ME 598
Introduction to Robotics

Fall 2022

Kinematics Validation & Path Planning

Group # R3
7 November 2022

Graduate students:
"I pledge that I have abided by the Graduate Student Code of Academic Integrity."

The report has been prepared by:

1. Lab leader: Krupansh Manish Shah
2. Sakshi Ranjeet Pol
3. Peera Tienthong
4. Shanmukha Sampath Kumar Tiriveedhi

Table of Contents

Abstract

The most important part about the manipulator is understanding its control and kinematics. How the end effector can be moved to make it arrive at a desired pose in a particular orientation is the vital knowledge required to perform any applications using a robotic arm. This lab experiment expatiates the process, computation, simulation, and examples which was theoretically learned in the class and later practically applied for this assignment. In this we have performed different tasks as per our previous report results and achieved the expected results. This task helped us in implementing and analyzing different challenges of working Dobot robot arm.

Introduction

In today's modern era use of Industrial robots is increasing worldwide and mostly they are used for material handling, welding, painting, assembling parts, packaging, handling, etc. Robot Manipulator is an electromechanical device that requires human expertise to accomplish a range of functions. Robotic arm kinematics deals with the analytical study of the geometry of the motion of a robot arm with respect to a fixed reference coordinates system without considering forces, moments that cause the motion. Typically, computation of homogeneous matrix and Denavit Hartenberg convention is used to determine the forward and inverse kinematics of any manipulator. In the Forward kinematics, the joint angles, and variable joint displacements (in case of prismatic joints) is known to us which is computed to find the position and orientation of the end effector in the world frame. The Inverse kinematics is opposite to the earlier one. Here, the end effector pose is known to us for which joint angles and displacement for each prismatic

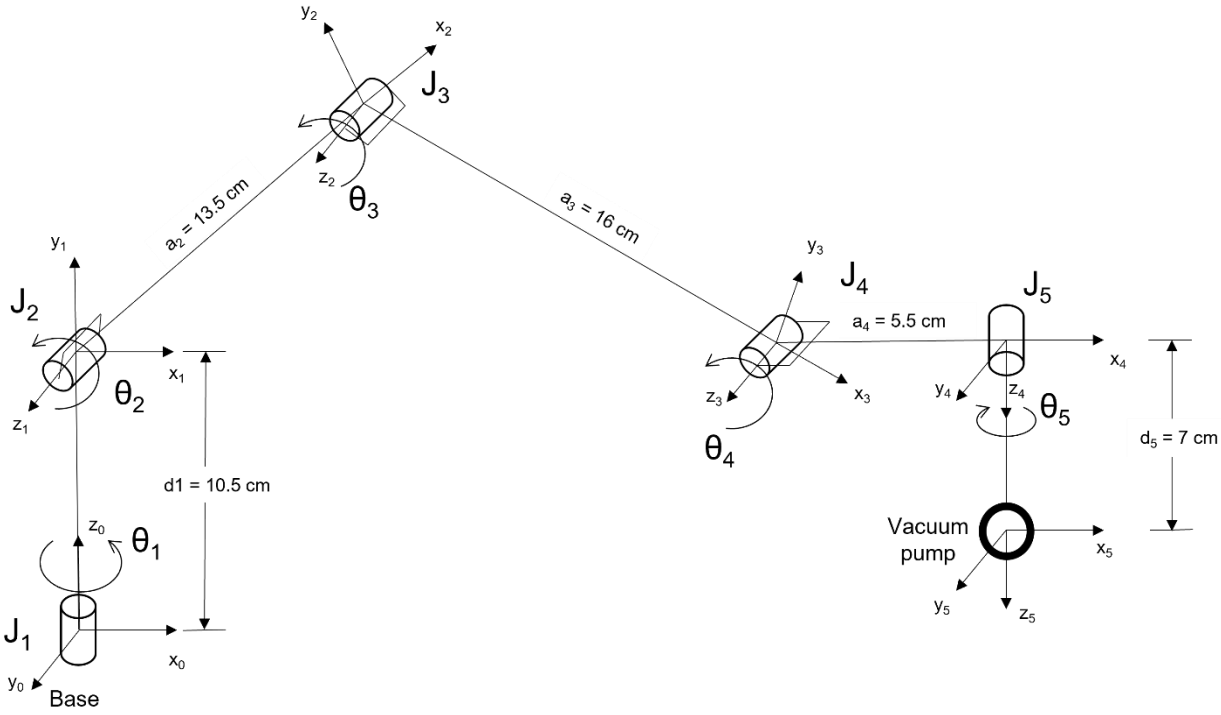

Figure 1: Dobot Schematics

link is computed. Reachable workspace, as the name suggest is the area if planar or volume if DOF more than or equal to 3 within which the end effector can move and perform functions/ applications within its boundary. The concept of singularity is also vital for understanding the kinematics and later applying it in a practical experimentation, but it was not covered within this report. Fig 1 shows the schematics of the dobot used in this laboratory experiment.

