ENPM662

Introduction to Robot Modeling

Final Project Proposal

Mobile Industrial Robot Arm

Authors: Rajan Pande (117306002) & Aditya Jadhav (117389616)

Table of Contents

Introduction

This document provides an in-depth report for our ENPM662 Final Project *Mobile Robot Industrial Arm. The document provides an in-depth proposal for ENPM662 Project 2.* The document contains different sections each with information about certain aspects of the project. The project was concerned with developing a mobile robot arm of industrial standards from scratch. The '***Introduction***' section describes the structure of the report. '***Application***' section describes the applications that this robot can be deployed for a useful function. The sections '***Robot Type***' and '***Dimensions***' detail the in-depth information of the type, structure of the robot, its specifications and physical dimensions, material, and actuators used, and the system specifications. The '***CAD models***' section describes and demonstrates the different stages of the development of the mobile robot arm. The next section '***DH Parameters***' shows the Denavit-Hartenberg parameters of the mobile robot arm which are calculated using the Spong's convention. Next four sections are '***Forward Kinematics & Inverse Kinematics***', '***Forward Kinematics Validation & Inverse Kinematics Validation***' respectively and are the sections about our study of the Forward and Inverse Kinematics of the mobile robot arm and the validation of FK and IK.

Section '***Workspace Study***' describes the workspace of the mobile robot arm containing the maximum reach of the arm with its available joint rotations. '***Assumptions***' section states the assumptions of the project for producing the optimal output out of the developed system. The next section '***Control Method***' is about the control mechanism used for the mobile robot arm which is the Proportional-Integral-Derivative controller. It also outlines the types of ROS controllers that exist and the ones that are used for the arm. The '***Gazebo Visualization***' is mainly about the Gazebo setup of the project containing details about the custom world and its components as well as the entire setup with the arm. '***RViz Visualization***' highlights the laser outputs which are obtained from RViz for the *LaserScan* topic. Sections '***Problems***' and '***Lessons Learned***' represent the problems we faced during several stages of the development process and the solutions/learnings we deployed. '***Conclusion***' section summarizes the project outcomes and concludes the project. '***Future Works***' outlines the ambitious goals that can be achieved building on this project. Finally, the '***References***' section enlists the references we used to acquire knowledge about different aspects of Robot Modeling and our project.

Application

The primary motivation for us to work on this project was the Stretch Robot developed by Boston Dynamics, which is deployed for mobile, automated case handling for more efficient warehouse operations. Stretch is a versatile mobile robot for case handling, designed for easy deployment in existing warehouses. It unloads trucks and builds pallets faster, eliminating the need for new fixed infrastructure. It can be used to automate case handling anywhere in the warehouse with advanced mobility, reducing the need for fixed infrastructure. The Stretch Robot boasts advanced perception which delivers fast and precise case detection. It has a 7 degrees-of-freedom manipulator arm which allows for long reach and large workspace. It also contains embedded sensing and active control which handles variety of package types while operating at high speeds. The lost lucrative feature of the Stretch Robot for us was the mobile base. We draw inspiration from the mobile base which maneuvers in any direction and navigates obstacles and ramps.

As we were working through the different phases of our development process, we had to change the design of the overall robot as well as the robot arm. We faced problems with center of gravity of the system, the limiting torque of each joint (these will be detailed in the '*Problems*' section). Consequently, we had to adapt our robots design to overcome these problems. We decided that we will look for inspiration in a robot which is a crucial part of human history, The Mars Rover Curiosity. Although, possessing abilities far beyond the ones we desired, it was perfect to look up to for us and proceed with our project.

The mobile robot arm applications such as the ones described above are extremely ambitious real-life applications. Our mobile robot arm, however, is not yet capable of such complex applications, but can definitely achieve the core objectives of *Pick 'n' Place* operations in a known and controlled environment.

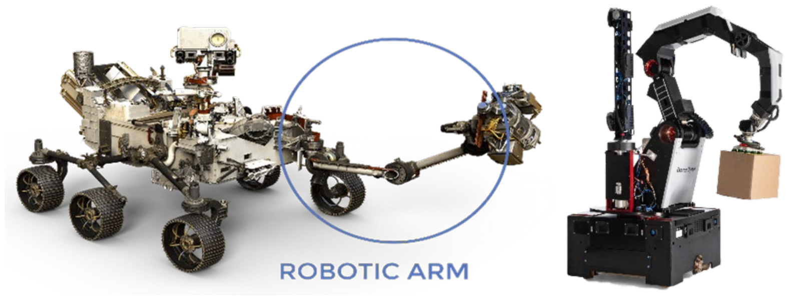


Figure 1: Curiosity *(left)* and Stretch *(right)*

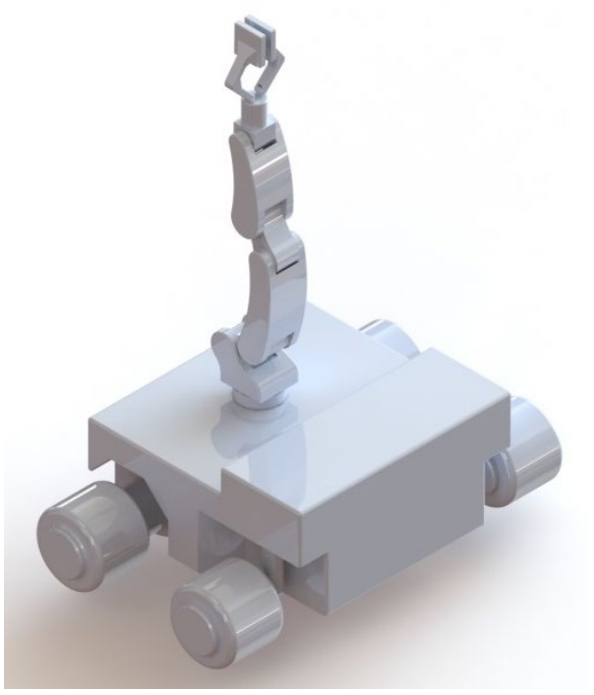
Robot Type

Under the umbrella of a multitude of currently recognized robot types in the world, our mobile robot arm falls under the Industrial Robot Type. An Industrial Robot is a robot system used for manufacturing and its related processes. Industrial Robots are automated, programmable, and capable of movement on three or more axes. Typical applications of robots include welding, painting, assembly, disassembly, pick and place, packaging and labeling, palletizing, product inspection and testing; all accomplished with high endurance, speed, and precision. They can assist in material handling.

