ENPM 662

Introduction to Robot Modeling

Final Project Report

PaintBot

By

Mothish Raj & Jai Sharma

Table of Contents

Introduction

The construction industry is one of the most labor-intensive industries in the world. Common limitations of the industry include rising labor costs and limited economies of scale. To tackle these variable issues, technologists are determined to automate several tasks in the construction industry. It is not just lucrative. Intelligent solutions can substantially mitigate the possibility of construction-related injuries and fatalities.

The specific task that we want to focus on through this project is interior wall painting. The traditional hand painting approach to painting new walls is often slow, costly, inefficient, and hazardous. However, the repetitive nature of the task means that a well-tuned intelligent robot could replace conventional painting teams. There is a need for this transition because it is well known that new house paint contains several toxic chemicals, including ‘volatile organic compounds’ [VOC]. Exposure to paint and its fumes have the potential to irritate the skin, eyes, and throat. Hence, painting walls is an appropriate task to be automated. Additionally, automation can considerably improve application appearance, consistency, and throughput of the paint, which often means savings in raw materials through greater application efficiency.

Robots like this are already in the market. The Okibo robot in Israel and the Canvas Construction robot in the US have already taken up substantial painting projects to date. But there are still a few aspects that can be improved and they have been listed below:

- The robots are bulky and challenging to maneuver in compact spaces.
- The robots have a large turning radius and are hence inflexible in certain scenarios.
- The manipulators used have a limited degree of motion.
- Overall aesthetics

We intend to build a painting robot that is sleek, compact, and has a small footprint. It should have a robust manipulator that can, unassisted, detect walls and paint them evenly. The base of the robot should be able to guide the robot to every corner of the room. Robots like these could potentially also be used for inspection purposes or as collaborative robots

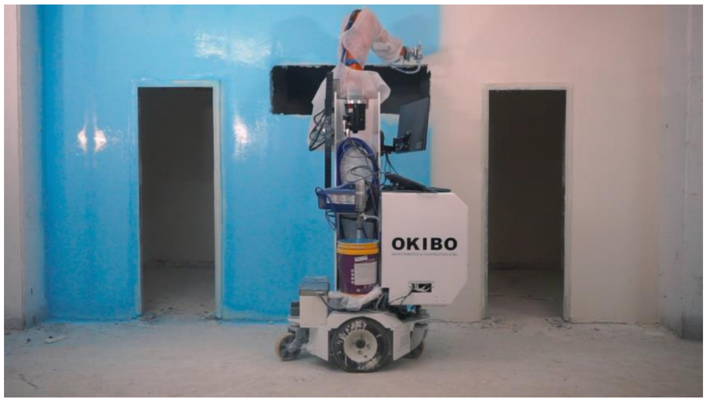


Figure 1: Autonomous Wall Plastering Robot by Okibo

Proposed Solution

We shall name our robot '*PaintBot'*. The PaintBot will consist of a square base (1m x 1m) with 4 omnidirectional wheels. This base will provide us with two degrees of motion and will be responsible for moving the robot around the room. The base will hold all required painting material, electrical components, batteries, and the manipulator's arm. For our design, we believe a KUKA KR16-2 should provide us with 6 degrees of motion. The reach of the robot is critical in this application as the robot must be able to reach all corners of the room. The in-line wrist with a rated payload of 16 kg offers a longer reach of 1,611 mm. The end effector will essentially be a paint nozzle that can uniformly and continuously spray paint at the target wall. The KUKA robot is fitted with a 3-axis in-line wrist allowing the end effector to orient in multiple ways. A rough sketch of our initial design has been shown in Figure 3. We also explored our market benchmarks, Okibo and Canvas, to see what design features could be adopted. A picture of the robot by Canvas Construction is shown in Figure 2.

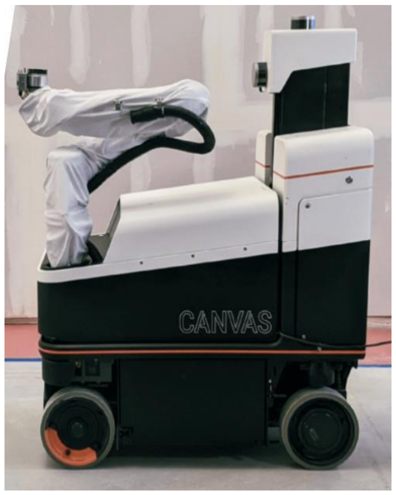


Figure 2: Robot by Canvas Construction

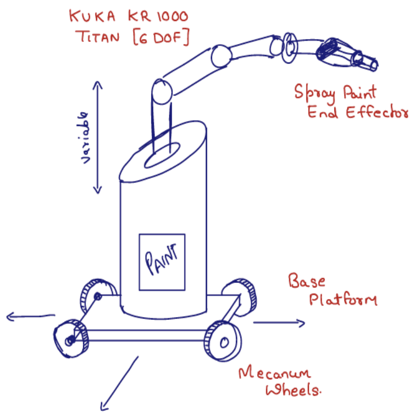


Figure 3: First Iteration of our Design

DOFs and Dimensions

The robot has three components: the chassis, the manipulator, and the end-effector. Details for each component can be found below.

