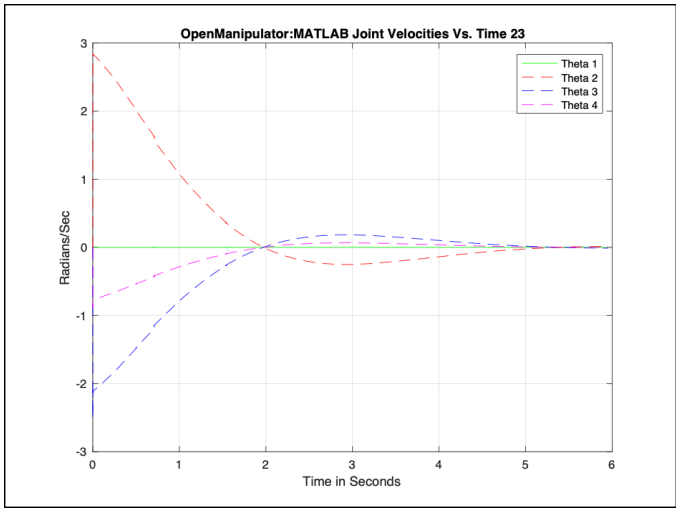

Fig. 18. OpenManipulator MATLAB Theta Velocity Values vs Time 2-3

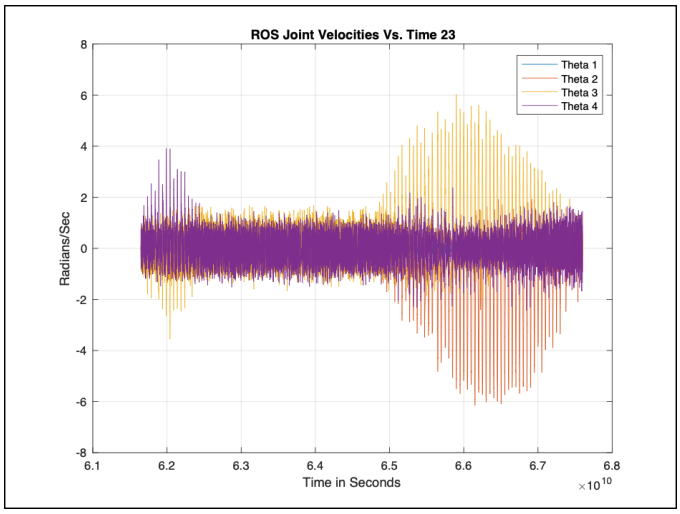

Fig. 19. OpenManipulator ROS Theta Velocity Values vs Time 2-3

The joint torques experienced a similar level of noise on the ROS model whereas the MATLAB model experienced large torque jerk in initial seconds of motion and normalized over time of motion for both path trajectories.

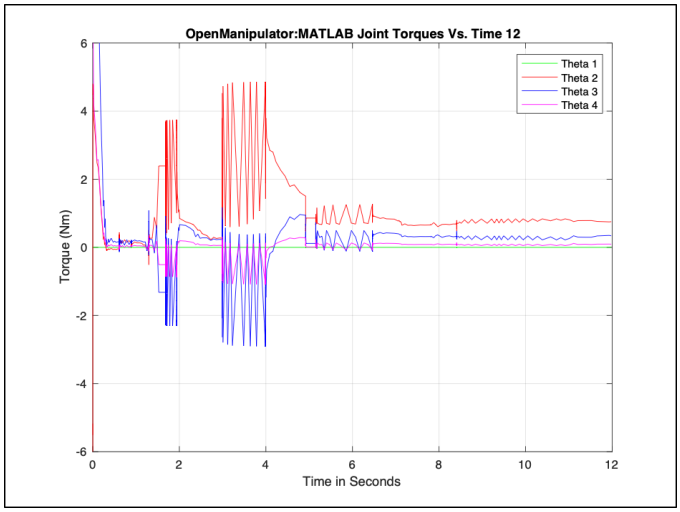

Fig. 20. OpenManipulator MATLAB Torque Values vs Time 1-2

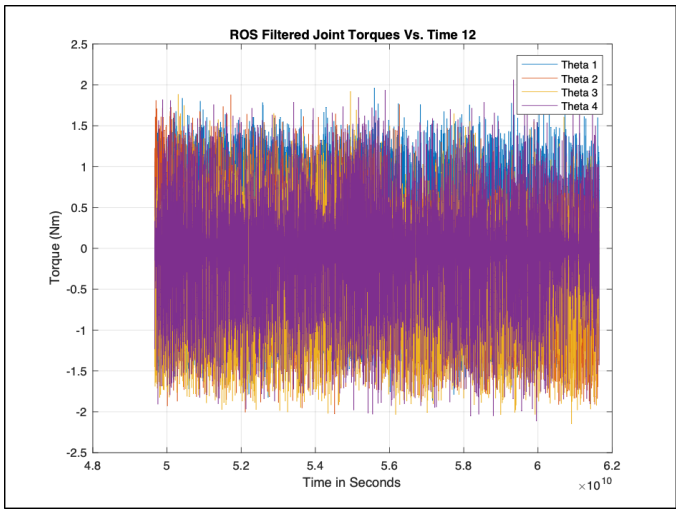

Fig. 21. OpenManipulator ROS Torque Values vs Time 1-2

V. Experimental Results

A. TurtleBot Burger

To verify the performance of the mobile robot simulations, we conducted an experiment using the TurtleBot Burger mobile robot. The Burger is smaller than the Waffle Pi model used in the simulation with the OpenManipulator, but it is still an appropriate candidate as it is a differential-drive robot that utilizes the same single-board computer, motor controller, LIDAR sensor, and dynamixel actuators as the Waffle Pi itself. Since the Burger is a smaller model, it is not a suitable base for the OpenManipulator arm. Therefore, we were unable to gather any experimental data for the OpenManipulator arm.