The robot arm developed in this project belongs to the Articulated Robots type in Industrial Robots. Articulated Robots are the most common Industrial Robots. They look like a human arm, which is why they are also called robotic arm or manipulator arm. Their articulations with several degrees-of-freedom allow the articulated arms a wide range of movements.

Dimensions

We have developed a 6 DOF manipulator on a 2 DOF mobile base:



Property	**Dimension**
Total Length	2.00 m
Total Width	3.22 m
Total Height	3.91 m
Vertical Reach	3.98 m
Horizontal Reach	2.16 m
Wheelbase	1.25 m
Trackwidth	2.62 m
Wheel Diameter	0.62m
Wheel Thickness	0.50 m
Rotating Arm Length	0.42 m
Extending Arm Length	0.90 m
Picking Arm Length	0.88 m
Jaw Arm Length	0.30 m
Jaw Length	0.54 m

CAD Models

1. Robot Arm 1.0:

The initial design had a simple design which would satisfy our basic fallback requirements. However, the drawback of this design was that it had a smaller workspace.

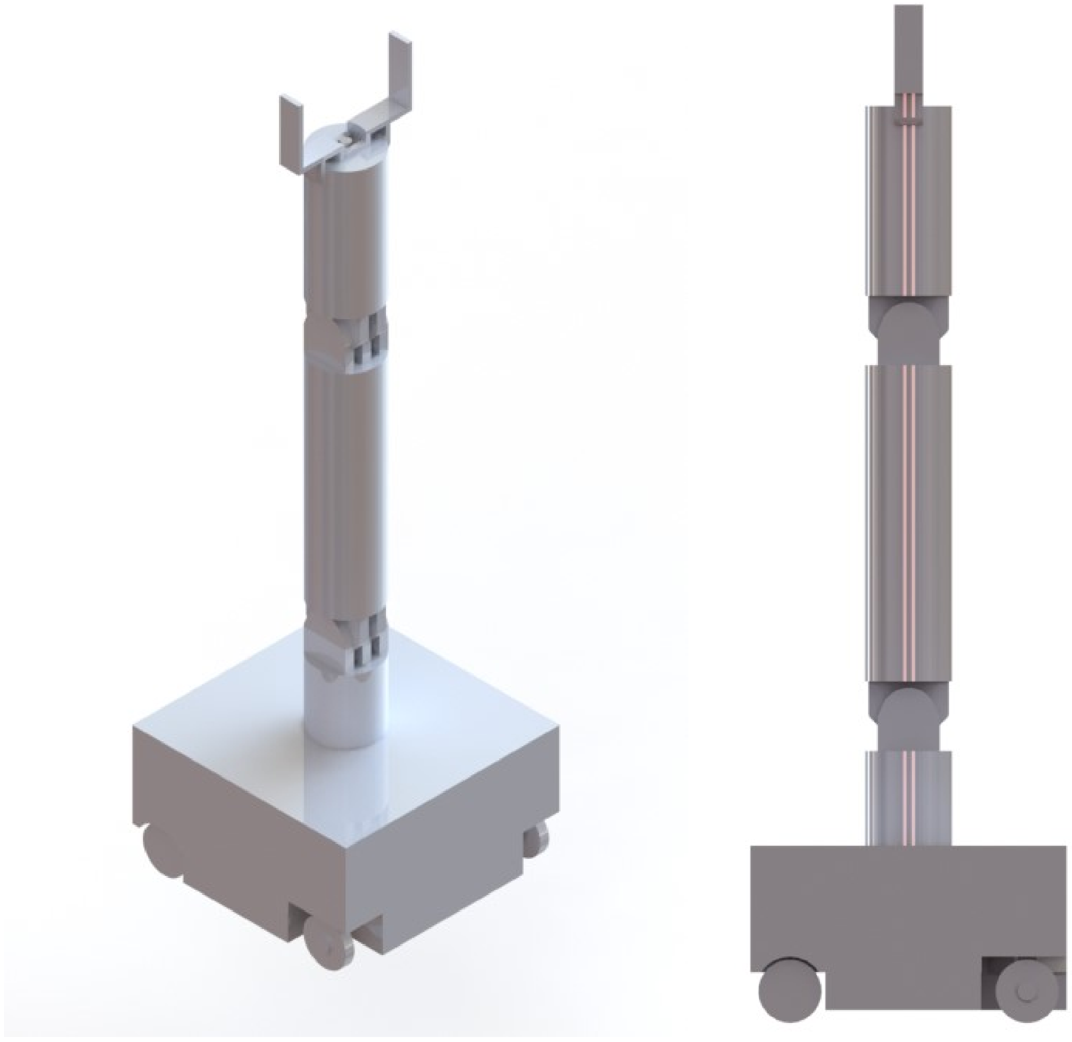


Figure 3: Robot Arm 1.0

2. Robot Arm 2.0:

To increase the workspace, we required to make a few changes to arm design. The major drawback of this design was inability to operate the jaws in ROS.