Table 1: manipulator specifications

Model	**KUKA KR16-2**
Reach	1,611 mm
Payload	16 kg
Number of Links	6
Degrees of Freedom	6
Weight	235 kg
Footprint	500 mm x 500 mm
Protection Classification	IP 65
Temperature during operation	+5℃ to +55℃

Table 2: range of movement/speed of each KUKA joint

Axis	**Joint Range**	**Speed with rated payload (16kg)**
Axis 1	± 185°	156 °/s
Axis 2	+ 35°/ - 155°	156 °/s
Axis 3	+ 154°/ - 130°	156 °/s
Axis 4	± 350°	330 °/s
Axis 5	± 130°	330 °/s
Axis 6	± 350°	615 °/s

Table 3: chassis specifications

Footprint	1000 mm x 1000 mm
Height to base	300 mm
Wheels	4
Movement	Omnidirectional

Table 4: end-effector specifications

Type	flat paint nozzle
Cylindrical radius	75 mm
Length	250 mm

CAD Models & URDF

The CAD model for the KUKA KR16-2 was available online. Both the individual part files and the assembly file were downloaded for the purposes of the project. The paint nozzle was designed to replicate what standard paint nozzles usually look like. The flat mouth of the nozzle allows for an even and wide release of paint. For the chassis, since the intended motion was omnidirectional, we decided to make the base a square in shape. The wheels were attached to the chamfered corners of the chassis. The indentation on the top surface of the chassis mainly facilitates in generating the mate between the chassis and the manipulator base.

Our initial assembly consisted of two subassemblies: the manipulator with the nozzle and the chassis with the wheels. This was not a good approach as we encountered issues while exporting URDF files. It seemed like the system had difficulties recognizing the individual joints between the subassemblies. Hence, a new approach was taken. We made the entire robot assembly from scratch. We first imported the chassis, followed by the four wheels, using only one assembly file. Next, we imported and mated each joint of the manipulator, going from link 6 to link one and eventually the end-effector. The URDF file generated responded much better in the Gazebo and the Rviz simulations. The CAD models for individual subassemblies and the final assembly can be found in figures 4,5,6 and 7.

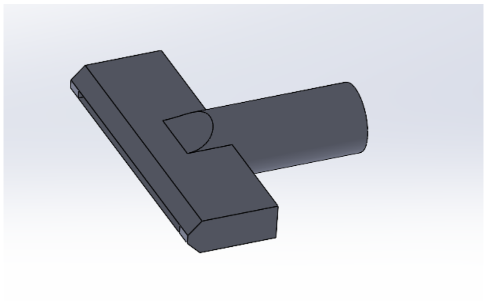


Figure 4: Paint-Nozzle CAD Model

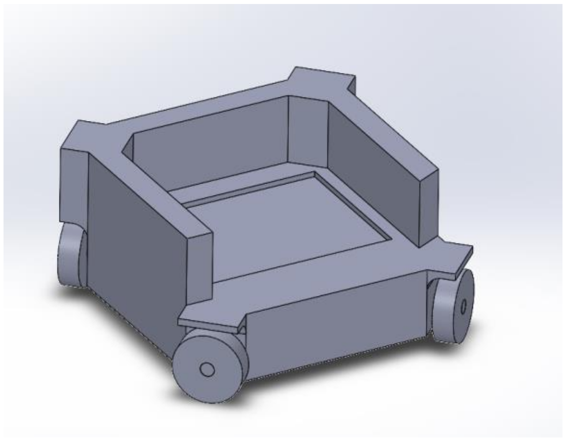


Figure 5: Chassis CAD Model

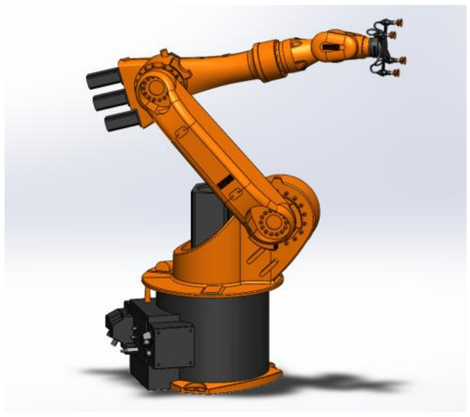


Figure 6: KUKA KR16-2 CAD Model, as downloaded

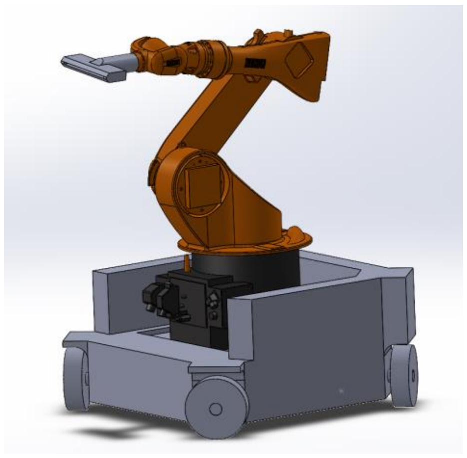


Figure 7: Full Assembly CAD Model

DH Parameters

As mentioned above, the chassis shall be operated by an operator. Hence, the kinematics and dynamics calculations were only carried out for the manipulator. All dimensions for the KUKA KR16-2 manipulator are readily available through online literature. A snippet of this information can be found in Appendix A. Accordingly, coordinate frames were built, as can be seen in figure 8. Next, the Denavit–Hartenberg table was created. Spong's convention was used for these steps.

