Final Project Report

ENPM662 FALL 2020

John DiNofrio & Divyam Garg

December 9th, 2020

Table of Contents

Mobile Robotic Arm Model & Simulation

Introduction & Organization

While robotic technology continues to advance, human workers in assembly lines are becoming a rarer sight to see. Robotic manipulator arms are very versatile and can complete many varieties of tasks. In this project, we will demonstrate a simulated manipulator arm with five degrees of freedom (DOF) capable of performing multiple tasks. This arm will utilize both forward and inverse kinematics to complete actions such as a grabbing and placing objects, drilling, and welding. The arm sits on top of a mobile base, eliminating the issues of singularities and workspace limitations.

The report will start with mathematical equations explaining how the arm will operate. Forward and inverse kinematics require heavy amounts of matrix theory and the use of Denavit–Hartenberg parameters. After the math is explained, the report will focus on the model itself. By creating the model in SolidWorks or altering an already existing model, the 5-DOF manipulator arm will be exported as an URDF to use in ROS. From here, the report will shift to the simulation performed in Gazebo using ROS. Lastly, we will discuss our results in the project.

Motivation

In the current economic demand of everything being mass produced, robots far exceed the production capabilities of humans. The monotonous, repetitive task of assembly can be performed way more efficiently by a robotic arm than a human ever could. Most manipulator arms are very
expensive, so the goal here is to create a small, versatile, and affordable robotic arm capable of doing tasks either professionally or in the household.

Robotic arms offer so many opportunities that would be unfeasible without their help. Whether a task requires extreme precision such as drawing or cutting a design, many

iterations of the task need to be performed, or the task is in a location unsuitable for humans, manipulator arms have the potential to accomplish all of this in one device. As students growing up in a very technological age, we are very intrigued by the idea of having one of these robots as an everyday household item. By working on this project and creating a small and affordable manipulator option, we hope to make that dream a reality.

Robot Description

The 5-DOF manipulator arm will have 5 revolute joints. With its links capable of being 3-D printed, the actuators for its revolute joints will use servo motors. This will allow for articulate control and a closed loop feedback design. Figure (1) shows what the model will be similar to (not actual model, only used for visual demo).

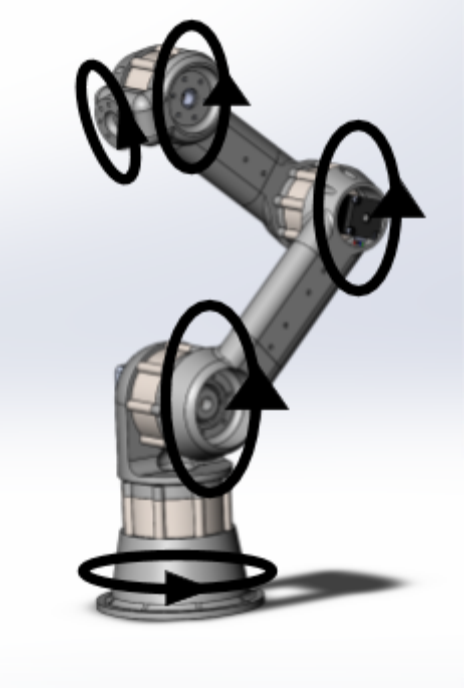

Figure 1. Represents the joint movement with the actuator (Not actual model)

There are three links in total along with an end-effector, allowing for precise maneuverability and dexterity. The base will be revolute around the z-axis and larger than the other portions of the robot. The base needs to be sturdy and heavy to allow for full extension of the arm in case of maximum torque on the base. The diameter will also be wider, about twice the diameter of the links, to ensure stability and full use of the workspace. Composition of the robot will consist of PLA (poly-lactic acid) with a low fill of 15-20 % for arms and high fill of 50% for the base link.

Figures (2) and (3) show the coordinate frames of the robot and the order of the robot from base to end-effector.