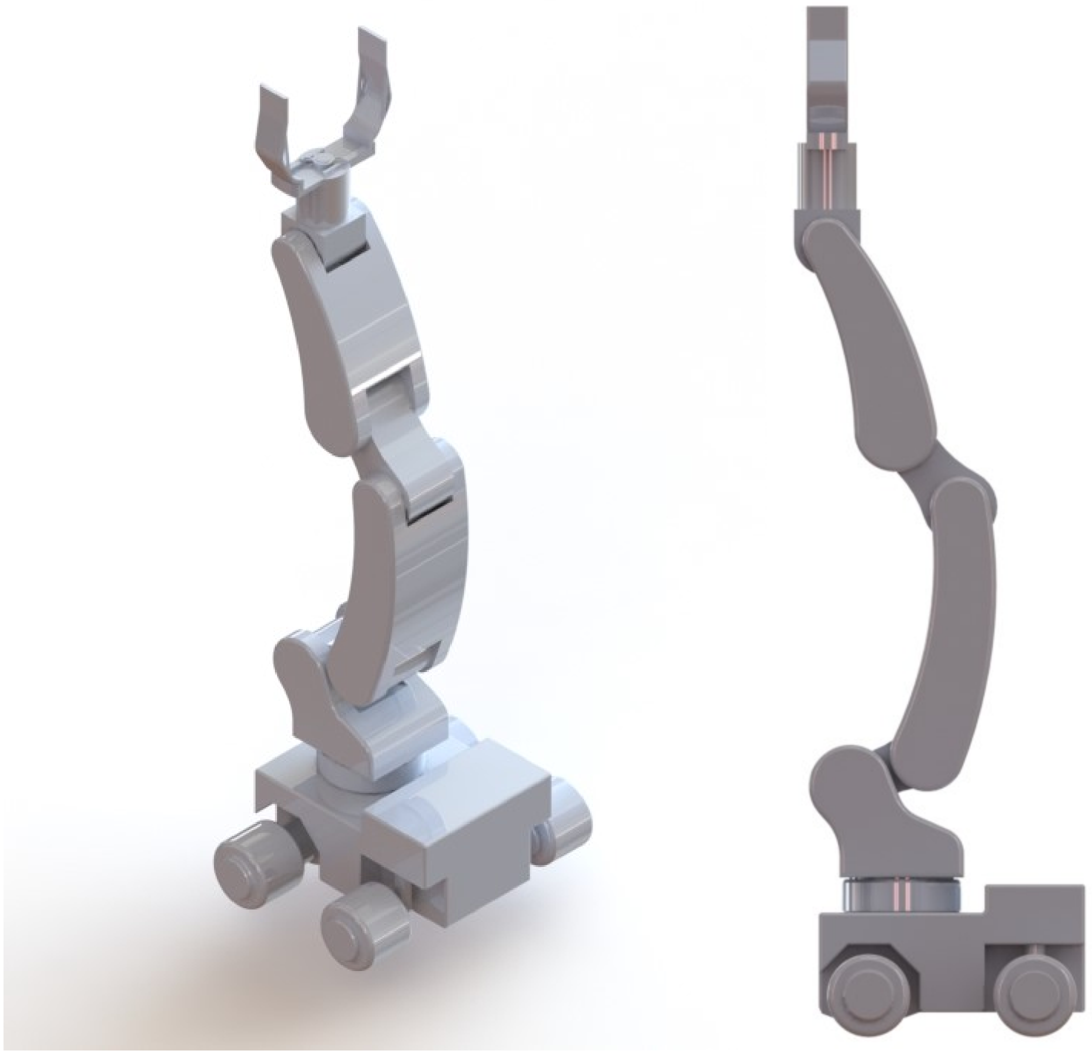


Figure 4: Robot Arm 2.0

3. Robot Arm 3.0:

We thought, maybe the issue with the earlier design was the complicated jaw mechanism. So, we converted it to a simple prismatic joint. However, we weren’t still able to actuate the jaws.

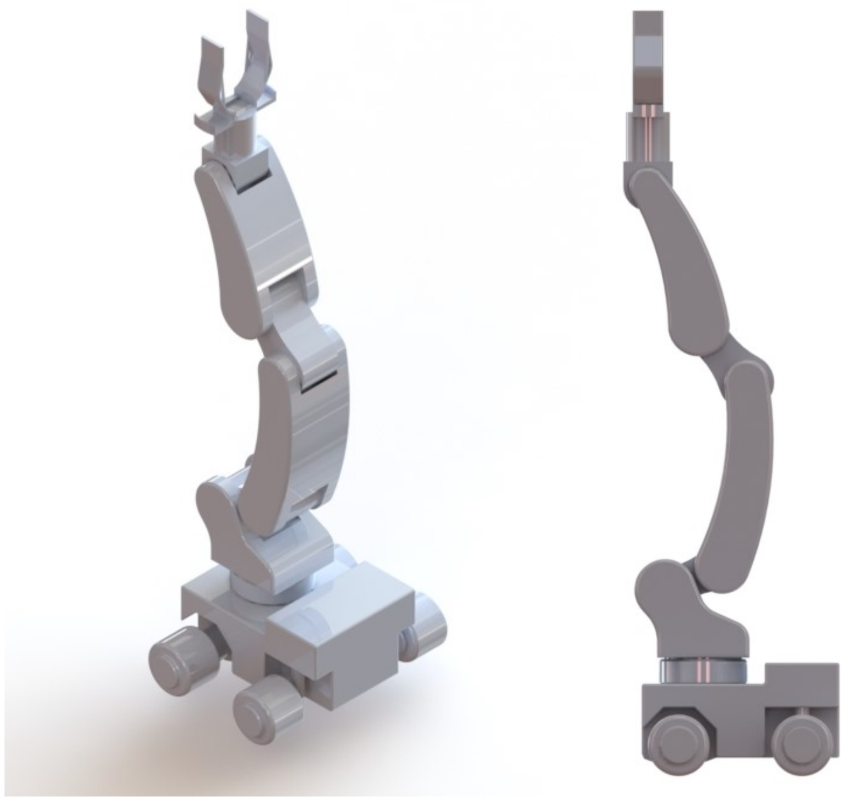


Figure 5: Robot Arm 3.0

4. Robot Arm 4.0:

We finally came up with a solution of designing a simple revolute mechanism for the jaws. Additionally, to increase the stability of the robot, we designed a wider mobile base for the manipulator.