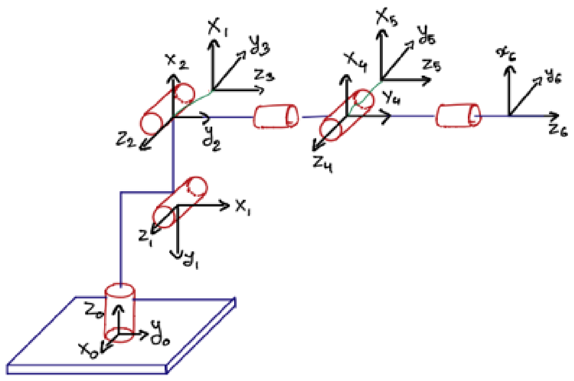
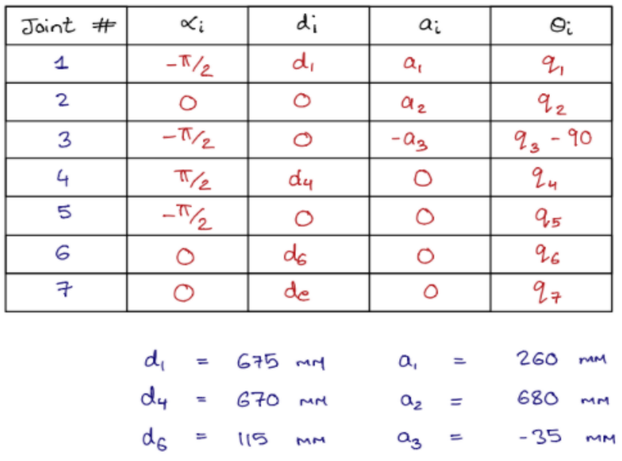


Figure 8: Coordinate Frames



Joint #	α_i	d_i	a_i	θ_i
1	$-\pi/2$	d_1	a_1	q_1
2	0	0	a_2	q_2
3	$-\pi/2$	0	$-a_3$	$q_3 - 90$
4	$\pi/2$	d_4	0	q_4
5	$-\pi/2$	0	0	q_5
6	0	d_6	0	q_6
7	0	d_e	0	q_7

$d_1 = 675$ mm $\quad a_1 = 260$ mm
$d_4 = 670$ mm $\quad a_2 = 680$ mm
$d_6 = 115$ mm $\quad a_3 = -35$ mm

Figure 9: Denavit–Hartenberg table

Forward/Inverse Kinematics

The forward and inverse kinematics was derived for the KUKA manipulator. The program was written in Python 3 using Google Colab, a web-based IDE for Python.

It was important to identify and understand what trajectory we want the manipulator to follow. In our case, we decided to replicate the action of painting by simply making the end-effector move in a linear, up and down motion along a flat wall.

The robot is tasked with making a 100 mm line on the yz- plane. the end effector. The robot's rest position is as shown in figure 13. The end effector's starting position is set at 1320 mm above the KUKA KR16-2 base. Snippets of the code can be found in Appendix B. The ipynb file will also be attached to the submission.

$$\begin{bmatrix} 0.0 & -1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 1295.0 \\ 1.0 & 0.0 & 0.0 & 1320.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Figure 10: transformation matrix T_{06} on firsts iteration (rest position)

$$\begin{bmatrix} 35.0 & 35.0 & 35.0 & 0.0 & 0.0 & 0.0 \\ 940.0 & 680.0 & 0.0 & 0.0 & 365.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 & 1.0 & 0.0 & 1.0 \end{bmatrix}$$

Figure 11: Jacobian Matrix $J_{\ n=1}$ on firsts iteration (rest position)

Validation

The validation was done using the python code. T_{06} matrix can be validated by the frame coordinates from Figure 8 and the dimensions given in Figure 13. The intended output and the resultant output are als compared at the bottom of the submitted python file. The result can also be seen below in figure 12.

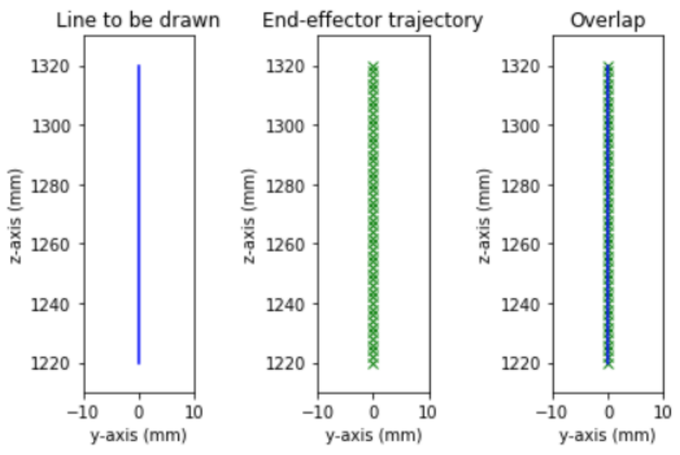


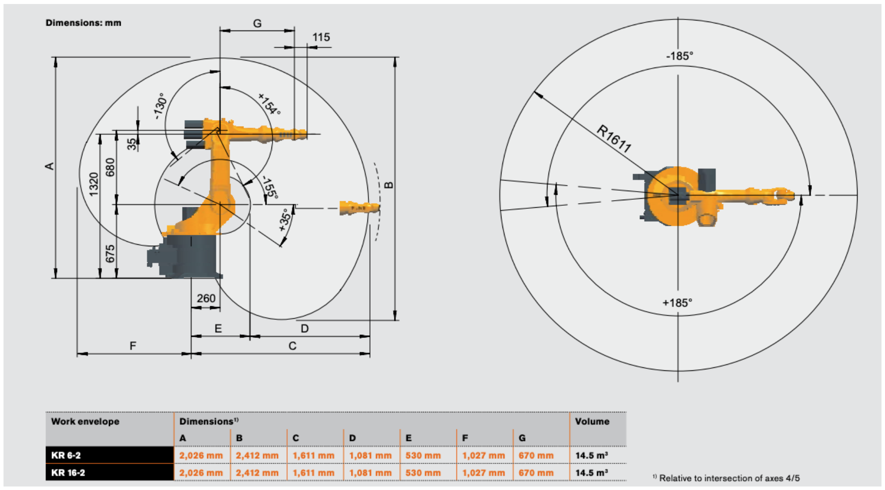
Figure 12: code-validation

The expected output is a 100 mm line, starting at 1320 mm above ground on a yz-plane. The end effector velocities are generated using inverse kinematics. This data is multiplied by Jacobian to attain an end-effector trajectory. The results can be seen in the middle plot, in green. Figure 12 proves that the expected trajectory and the resultant trajectory are the same. Hence, the forward-inverse program is correct. We can rely on this code when carrying out any further trajectory planning tasks for the KUKA KR16 robot.