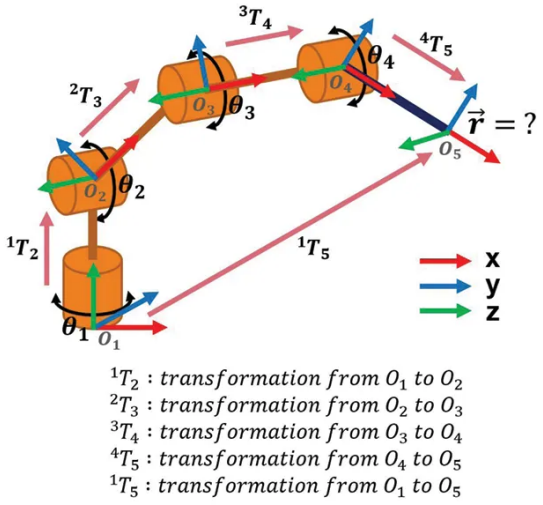

Figure 2. Coordinate frames for a 4-DOF robotic arm

Figure 3. Order of manipulator parts

The mobile base will be made of the same material as the arm with rubber tires. The base is driven by effort velocity controllers for now, but differential drive will be implemented later in future works. The robot arm will sit at the front edge of the mobile base so that the arm can pick up items and drop them off onto the base if needed. Figure (4) shows what the

Table 1. Joints Types

World - Base	Base - Arm1	Arm1 - Arm2	Arm2 - Arm3	Arm3 - end-effector
Fixed Wide/ heavy base to keep robot grounded	Revolute Longest link to support torque forces on motor closest to CG	Revolute Smaller than first link Longest than third link	Revolute Smallest link to reduce torque on base	Free Multi-functionality for variety of tasks

base and arm look like together. The base has a rack-like body to reduce weight. By combining both the body and the arm, this allows the robot to access any point in the simulated world within its height. This eliminates the restrictions of workspace and singularities in the arm.

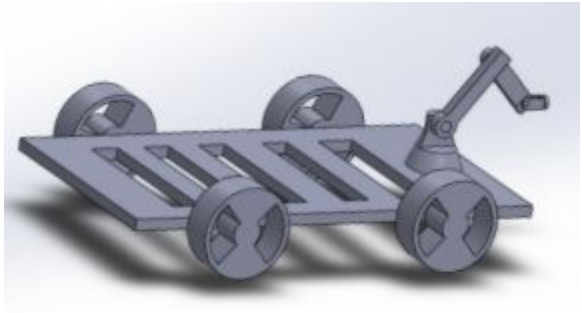

Figure 4. Solidworks base and arm

Robot Appropriateness for Task

When preparing for precision required tasks such as drilling and welding, more dexterity is better. With three links instead of the standard two, the robot will have more maneuverability and a larger workspace. Links two and three are shorter than the first link to appropriate torque forces and create a more balanced robot. This will prevent the arm from falling over when performing tasks that require a wide range of movement like picking and placing objects or following design patterns. The different fill densities helps the link torques even further. The servo motors will have a threshold static limit. The reason for this and the limits is so the motors can always return to the home position after tasks and also to prevent kinematic failures (singularities).

Scope Description

The study will be focused on implementing correct forward and inverse kinematics and apply them to the SOLIDWORKS model robot we make. This study will also be on design as well and implementation because the model is going to be made from scratch. Dynamics of the robot will also be considered so that the actuators can perform without failure in the real world. If time permits, we will test the functionality and success of our arm on a mobile robot.

Scope Appropriateness

As robotic manipulators are in almost every factory and laboratory setting, modeling a similar device from scratch is both challenging and beneficial. Not only do we learn key concepts in modeling and simulation but we also learn the fundamentals of a very practical robot. Forward and inverse kinematics are key concepts in robotics and are not only applicable to manipulator arms. The kinematics and dynamics we will use in modeling this robotic arm can be translated to mobile robots and other items too.