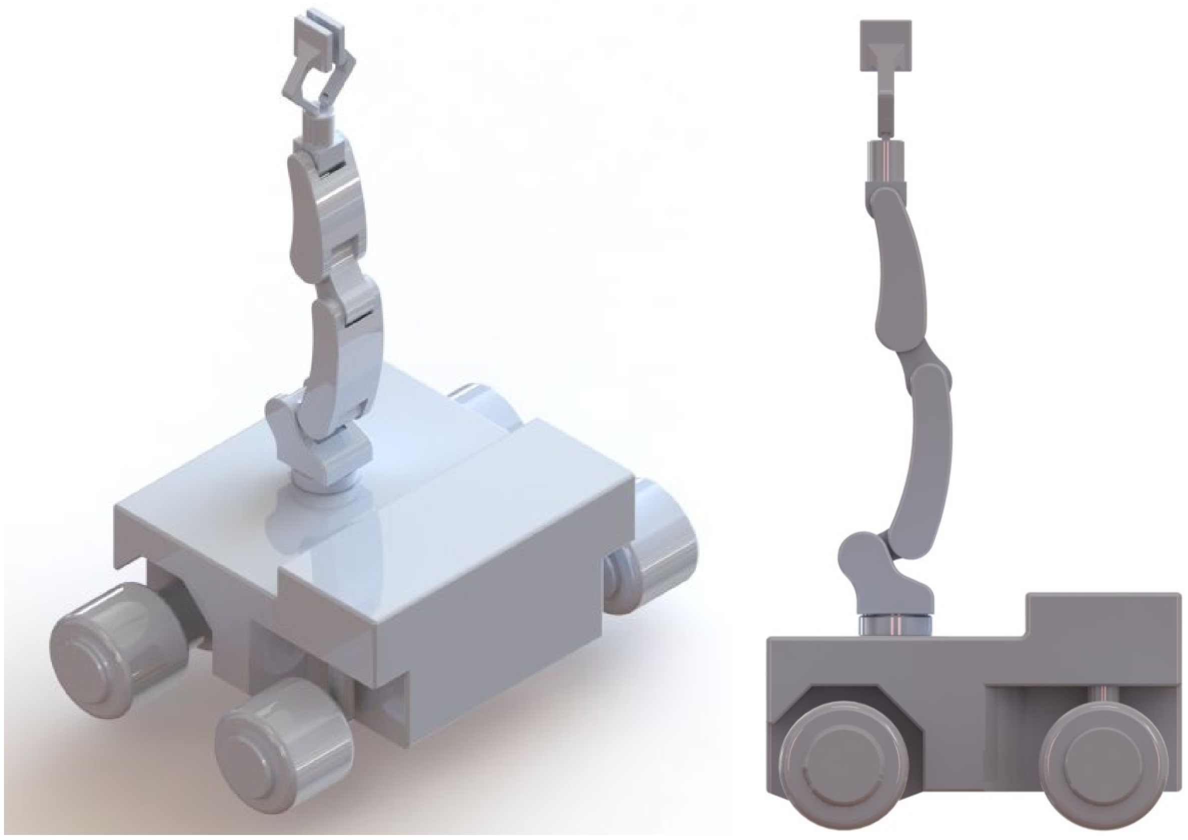


Figure 6: Robot Arm 4.0

DH Parameters

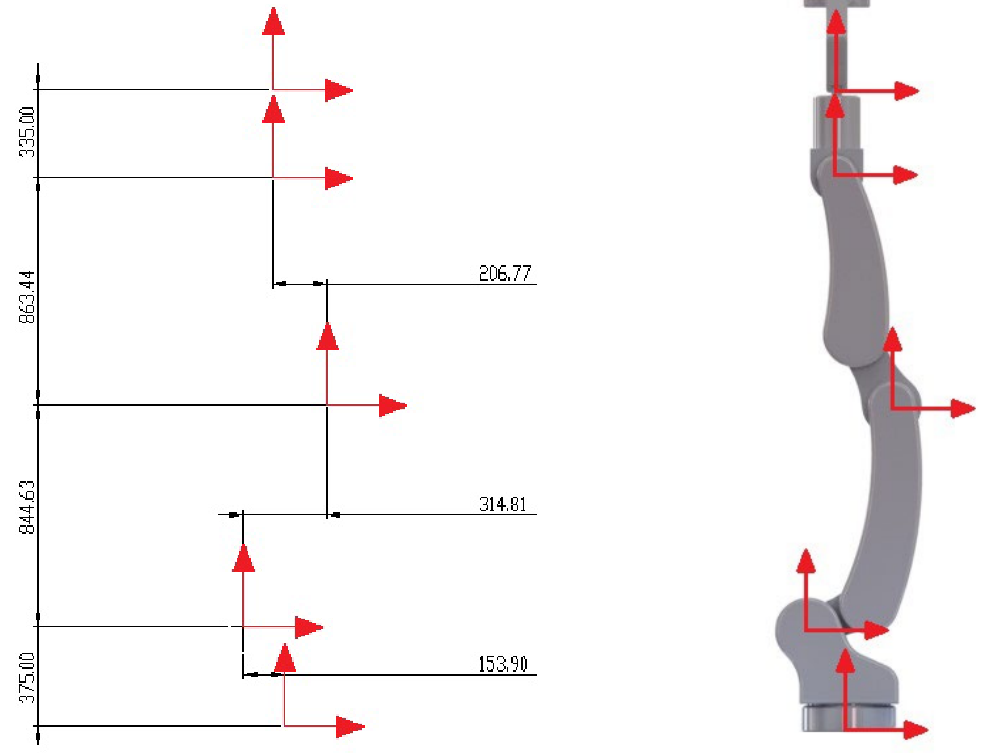


Figure 7: Co-ordinate Frame

Link	a_i	d_i	α_i	d_i
1	153.90	375	π/2	th1*
2	314.81	844.63	0	th2*
3	206.77	863.44	0	th3*
4	0	335	0	th4*