Workspace Study

Our robot is essentially a mobile manipulator. The manipulator by itself, would have a limited workspace. However, mounting it on a mobile platform, like our chassis, offers unlimited workspace to the manipulator arm. The chassis can go anywhere around a room and hence, the end effector can access all points from that specific stationary position of the base.

The workspace for the 6-DOF manipulator by itself can be seen in figure 13:



Work envelope	Dimensions[1]							Volume
	A	B	C	D	E	F	G	
KR 6-2	2,026 mm	2,412 mm	1,611 mm	1,081 mm	530 mm	1,027 mm	670 mm	14.5 m³
KR 16-2	2,026 mm	2,412 mm	1,611 mm	1,081 mm	530 mm	1,027 mm	670 mm	14.5 m³

[1] Relative to intersection of axes 4/5

Figure 13: Workspace for KUKA KR16-2

From the figure above, one can deduce that link 4 and link 3 are the fairly important with regards to the overall reach of the manipulator. Figure 14 shows how the reach of the manipulator is affected when joint 4, that controls link 4 is at its extreme joint positions: + 35° and - 155°.

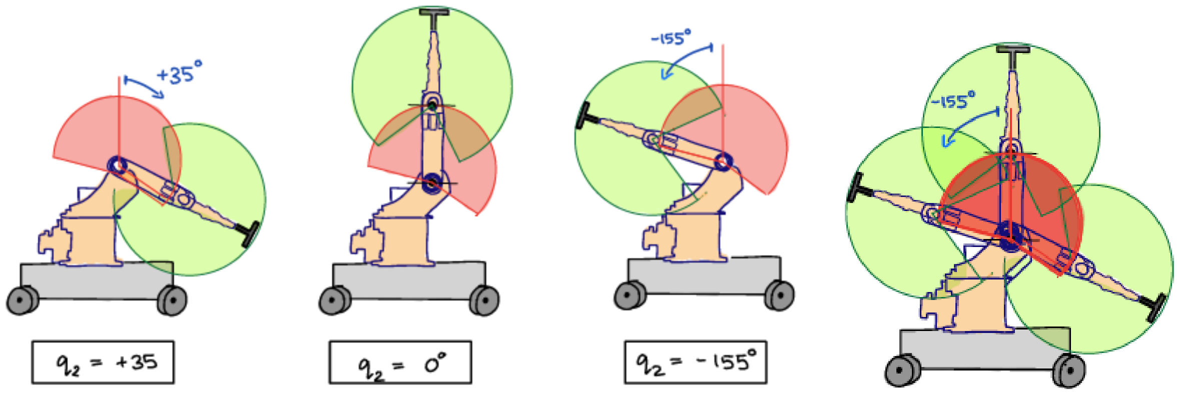


Figure 14: Workspace illustration at extreme joint 4 configurations, and an overlap image

The planar workspace in figure 14 can be used to validate the workspace shown in figure 13. The general shape will be the same. Only difference being that the outer radius of the workspace extends from 1611 mm to 1861 mm. The internal radius depends on several factors like the link lengths and joint ranges for the spherical wrists. The planar workspace can be visualized in 3d using advanced software. Joint 1, with a ± 185° range, can be used to attain the 3D workspace, which could be expected to be roughly hemispherical in shape.

There is a workspace limitation that has not been accounted for in our proposed solution. The mobile platform is fixed in the z-direction. So, there is a limit on the height that the manipulator can reach. In our case, the highest point that the end effector can reach is 2161 mm above ground, as shown below, in figure 14.

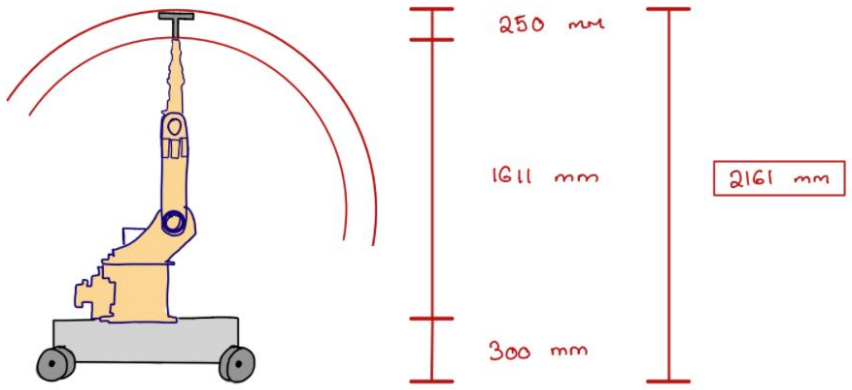


Figure 15: height-limit on the robotic system

Standard room heights can be as much as 2500 mm. Hence, it is important to address this problem. Adding a platform that can move vertically up and down on the chassis is a potential solution. A hydraulic scissor lift can be installed quite easily. With this design feature, one can be sure that the robot can access and paint all sections of a wall in a standard room, offering a larger workspace.