Model Assumptions

The CAD model will be assumed to be a homogenous material with no irregular pores. The physics will be assumed to be correct i.e the

model will stand upright. Servo motors will be assumed to be attached at each joint. Torque calculations will be assumed to align according to each joint.
The ground to base joint will be assumed to be fixed hence no movement between them.
No lag will be assumed between controller and actuator.

Approach to Performing the Work

The first step is setting the coordinates and drawing the model. The DH- parameters of this model will be calculated. Next, we will calculate the inverse kinematics. Once that is done, the model will be designed in SOLIDWORKS 2019. After assembling all the parts, the design will be exported as URDF. During the assembly phase the two separate assemblies of mobile robot and robotic arm were assembled together using parallel mates to get the proper home configuration. Later the mates were deleted and the joints were made revolute. While exporting the mass of inertia along each axis was set up slightly higher than the actual value as later on in the ROS package controllers were supposed to be added with high pid values.

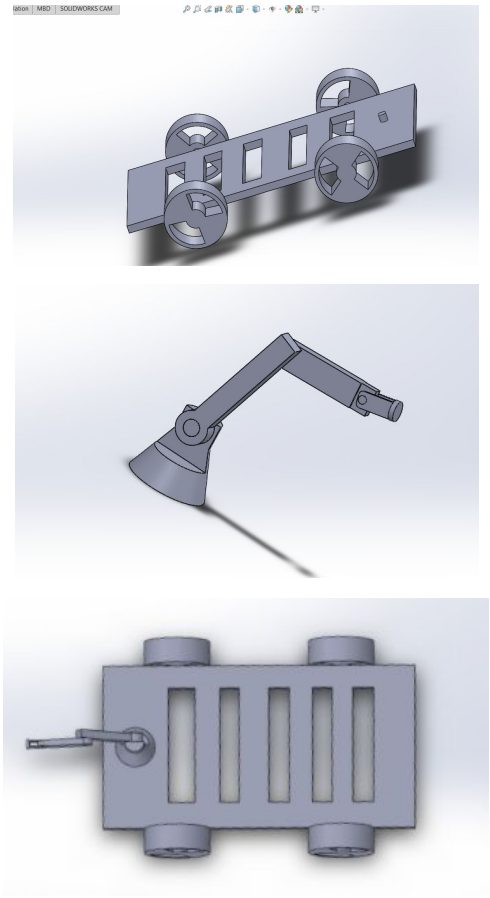

Figure 5. SOLIDWORKS model

Once the assembly was done the package was exported to ROS. The next step was to add all the necessary dependencies while creating a package in the workspace. The URDF file was integrated in a new xacro file with gazebo plugin for effort joint position/velocity controllers. The controller plugin used for this project was

gazebo_ros_control from the file libgazebo_ros_control.so. To execute the controllers for the specific joints, a configuration yaml file was made with the same namespace of the plugin. Here all the joints were defined which were to be controlled. The pid values were tuned using the inbuilt gui of rqt. The launch file was modified to initialise the controller. After the basic setup the model was spawned in a gazebo environment and was visualised in rviz. Finally a python script was written as a ros publisher which takes in the input and calculates the inverse kinematics and outputs the joint/actuator angles. See Appendix (1) for code preview.

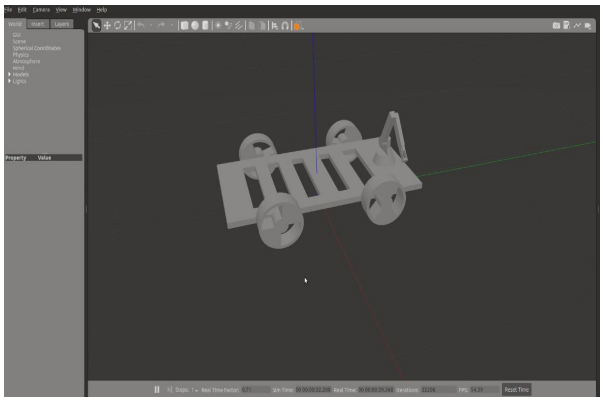

Figure 6. Model simulated in Gazebo

Table 2. Tools for the model and simulation

Tools Used
SolidWorks 2019 with urdf exporter package
References from GRABCAD
Matlab for kinematics calculations and verifications
Rviz for sensors(if any) visualization (ROS)
Gazebo for spawning the model and performing the task. (ROS)
Move-it (open source library for path planning/ inverse kinematics)

Kinematics

Creating the forward and inverse kinematics of the arm is an essential step in creating the simulated model. Figure (7) shows the coordinate frame by frame of the arm.