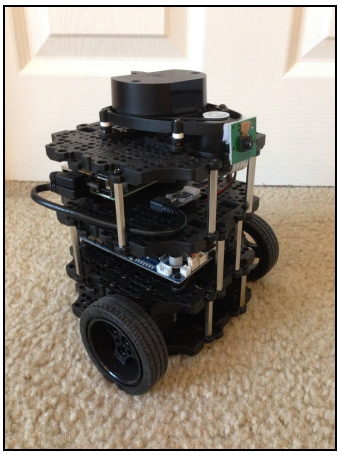

Fig. 13. TurtleBot Burger

In the experimental setup, three target posts were positioned to correspond with the first three locations described in Table 1. Next, the Burger was placed in the environment and a SLAM

node was initialized. The Burger was then manually driven around to collect LIDAR data and build a map of the environment. Once the map was sufficiently complete, the Burger was placed at the first target post. In order to collect position and velocity data, we then executed a Python listener file that subscribed to the odometry topic in ROS. From a remote computer, we then identified target locations in RViz corresponding to the Via Point A and Can locations. The robot moved along a trajectory to both locations and the odometry data can be seen in the graphs listed in the previous section. In general, the experimental data closely matched the ROS simulation, but deviated from the idealized Error Dynamics data.

VI. Conclusion

A. Discussion

In conclusion, we were successfully able to perform path planning for a manipulator arm mounted on a mobile base. We were able to implement analytical control methods in MATLAB, retrieve odometry data from ROS simulations, and log experimental data from the TurtleBot Burger.

Based on our data for the manipulator arm we successfully modeled the coordinate motion of the robot. The ROS simulation does an accurate and repeatable performance of the motions modelled in MATLAB with compensation for real world noise.

Based on our data for the mobile robot, we can conclude that we can rely on the ROS simulations for fairly accurate representations of the real-world performance of differential-drive robot trajectory planning.

Given the data produced by the Error Dynamics model in MATLAB, we can conclude that the path did successfully converge on the desired positions, but the path was unrealistically rectilinear. Furthermore, the reported orientation, linear velocity, and angular velocity throughout the trajectory appeared to be discontinuous and incorrect. Given this information, we can not rely on this Error Dynamics model for accurate prediction of mobile robot trajectory planning.

B. Future Work

In the future, it would be useful to derive and implement more accurate analytical solutions for the mobile robot. The analytical models could then be co-plotted and their performance could be characterized for a variety of tasks, trajectories, and environments.

It would also be helpful to pull the specific gain information from ROS to have a more complete understanding of the control laws that are being implemented in the actual ROS simulation. With this level of detail, we could more accurately compare the predictive ROS simulations with our own analytical models. We would also be able to modulate the gains within ROS and our own models to observe the impact on robot performance.

Ideally, we would also be able to perform an actual experiment with the full OpenManipulator and TurtleBot Waffle Pi. An accurate experimental setup could be constructed to mirror the virtual environmental in ROS. With a highly accurate experimental setup, we could validate our analytical and simulated models with a higher degree of certainty.

Improvements to be made to the designed controllers in MATLAB can also be made. Through our research it has become clear that motion control can also account for a number of other parameters not explored in this project such as “jerk” mitigation, trajectory optimization, and of course ODOA (Object Detection with Obstacle Avoidance).

Acknowledgment

We would like to acknowledge Dr. Mohammad Mahdi Agheli Hajiabadi and Vishnu Vardhan Rudrasamudram for their support and technical guidance throughout the semester. We would also like to thank the online ROS community.

References

[1] Kelly, R., Santibáñez, V. and Loría, A. (2005). Control of Robot Manipulators in Joint Space. Berlin: Springer.

[2] ROBOTIS e-Manual. (2019). ROBOTIS e-Manual. [online] Available at: http://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/ [Accessed 7 Apr. 2019].

[3] Spong, M., Hutchinson, S. and Vidyasagar, M. (2012). Robot Modeling and Control. United States: John Wiley & Sons.

[4] Zak, S. (2003). Systems and Control. New York: Oxford University Press.

[5] Omicsonline.org. (2019). Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework. [online] Available at: https://www.omicsonline.org/open-access/dynamic-modelling-of-differentialdrive-mobile-robots-using-lagrange-and-newtoneuler-methodologies-a-unified-framework-2168-9695.1000107.pdf [Accessed 7 Apr. 2019].

[6] ROBOTIS e-Manual. (2019). ROBOTIS e-Manual. [online] Available at: http://emanual.robotis.com/docs/en/platform/turtlebot3/manipulation/#turtlebot3-with-openmanipulator/ [Accessed 7 Apr. 2019].

[7] Y. Pyo, H. Cho, R. Jung and T. Lim, ROS Robot Programming, 1st ed. Seoul: ROBOTIS Co.,Ltd., 2017.

[8] Mohammed, Amin & Sunar, Mehmet. (2015). Kinematics Modeling of a 4-DOF Robotic Arm. 10.1109/ICCAR.2015.7166008.