Assumptions

The project has the following assumptions:

1. Any frictional or external force is not considered.
2. The Omni-wheel joints are continuous, and the robotic arm joint with the frame is fixed
3. A paint spray nozzle is being attached to the end of the robot and they will be operated autonomously
4. The paint spray is not mechanically operated; thus, it does not require any mechanical input from the robot.
5. Collision with the external obstacles is considered, but the internal collision is not considered.
6. Optimal path for arm movement is not considered in the robot for the scope of the project.
7. The robot is not considered to be autonomous. A pre-programmed motion will be fed to the robot for simulation purposes.
8. The mass of each link is reduced to make it almost negligible.
9. For Simulations, the mass of each manipulator link is considered to be negligible. The mass of the chassis is assumed to be higher than it actually is.

Control Strategy

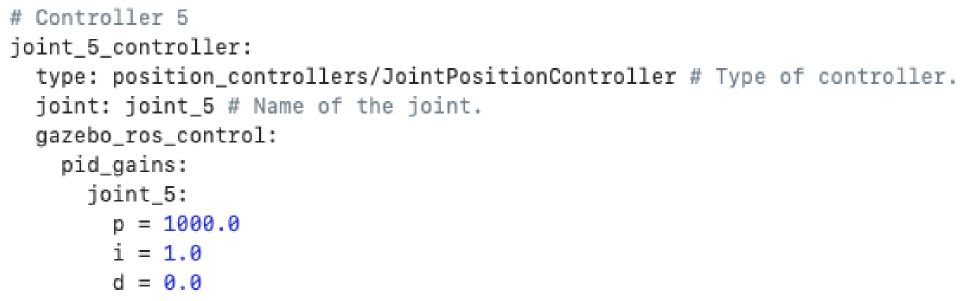
We need a total of 10 controllers to operate the robot system. Four of them utilize the motors in the chassis. The remaining 6 control the revolute joints on the manipulator's arm. The chassis and the manipulator will have two separate control systems. They will work independently, and each can be controlled remotely by an operator.

The KUKA KR robots can be purchased as ‘Ready Packs.’ This means that the robot will come with a controller, an operator control element and suitable software. Being a preconfigured package solution, we would not need to come up with an elaborate control strategy. Instead, the operator simply needs to learn the functions to operate the manipulator, as per the requirement.

The KR CYBERTECH Ready Pack comes with the following equipment:

- robot controller KR C4 / RP SC5.1 X11 X51 X66 IO
- KUKA smartPAD ergonomic control panel with capacitive touch display
- expandable software KUKA.SystemSoftware
- freely configurable application software
- connecting cable set
- accessories

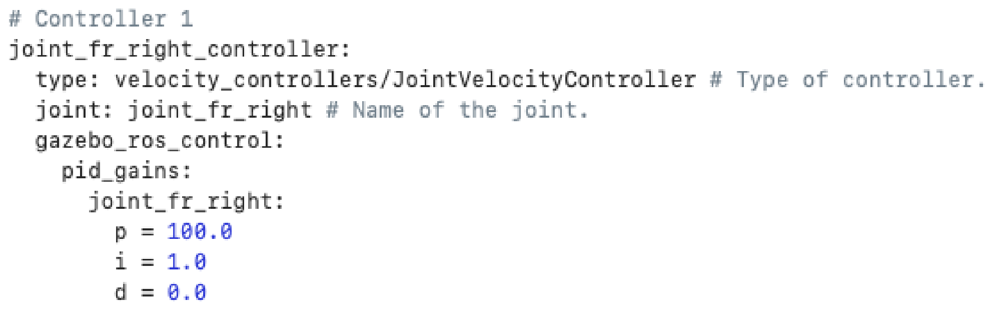
However, for our simulations, we used position controllers with a tuned PI control strategy to operate the Manipulator's joints. The input for the control was the joint angle, in radians. A sample code for joint 5 controller from the yaml file can be found in figure 15:



```
# Controller 5
joint_5_controller:
  type: position_controllers/JointPositionController # Type of controller.
  joint: joint_5 # Name of the joint.
  gazebo_ros_control:
    pid_gains:
      joint_5:
        p = 1000.0
        i = 1.0
        d = 0.0
```

Figure 16: Manipulator Arm Controller code snippet

For the omnidirectional Chassis, we initially intended on using a PID controller. However, we recognized that a PI control would be sufficient for our purposes. just as it was for the joint control. Even when tested, with some tuning, the chassis responded well and moved as desired. The controller type for the wheels is velocity control. The input we give to control the motors is velocity, in radians per second. The final PI gains that we used, can be seen in the snippet below from our yaml file.



```
# Controller 1
joint_fr_right_controller:
  type: velocity_controllers/JointVelocityController # Type of controller.
  joint: joint_fr_right # Name of the joint.
  gazebo_ros_control:
    pid_gains:
      joint_fr_right:
        p = 100.0
        i = 1.0
        d = 0.0
```

Figure 17: Chassis Wheel Controller code snippet

Gazebo/RViz Simulation

The URDF was exported and saved under the name ‘paintbot_29.’ This became the package name as well. The transmission blocks and the gazebo plug in had to be added to the URDF file. We built a yaml file called 12345.yaml and added it to the config folder. This contained all information regarding the controllers. We wanted to simulate the workings of a robot in a world that had walls. Hence, we incorporated ‘competition_arena.world’, a map we have previously worked with. To integrate the map, controller, URDF and meshes file, we created a template_launch.launch file. This allowed us to call all required files with a simple command, making the process of testing much more efficient.