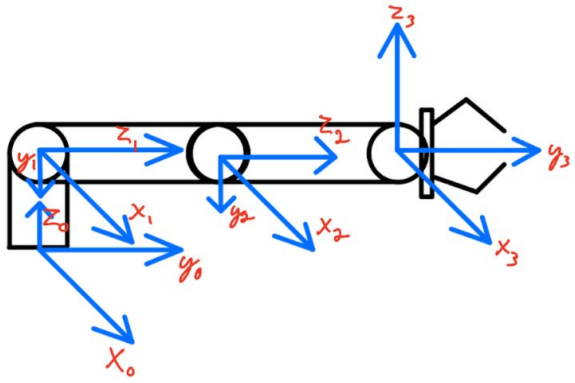

Figure 7. Coordinate frames of the arm

Based on this model the Denavit– Hartenberg (DH) table can be computed. This table gives the reference frames for the model to the links of spatial kinematic chains. By assigning the specific joint angles desired, this table gives us the transformation matrices for the configuration between the frames.

```
T01 =

[ cos(O1), -cos(O2 + pi/2)*sin(O1),  sin(O1)*sin(O2 + pi/2),  0]
[ sin(O1),  cos(O1)*cos(O2 + pi/2), -cos(O1)*sin(O2 + pi/2),  0]
[       0,          sin(O2 + pi/2),          cos(O2 + pi/2), L0]
[       0,                       0,                       0,  1]


T12 =

[ 1,       0,        0,  0]
[ 0, cos(O3), -sin(O3),  0]
[ 0, sin(O3),  cos(O3), L1]
[ 0,       0,        0,  1]


T23 =

[ 1,              0,               0,  0]
[ 0, cos(O4 - pi/2), -sin(O4 - pi/2),  0]
[ 0, sin(O4 - pi/2),  cos(O4 - pi/2), L2]
[ 0,              0,               0,  1]


T03 =

[ cos(O1), sin(O2 - O1 + O3 + O4)/2 - sin(O1 + O2 + O3 + O4)/2,   cos(O2 - O1 + O3 + O4)/2 - cos(O1 + O2 + O3 + O4)/2,  sin(O1)*(L2*cos(O2 + O3) + L1*cos(O2))]
[ sin(O1), cos(O2 - O1 + O3 + O4)/2 + cos(O1 + O2 + O3 + O4)/2, - sin(O1 + O2 + O3 + O4)/2 - sin(O2 - O1 + O3 + O4)/2, -cos(O1)*(L2*cos(O2 + O3) + L1*cos(O2))]
[       0,                                       sin(O2 + O3 + O4),                                       cos(O2 + O3 + O4),       L0 - L2*sin(O2 + O3) - L1*sin(O2)]
[       0,                                                       0,                                                       0,                                      1]
```

Figure 8. Forward Kinematics Eq

Table (3) shows the DH table parameters up to the end-effector since it is replaceable. The table is based on the Figure (7) frames.

Table 3. DH table parameters for FK

	θ_i	d_i	a_i	α_i
Link 1	θ_1^*	l_0	0	$\theta_2^* + \pi/2$
Link 2	0	l_1	0	θ_3^*
Link 3	0	l_2	0	$\theta_4^* - \pi/2$

Using the DH table, the transformation matrices between the links can be found shown in Figure (8). These equations were computed in MATLAB. By adding these to the script used in ROS, the arm can now perform forward kinematics (FK).

The next step is finding the inverse kinematic (IK) equations. There are multiple ways to find these. The forward kinematic equations can be found using the transformation matrices or by using a geometric approach. In this report, the geometric approach will be described.