Forward Kinematics and Inverse Kinematics

Jacobian and Transformation Matrix Calculator:

```python
import numpy as np
import sympy as sy
from sympy import nsimplify

def s(x):
    return sy.sin(x)

def c(x):
    return sy.cos(x)

th, al, a, d, th1, th2, th3, th4, Z1, Z2, Z3, Z4, d1, d2, d3, d4, t = sy.symbols(
    'th, al, a, d, th1, th2, th3, th4, Z1, Z2, Z3, Z4, d1, d2, d3, d4, t')

A = sy.Matrix([[c(th), -s(th) * c(al), s(th) * s(al), a * c(th)],
               [s(th), c(th) * c(al), -c(th) * s(al), a * s(th)],
               [0, s(al), c(al), d],
               [0, 0, 0, 1]])

th_list = [th1, th2, th3, th4]
al_list = [pi / 2, 0, 0, 0]
d_list = [375, 844.63, 863.44, 335]
a_list = [153.9, 314.81, 206.77, 0]
init_th = [pi/3, 0, -pi/4, pi]

def calculate_A_matrix(ali, ai, thi, di):
    global A, e     A_transform = nsimplify(A.subs({al: ali, a: ai, th: thi, d: di}),
                                            tolerance=1e-3, rational=True)
    return A_transform

def calculate_T_matrix():
    A1 = calculate_A_matrix(al_list[0], a_list[0], th_list[0], d_list[0])
    A2 = calculate_A_matrix(al_list[1], a_list[1], th_list[1], d_list[1])
    A3 = calculate_A_matrix(al_list[2], a_list[2], th_list[2], d_list[2])
    A4 = calculate_A_matrix(al_list[3], a_list[3], th_list[3], d_list[3])

    T1 = A1
    T2 = (T1 * A2)
    T3 = (T2 * A3)
    T4 = (T3 * A4)
    return T1, T2, T3, T4

T1, T2, T3, T4 = calculate_T_matrix()

def ZMatrix_calculator(T1, T2, T3, T4):
    Z1 = T1[:3, 2]
    Z2 = T2[:3, 2]
    Z3 = T3[:3, 2]
    Z4 = T4[:3, 2]
    ZMatrix = [Z1, Z2, Z3, Z4]
    return ZMatrix

def Jcalculator(joint_angle):
    global T4
    th = [th1, th2, th3, th4]
    Th_values = [joint_angle[0], joint_angle[1], joint_angle[2], joint_angle[3]]

    j = T4[:3, 3]
    ZMatrices = ZMatrix_calculator(T1, T2, T3, T4)

    j1 = j.jacobian(th)

    Z = ZMatrices[0].row_join(ZMatrices[1].row_join(ZMatrices[2].row_join(ZMatrices[3])))

    j2 = j1.col_join(Z)
    j2 = nsimplify(change_values(j2, Th_values), tolerance=1e-3, rational=True)
    j2 = np.array(j2)
    return j2

def change_values(J_updated, theta):
    J_updated = J_updated.subs({th1: theta[0],
                            th2: theta[1], th3: theta[2], th4: theta[3]})
    J_updated = nsimplify(J_updated, tolerance=1e-3, rational=True)
    return J_updated

joint_angles =[0, 0, pi/2, 0]
T4 = nsimplify(T4.subs({th1:joint_angles[0], th2:joint_angles[1], th3:joint_angles[2],
                        th4:joint_angles[3]}), tolerance=1e-5, rational=True)
J = Jcalculator(th_list)
print("Final Transformation Matrix", T4)
print("Jacobian Matrix", J)
```

FK Validation and IK Validation

Libraries required IKPY [5]:

```
pip install ikpy
```

Forward Kinematics Validator:

```python
import ikpy.chain
import numpy as np
def deg2rad(th):
    return th * np.pi / 180


my_chain = ikpy.chain.Chain.from_urdf_file("robot_arm.urdf")
th1 = float(input("Enter rotation (in deg) for Rotating Arm: "))
th2 = float(input("Enter rotation (in deg) for Extending Arm: "))
th3 = float(input("Enter rotation (in deg) for Picking Arm: "))
th4 = float(input("Enter rotation (in deg) for Jaw Arm: "))

target_angle = [0, deg2rad(th1), deg2rad(th2), deg2rad(th3), deg2rad(th4), 0]
position = my_chain.forward_kinematics(target_angle)

print("Input joint angles: (%s, %s, %s, %s, %s, %s)" % (target_angle[0],
                                                        target_angle[1],
                                                        target_angle[2],
                                                        target_angle[3],
                                                        target_angle[4],
                                                        target_angle[5]))

print("X coordinate for end effector: ", position[0, 3])
print("Y coordinate for end effector: ", position[1, 3])
print("Z coordinate for end effector: ", position[2, 3])
```

Inverse Kinematics Validator:

```python
import ikpy.chain
import numpy as np
def rad2deg(th):
    return th * 180 / np.pi


my_chain = ikpy.chain.Chain.from_urdf_file("robot_arm.urdf")
x = float(input("Enter x coordinate: "))
y = float(input("Enter y coordinate: "))
z = float(input("Enter z coordinate: "))
target_position = [x, y, z]
angles = my_chain.inverse_kinematics(target_position)
print("Input position: (%s, %s, %s)" % (target_position[0],
                                        target_position[1],
                                        target_position[2]))
print("Angle for Rotating Arm: %s deg" % rad2deg(angles[1]))
print("Angle for Extending Arm: %s deg" % rad2deg(angles[2]))
print("Angle for Picking Arm: %s deg" % rad2deg(angles[3]))
print("Angle for Jaw Arm: %s deg" % rad2deg(angles[4]))

```

Workspace Study

The kinematic analysis of a manipulator is an important part of the robot study. workspace study is a part of the kinematic analysis of the robot. Working space of robotic arm is the reference point at the end of the robot arm to achieve the set point of space. Working space is discussed from the geometric aspects of the work performance of the robot arm. Analytical work is used to determine the spatial configuration of the robot arm. Working space represents the range of activities, which is a parameter of the ability to work the robot arm kinematics and is also an important indicator. Robot arm and the sizes of workspace according to the design requirements, must reach the required working space, which is reasonable to judge about the structure, operability, and flexibility of the robot arms. Workspace model for robot planning and control information provide important constraints, such as working space was determined according to the mechanical interference between the robot arm and the robot arm motion planning to avoid obstacles.[4]

Horizontal reach of the mobile robot arm is 2.16 m, and the vertical reach is 3.98 m. The workspace is of a parabolic cone shape

Assumptions

Following assumptions are made for our project:

- The environment of the robot (world) obeys gravity, and the surface is continuous and flat.
- The objects are placed on the edge of the platforms so as to be within the workspace of the Robot Arm when it moves closer to them.
- The obstacles are detected by the laser and a soft warning is displayed. It is assumed that the operator is on the lookout for such warnings when the robot arm is visibly close to any obstacle.
- The platform heights are adjusted according to the workspace of the robot arm.
- The end-effector of the robot arm is designed considering the shape of the object. They make grasping of the object drastically convenient.