Controlling the robot was achieved by implementing publisher scripts. There were two goals for the simulation:

1) show that the manipulator can replicate the action of brushing through a linear up-and-down motion.
2) show that the chassis can work in an omnidirectional manner and that it is fully controllable by us.

The scripts implemented for the project is a simple publisher file to control the robot and provide an optimum painting motion for the robot. Since the motion of the robot is repetitive, a teleop was not included in the aspect of the project and the omni wheel motion is also covered using the publisher method.

The robot was imported as required into both Gazebo World, and RViz. The robot can be seen in its rest position in the two pictures below.

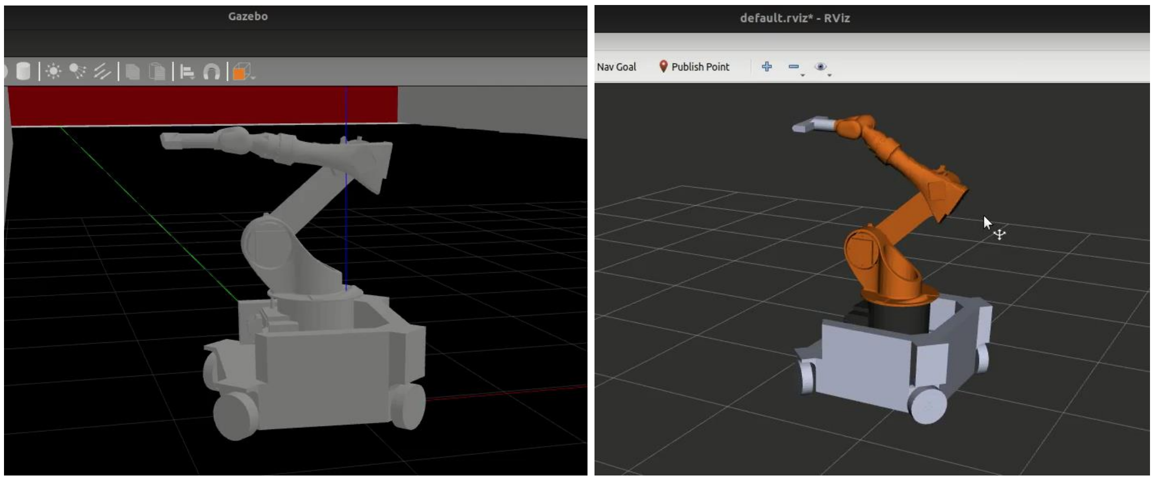


Figure 18: Robot in Gazebo world and Rviz

The Google Drive links attached in Appendix C can be used to access the simulation videos for the robot.

Problems and Lessons Learned

We faced several problems through the course of this project. We will use the help of the simulation videos to go over the problems we faced and how we went about solving them.

In the first video, one can see that the robot links do not stay in place. They fall over, breaking the joint limits we have provided through the URDF file. They also self-collided with each other which was a major cause of concern for us. When the publisher script was executed to make the robot move, the links followed a trajectory that was not optimal, they passed through each other to achieve the programmed task. We eventually realized where we could be going wrong. Below is a list:

- The mass of the manipulator links was too heavy.
- The 'effort' input in the URDF file was too small.
- The URDF file had difficulty working with sub-assemblies.
- Effort Joint controllers is a difficult controller to implement through publisher scripts.

To resolve all these issues, we made several adjustments to our CAD model, URDF files and yaml files. As mentioned previously, the CAD model was made from scratch. Each part was assembled using just one file, not through subassemblies. The masses of the joint links were altered to assure that the links don't fall over when imported into the Gazebo world. The mass of the base link, the chassis, was not altered, acting as a strong foundation for the system. Next, the effort was increased to almost 5000 Nm in some cases. This meant that our controllers associated with the joint can provide a maximum effort of 5000 Nm. We speculate this helped with making the link motions smoother. Finally, we replaced the controllers for each joint from effort_controller to position_controller. The biggest benefit was the ease of writing the publisher script. Instead of estimating linear forces that cause the arm links to move, we use joint angle as input. Hence, passing 0.79 radians for joint 5 for example, will make the joint 5 move by 45 degrees.

Overall, the aforementioned changes resolved all the issues we faced. The robot imported into the Gazebo world was not collapsing and the manipulator arm was moving smoothly as we programmed it to move. The goal of having a omnidirectional chassis was also accomplished.

Future Work

Several improvements can be made to our existing solution, especially simulation. At this point, the robot is controlled independently of the surrounding environment. As a result, the robot moves smoothly and responds it the way we program it to respond. However, the robot can be made smarter by incorporating LIDAR sensors and a robot vision camera. This would help us create more comprehensive codes, ultimately making the robot autonomous. The features that we could include are as follows:

- the lidar sensor will recognize walls and make sure the robot does not hit or get too close to the wall.
- the camera can distinguish between walls and non-wall features, like windows.
- the camera can distinguish between painted and unpainted parts of a wall.

On the design side, adding a scissor lift on top of the chassis would be a great addition. This would make sure that the robot has a truly infinite workspace, at least in the context of a standard room for which the robot is designed.