In IK, the end-effector position is given instead of the joint angles, so the angles must be calculated based on the desired location. Figure (9)

shows the top- down view of the arm with the first joint angle. Solving for θ1 is a simple geometric equation shown in Equation (1).

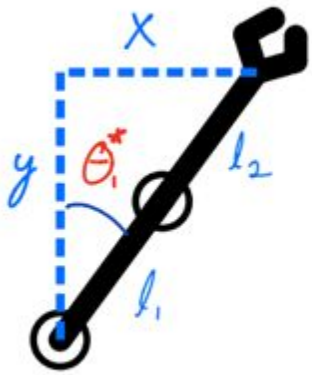

Figure 9. Top-down view of arm

$$\theta_1^* = atan2(x, y) \quad \text{Eq. 1}$$

Computing θ2 and 3 becomes a lot more challenging. Since the end-effector position depends on the hypotenuse of x and y and θ3 needs θ2 as a reference, some interesting formulas arise.

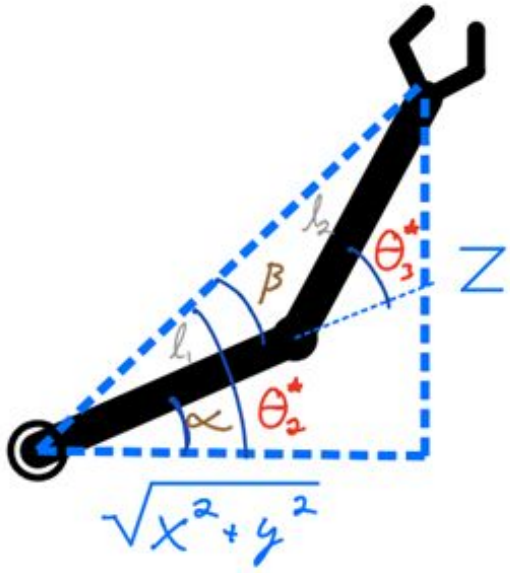

Figure 10. Side view of arm

Figure (10) shows the side view of the robotic arm and the joint angles θ2 and θ3. In order to compute these, temporary variables ∝ and β are created. θ2 is the difference between ∝ and β, and these values can be found with Equations (2-4). The horizontal length of the arm can be found by using the pythagorean theorem of x and y. Then some trigonometry can be used to find the remaining variables.

$$\alpha = atan2(z, \sqrt{x^2 + y^2}) \quad \text{Eq 2.}$$

$$\beta = acos(l_2^2 - l_1^2 - (x^2 + y^2), -2l_2\sqrt{x^2 + y^2}) \quad \text{Eq 3.}$$

$$\theta_2^* = \alpha - \beta \quad \text{Eq 4.}$$

Lastly, finding θ3 requires using θ2 as a reference. Using Equations (5-7) will give the last joint angle θ3.

$$m = \sqrt{x^2 + y^2} - l_1 \cos(\theta_2^*) \quad \text{Eq 5.}$$

$$n = z - l_2 \sin(\theta_2^*) \quad \text{Eq 6.}$$

$$\theta_3^* = atan2(n, m) - \theta_2^* \quad \text{Eq 7.}$$

With θ3 solved, the inverse kinematics are complete and can be converted into the script file and run in simulation.

Validation Plan

In order to validate our model, firstly all the transform coordinate frames were visualized in rviz and were cross-checked with our DH parameters.