Control Method

The conventional Proportional-Integral-Derivative method is used as the control mechanism of the mobile robot arm. The PID control method is intended for use where you have a straightforward control problem that you just need to throw a PID loop at. It has one purpose and focuses on doing it well. It has numerous features that ease the task of adding a controller and tuning the control loop. Several examples are provided.

K_p, K_i, K_d: Are the values to be used for proportional, integral, and derivative gains. These values are used by the node unless overridden by dynamic reconfiguration. Defaults are 1.0, 0, and 0 for K_p, K_i and K_d.

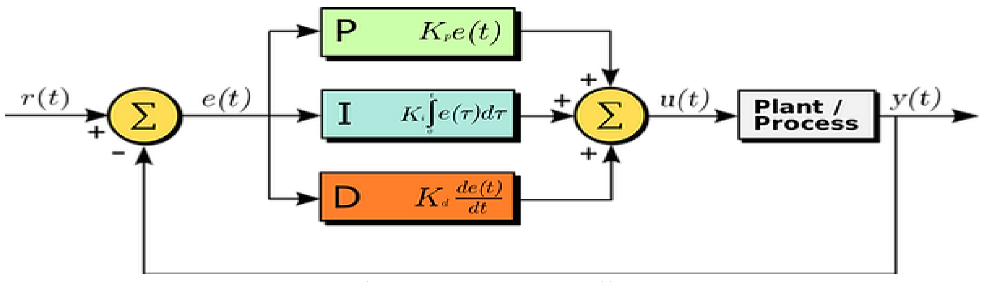


Figure 8: PID Controller

We have tuned our PID values for all the joints present in our system and included in the ROS controller mechanism.

ROS Control is a set of packages that includes controller interface, controller manager, transmissions, hardware interfaces and control toolbox. All these packages together will allow you interact and control the joint actuators of the robot.

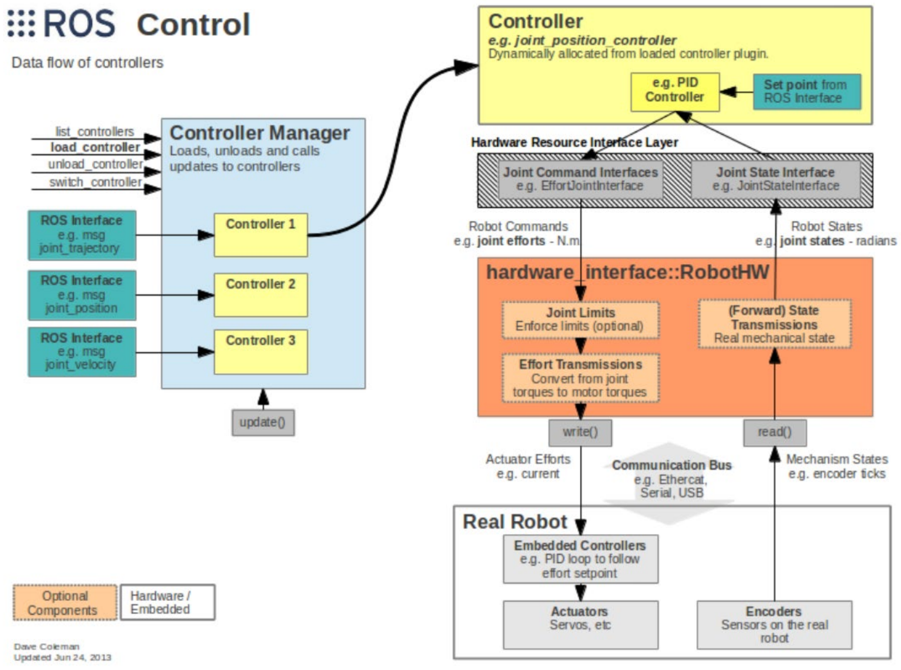

ROS Control
Data flow of controllers
list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()
Controller
e.g. joint_position_controller
Dynamically allocated from loaded controller plugin.
e.g. PID Controller
Set point from ROS Interface
Hardware Resource Interface Layer
Joint Command Interfaces e.g. EffortJointInterface
Joint State Interface e.g. JointStateInterface
Robot Commands e.g. joint efforts - N.m
Robot States e.g. joint states - radians
hardware_interface::RobotHW
Joint Limits Enforce limits (optional)
(Forward) State Transmissions Real mechanical state
Effort Transmissions Convert from joint torques to motor torques
write()
read()
Actuator Efforts e.g. current
Communication Bus e.g. Ethercat, Serial, USB
Mechanism States e.g. encoder ticks
Real Robot
Embedded Controllers e.g. PID loop to follow effort setpoint
Actuators Servos, etc
Encoders Sensors on the real robot
Optional Components
Hardware / Embedded
Dave Coleman
Updated Jun 24, 2013

Figure 9: ROS Control

The basic and most used ROS controllers are:

- *joint_state_controller*
- *effort_controllers*
- *velocity_controllers*
- *position_controllers*
- *joint_trajectory_controllers*