Conclusion

The goal of the project was to come up with a robot that could potentially be used to paint walls in a room. The robot we designed, PaintBot, is essentially a mobile manipulator with a paint nozzle as the end effector. It is capable of moving around a room comfortably, while using the 6 Degrees of Freedom KUKA manipulator to reach different regions of the wall with an appropriate orientation. To better test the functionality of our solution, we carried out several tasks in both, the dynamics and the simulation front. The forward, inverse kinematics was derived using Denavit–Hartenberg parameters and Spong's convention. For simulation, the mobile manipulator was imported into Gazebo and maneuvered using publisher scripts. We wanted to use the tools to achieve two specific goals:

1) replicate the action of wall painting using the KUKA robot.
2) to test and control the chassis in an omnidirectional manner

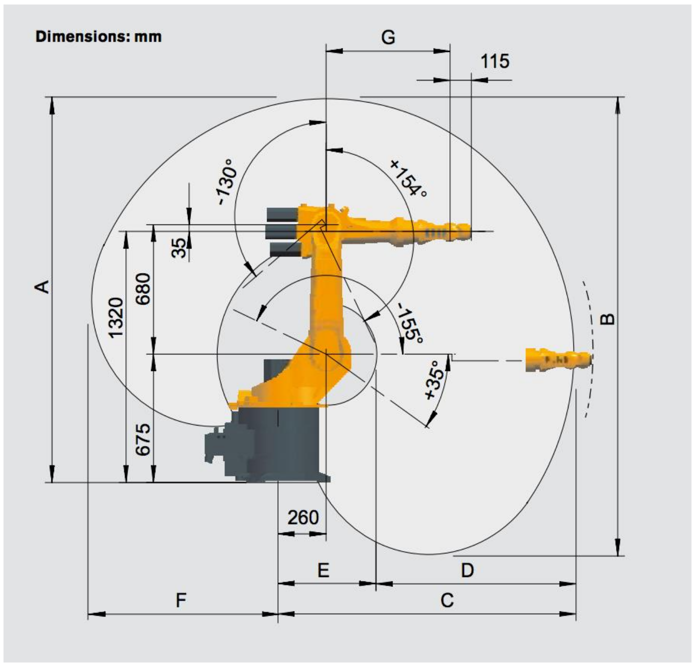
Both goals were accomplished as planned. The team looks forward to making further improvements to the solution over time, aiming to eventually make PaintBot a more autonomous robot.

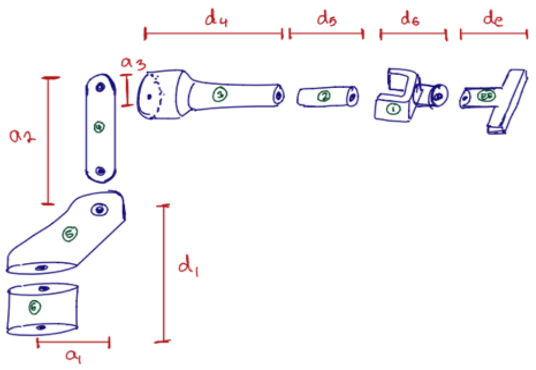
References

1. Okibo. (2020, February 27). *Autonomous Painting Robot*. YouTube. https://www.youtube.com/watch?v=hm9ZSN37jVM&ab_channel=Okibo
2. SMART Robotics for construction sites. OKIBO. (n.d.). Retrieved December 12, 2021, from https://okibo.com/
3. BigRentz, Inc. (2021, April 15). *How Construction Robots Will Change The Industry | BigRentz*. https://www.bigrentz.com/blog/construction-robots
4. Radcliffe, S. (2018, March 28). *Here Are 5 Ways to Reduce Your Exposure to Lead*. Healthline.https://www.healthline.com/health-news/5-ways-to-reduce-your-exposure-to-lead#Play,-work,-and-live-safe
5. *What is VOC Paint? - The Science of Paint | WOW 1 DAY PAINTING*. (2019). 1 Day Painiting.https://www.wow1day.com/blog/interior-painting/what-voc-paint-science-paint/
6. nicole.busenbark@mcri-us.com. (2021, July 6). *Robots and their Ingress Protection Rating*. Motion Controls Robotics - Certified FANUC System Integrator. https://motioncontrolsrobotics.com/robots-ingress-protection-rating/
7. Sales Center, KUKA. (2015). The masterful movers in the low payload category - robolution. Retrieved December 4, 2021, from https://robolution.eu/letoltes.php?type=media&name=960-kuka-kr-16-2-robot-adatlap.pdf
8. Piotrowski, N., & Barylski, A. (2014). Modelling a 6-DOF manipulator using Matlab software.
9. Spong, M. W., Hutchinson, S. A., & Vidyasagar, M. (2006). Robot modeling and control. IEEE Control Systems, 26(6), 113-115. https://doi.org/10.1109/MCS.2006.252815