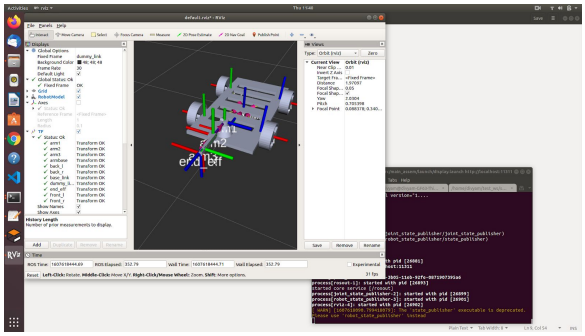

Figure 11. Rviz visualization

The tasks we use to demonstrate our model will be performed by hand-calculations as well. The end-effector location in both the simulation and the hand-drawn calculations should result with the same coordinates and orientation. The entire manipulator arm will be validated by comparing the end-effector coordinates because the arm joints must be in certain locations to create that configuration. If the model is correct and the simulation runs properly, the hand-calculations and ROS coordinate outputs will be the same. For the inverse kinematics after executing the script the points of end effector were compared with the input points which was accurate and hence the model was validated. As one of the test input positions was given as (-0.2, 0.4, 0.2) and the output position of the end effector shown in gazebo image was the same.

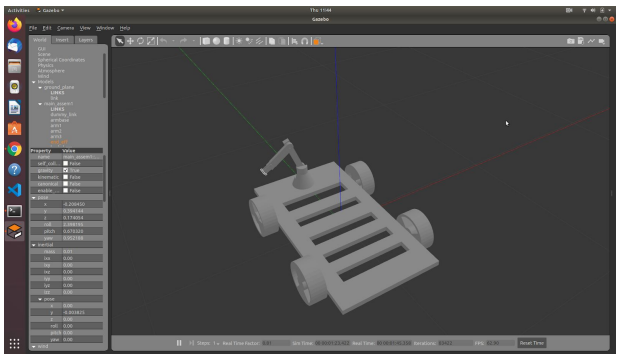

Figure 12. Validation testing

Conclusion

The mobile robotic arm was a success. Though it did not reach our ambitious goals of performing several varieties of complex tasks in simulation, it did meet all the set expectations. The mobile base wasn't added until halfway through the project, and this really gave the design a unique factor to the project. The robot model successfully validated tests proposed in the project with inverse kinematics having a success rate of fifty percent. Even though this is not a good number in the field, fifty percent was around expectations.

Appendix 1

publisher.py

home > divyam > test_ws > src > main_assem > scripts > publisher.py > ...

Set as interpreter

```
#!usr/bin/env/python
import rospy
from std_msgs.msg import Float64
import math
import time

if __name__ == "__main__":
    rospy.init_node ('ik_node')

    pub_armbase = rospy.Publisher('/main_assem/armbase_contro
    pub_arm1 = rospy.Publisher('/main_assem/arm1_controller/c
    pub_arm2 = rospy.Publisher('/main_assem/arm2_controller/c
    pub_arm3 = rospy.Publisher('/main_assem/arm3_controller/c
    pubwheell = rospy.Publisher('/main_assem/back_l_controlle
    pubwheelr = rospy.Publisher('/main_assem/back_r_controlle

    length = 0.16
    height = 0.2
    l1 = 0.17
    l2 = 0.15
    x = -0.2
    y = 0.24 + length
    z = 0 + height
    theta1 = math.atan2(x,y)

    alpha = math.atan2(z,(math.sqrt((x*x) + (y*y))))
    a = (l2*l2 - l1*l1 - (x*x + y*y))/(-2*l2*math.sqrt(x*x + y
    beta = math.atan2(a, math.sqrt(1 - a*a))
    theta2 = alpha - beta


    m = math.sqrt(x*x - y*y) - l1*math.cos(theta2)
    n = z - l2*math.sin(theta2)
    theta3 = math.atan2(n,m) - theta2

    theta4 = -0.1

    r = rospy.Rate(10)
    time_diff = 0
    start_time = time.clock()
    while time_diff <= 10.0:
        pubwheell.publish(100)
        pubwheelr.publish(-100)
        current_time = time.clock()
        time_diff = current_time - start_time
    pubwheell.publish(0)
    pubwheelr.publish(0)
    while not rospy.is_shutdown():
        pub_armbase.publish(theta1)
        pub_arm1.publish(theta2)
        pub_arm2.publish(-theta3)
        pub_arm3.publish(theta4)
        r.sleep()
```