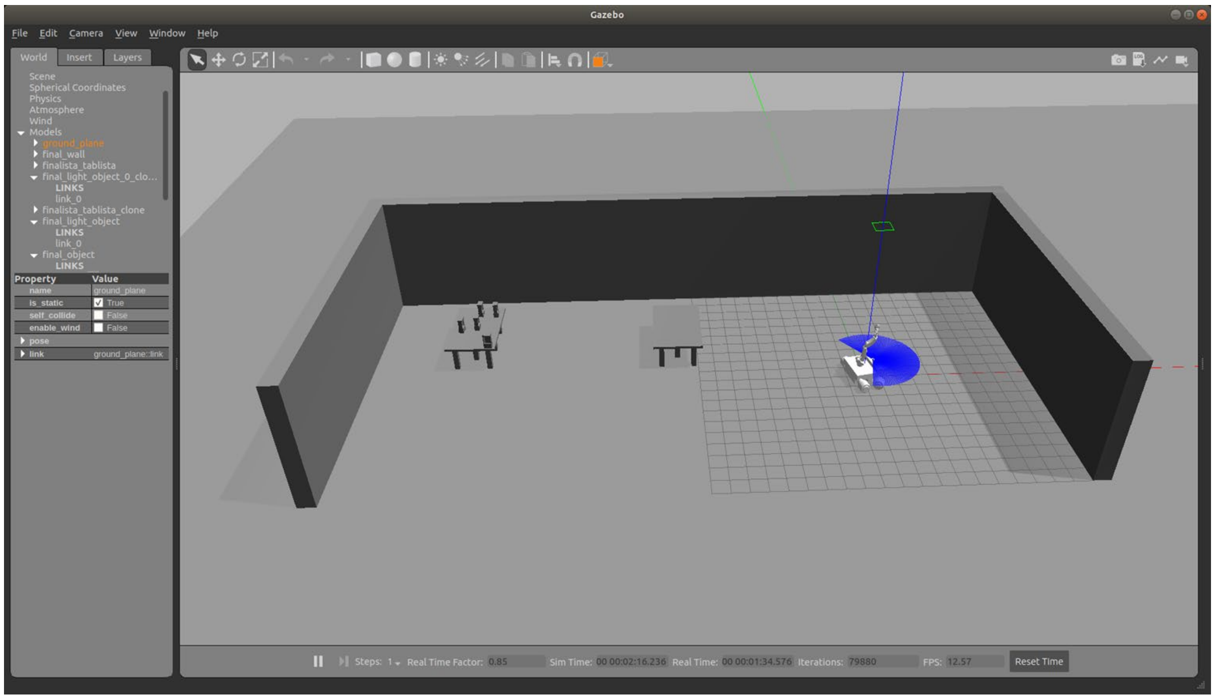
We have used *joint_state_controller*, *effort_controllers*, and *velocity_controllers* in this project.

The *joint_state_controller* reads all joint positions and publishes them on to the topic *"/joint_states"*. This controller does not send any command to the actuators.

The *velocity_controller* is used for the rear-transmission of the base to provide velocity. The *effort_controller* used as *effort_controllers/JointPositionController*, accepts the position [radians (or) meters] value as input. Error (goal position - current position) is mapped to output effort command through a PID loop.

Gazebo Visualization

- We have designed a custom world from the ground up which includes a C-Shaped wall encapsulating the warehouse-like setup.
- The warehouse like setup consists of two tables which are designed with a height that considers the Robot Arm's reach.
- The Objects which are placed on table 1 (close to the north-west corner of the wall) are placed and designed according to the assumptions stated in the 'Assumptions' section.
- The mobile Robot Arm spawns at the Origin (Blue line: Y-Axis) of the custom world.



Property	Value
name	ground_plane
is_static	✓ True
self_collide	False
enable_wind	False
pose	
link	ground_plane::link

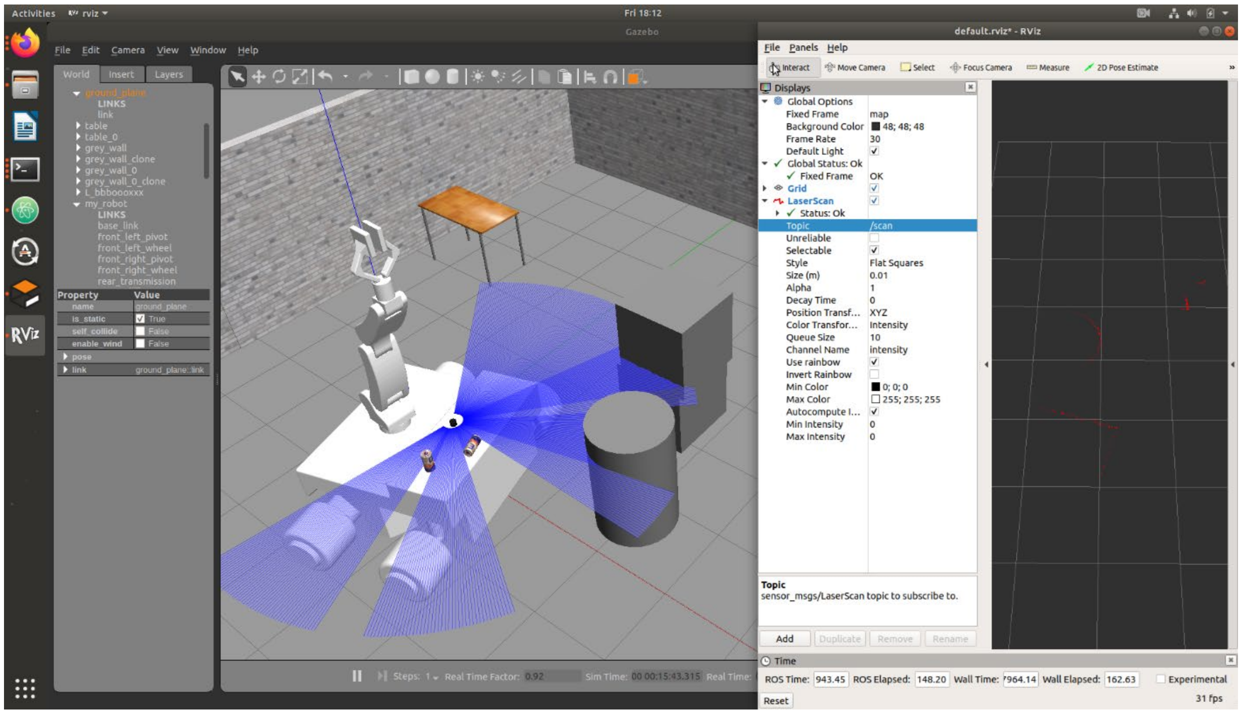
Steps: 1 Real Time Factor: 0.85 Sim Time: 00 00:02:16.236 Real Time: 00 00:01:34.576 Iterations: 79880 FPS: 12.57