Appendix

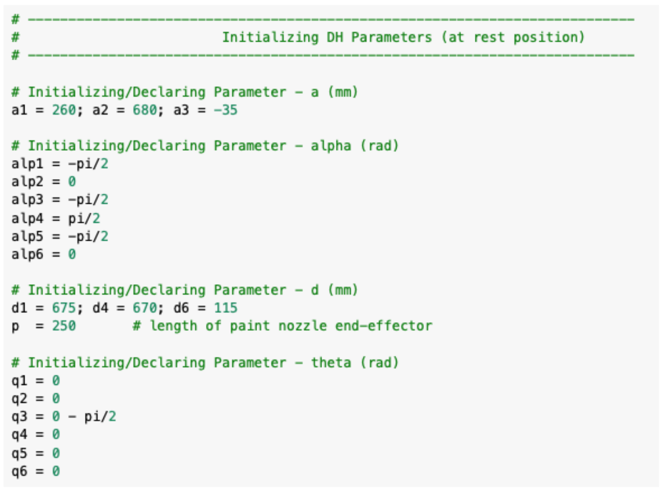
Appendix A: Dimensions of KUKA KR16-2



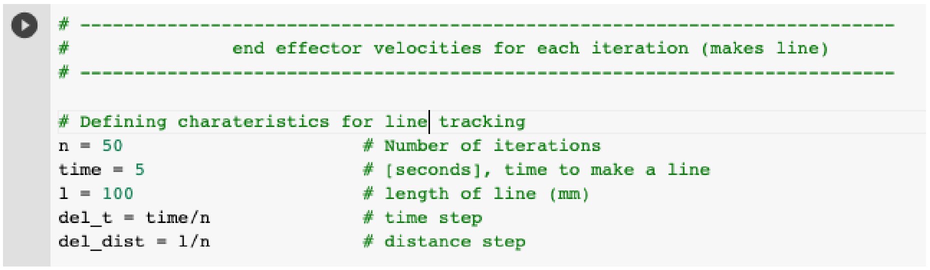


d_1	=	675	mm	a_1	=	260	mm
d_4	=	670	mm	a_2	=	680	mm
d_6	=	115	mm	a_3	=	-35	mm

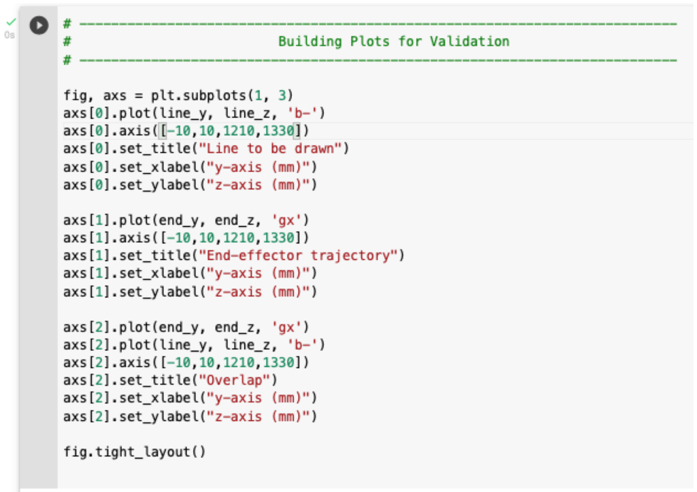
Appendix b: Python Program for Forward-Inverse Kinematics



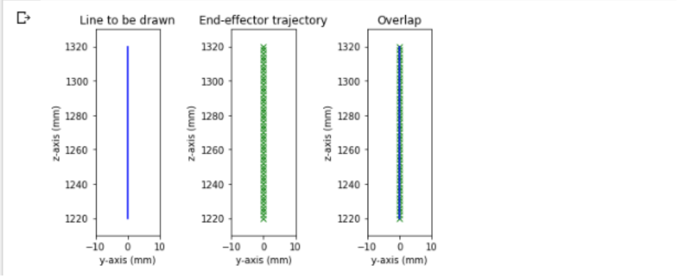
```
# ----------------------------------------------------------------------------
#                          Initializing DH Parameters (at rest position)
# ----------------------------------------------------------------------------

# Initializing/Declaring Parameter - a (mm)
a1 = 260; a2 = 680; a3 = -35

# Initializing/Declaring Parameter - alpha (rad)
alp1 = -pi/2
alp2 = 0
alp3 = -pi/2
alp4 = pi/2
alp5 = -pi/2
alp6 = 0

# Initializing/Declaring Parameter - d (mm)
d1 = 675; d4 = 670; d6 = 115
p  = 250       # length of paint nozzle end-effector

# Initializing/Declaring Parameter - theta (rad)
q1 = 0
q2 = 0
q3 = 0 - pi/2
q4 = 0
q5 = 0
q6 = 0
```



```
# ----------------------------------------------------------------------------
#               end effector velocities for each iteration (makes line)
# ----------------------------------------------------------------------------

# Defining charateristics for line tracking
n = 50                       # Number of iterations
time = 5                     # [seconds], time to make a line
l = 100                      # length of line (mm)
del_t = time/n               # time step
del_dist = l/n               # distance step
```



```
# ----------------------------------------------------------------------------
#                             Building Plots for Validation
# ----------------------------------------------------------------------------

fig, axs = plt.subplots(1, 3)
axs[0].plot(line_y, line_z, 'b-')
axs[0].axis([-10,10,1210,1330])
axs[0].set_title("Line to be drawn")
axs[0].set_xlabel("y-axis (mm)")
axs[0].set_ylabel("z-axis (mm)")

axs[1].plot(end_y, end_z, 'gx')
axs[1].axis([-10,10,1210,1330])
axs[1].set_title("End-effector trajectory")
axs[1].set_xlabel("y-axis (mm)")
axs[1].set_ylabel("z-axis (mm)")

axs[2].plot(end_y, end_z, 'gx')
axs[2].plot(line_y, line_z, 'b-')
axs[2].axis([-10,10,1210,1330])
axs[2].set_title("Overlap")
axs[2].set_xlabel("y-axis (mm)")
axs[2].set_ylabel("z-axis (mm)")

fig.tight_layout()
```



Appendix C: Google Drive Links for Simulation

PaintBot Simulation with Errors

- https://drive.google.com/file/d/1eZQPgkC-YV29ssKgescTMMSTPjffXoEI/view?usp=sharing

PaintBot Final Simulation Video

- https://drive.google.com/file/d/1eZQPgkC-YV29ssKgescTMMSTPjffXoEI/view?usp=sharing