Figure 10: Gazebo Initial Setup

➢ Final Gazebo Pick and Place Video: Link

RViz Visualization

A ROS and Gazebo compatible Laser is used for Obstacle Detection of the system. The laser uses *gazebo_ros_head_hokuyo_controller* as the gazebo control plugin. The Tele-Op program uses 0, 30, 90, 120, and 300 degree beams of the laser sensor to detect the obstacles. The Robot Arm can be seen in front of multiple objects and the laser detects all the objects comfortably which is evident from the RViz laser beams shown below.


Activities
rviz
Fri 18:12
Gazebo
File Edit Camera View Window Help
World Insert Layers
default.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate
Displays
Add Duplicate Remove Rename
Time
Reset

ground_plane
LINKS
link
table
table_0
grey_wall
grey_wall_clone
grey_wall_0
grey_wall_0_clone
L_bbboooxxx
my_robot
LINKS
base_link
front_left_pivot
front_left_wheel
front_right_pivot
front_right_wheel
rear_transmission

Property	Value
name	ground_plane
is_static	✓ True
self_collide	False
enable_wind	False
pose	
link	ground_plane::link

Steps: 1 Real Time Factor: 0.92 Sim Time: 00 00:15:43.315 Real Time:

Global Options	
Fixed Frame	map
Background Color	48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
LaserScan	✓
✓ Status: Ok	
Topic	/scan
Unreliable	
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transf...	XYZ
Color Transfor...	Intensity
Queue Size	10
Channel Name	intensity
Use rainbow	✓
Invert Rainbow	
Min Color	0; 0; 0
Max Color	255; 255; 255
Autocompute I...	✓
Min Intensity	0
Max Intensity	0

Topic
sensor_msgs/LaserScan topic to subscribe to.

ROS Time: 943.45 ROS Elapsed: 148.20 Wall Time: '964.14 Wall Elapsed: 162.63 Experimental
31 fps

Figure 11: RViz Visualization

➢ Final RViz Video: Link

Problems

- **Center of Gravity Issue:**

  We encountered a CG issue with the design of the Robot Arm. Initially, the height of the Robot Arm was close to 15 feet, but the wheel-base was not wide enough to support most moments of the links of the Robot Arm. The robot would topple from the slightest of link rotations.

- **Controller and Effort Issue:**

  We faced a controller issue for the arm joints which caused the arm links to collapse into themselves. The limit efforts for each link were not set properly in the initial stages, this combined with the wrong controllers made the arm links get tangled into each other and defy gravity.

- **Effort Limits Issue:**

  Apart from setting the effort values limit, we needed to set the upper and lower limit values with positive or negative signs, which specified the directions of the link limits.

- **Oscillating Object:**

  We have designed a custom world with custom objects inside it. One of the initial objects we designed was defying gravity and swinging like a pendulum when placed on a surface. We got around this by opening Gazebo with admin permissions and then designing.

Lessons Learned

- SolidWorks:
  - Learned to design Prismatic joints for end-effectors.
  - Learned design of manipulator for a wider workspace.
  - Learned URDF export method in SolidWorks.
- ROS:
  - Learned structure and working of a ROS package
  - Learned Publisher – Subscriber architecture
  - Learned how nodes communicate with each other over different topics.
  - Learned the use of *.launch*, *.xacro*, and integration files in a ROS package.
  - Learned about ROS Control Architecture: Controller types to use, when and where to use them.
- Gazebo:
  - Learned how to design a custom world using custom made objects.
  - Learned how to use Model Editor to design custom objects.
  - Learned how to insert, move, rotate, and scale objects inside a world.
- Other:
  - Learned about Forward and Inverse Kinematics of a robot manipulator.
  - Learned about various tags of the URDF and XACRO files.
  - Learned how to interface and work with sensors of a robot.
  - Learned about PID tuning of components of a robot.

Conclusions

In this Project, we started from learning the basic Robot Modeling concepts and worked our way up to some complex concepts used in the field of Robotics. We were able to achieve all of our primary goals designing and implementing a 6 DOF (8 DOF system), lightweight, durable, and end-effector flexible mobile robot arm with at least once sensor for object detection/avoidance and perform pick and place task. In the process of completing this project, we learned many concepts, architectures, and structures in ROS, SolidWorks, Gazebo, FK, and IK.

GitHub: Link

Future Works

We set out developing this project with a set of primary as well as ambitious goals. We were able to achieve the primary set of goals and one of our ambitious goals. We had a goal of designing at least a 4 DOF manipulator which is durable and lightweight. We were able to design a 6 DOF manipulator (8 DOF system) with additional links to increase workspace.

The future work for this project includes making the mobile robot arm completely autonomous given a set of goals. We would like to add extra sensors to help with the autonomy of the robot. To go hand in hand with the autonomous behavior of the robot, we would be aiming to implement a planning algorithm which enables the robot arm to automatically avoid colliding with obstacles.

References

[1] SolidWorks Official Tutorials.

[2] ROS Textbook: A Gentle Introduction to ROS by Jason M. O'Kane.

[3] Stretch Article on Spectrum IEEE.

[4] Kinematics Analysis and Workspace Calculation of a 3-DOF Manipulator, Jianqiang Zhu and Fang Tian 2018 IOP Conf. Ser.: Earth Environ. Sci. 170042166.

[5] IKPY Library