There are many different types of the robotic arm used for different purposes. In this report we have validated Dobot forward- and inverse-kinematics models and performed simple path planning for object manipulation using the Dobot robot arm with MATLAB. The Dobot robot arm consists of five revolute joints and a vacuum end-effector for lifting and moving ball. On the Dobot robot arm, we tested our earlier MATLAB simulation results. We have Validated following:

- Forward-Kinematics Model
- Inverse-Kinematics Model
- Basic Path Planning
- Obstacle Avoidance Using PRM Methods

In sum we have achieved the desired results perfectly.

Theory and Experimental Procedure

Part 1: Validate Forward-Kinematics Model

In this part of the experiment, the forward kinematics function developed in Lab 1, the provided dobot simulator, and the actual dobot were used. Three arbitrary joint configurations were set. Each of these were tested on each of the above-mentioned tools, respectively.

First, each of the configurations was given as input for the forward-kinematics function and the dobot simulator. Comparing the output end-effector positions, the results were pretty accurate between the two as expected, since both were carefully developed using ideal conditions, neglecting any possible source of errors.

Next, each configuration was tested on the actual dobot. Measuring the final end-effector positions using a physical ruler, discrepancies were evident all throughout the procedure. This was also expected as there was a handful of significant factors contributing to the skewed output including, but not limited to, human error in measurement, accuracy in initial position setting, platform height, and centering of the dobot axes. Taking the mentioned factors into account, the repeatability of the process was assessed. Although the final coordinates are accurate, they are relevant to the dobot position. So, the performance is consistent in the relevant coordinate system. The observations are discussed in the Results section.

Part 2: Validate Inverse-Kinematics Model

The theory and experimental procedure for this part of the experiment is the same with Part 1. The only difference is that the inverse kinematics function developed in Lab 1 was used.

Instead of directly setting arbitrary joint configurations, arbitrary end-effector positions were used. The joint configurations were then computed and determined through the inverse kinematics function of Lab1 using the set end-effector positions as input. The function's output was then used for the results comparison like Part 1. The observations are discussed in the Results section.

Part 3: Basic Path Planning

In this part, the workspace is assumed to be obstacle free. In theory, this implies that any path within the reachable workspace is valid and effective when moving the ball from one position to another. The only consideration made was to make sure that the ball will not be dragged across the surface of the platform during its transfer. Thus, intermediate path points were also developed. Since there were some discrepancies between the ideal and the actual values, trial-and-error in the physical measurement of the ball's destinations were made. This made sure that the ball landed exactly inside the platform circles.

First, the ball was moved from the home position to its first destination. It was supposed to be the outermost circle of the platform, but the dobot showed some significant erratic behavior when stretched up to the said point, so a closer circle was considered instead. Since the ball must not be dragged across the surface towards its destination, the first intermediate position exactly above the home position was inserted into the path. This provided lift for the ball before heading to the first destination. A second intermediate

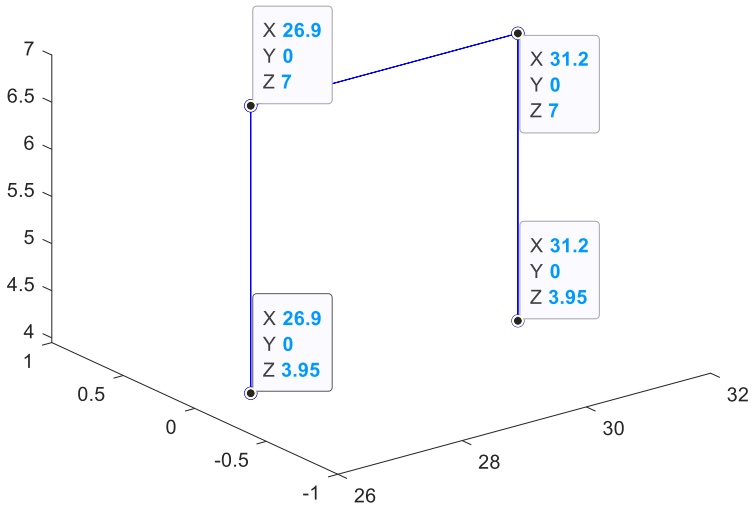

Figure 2: Part 3 - Step 1

position was also inserted into the path which is a position exactly above the first

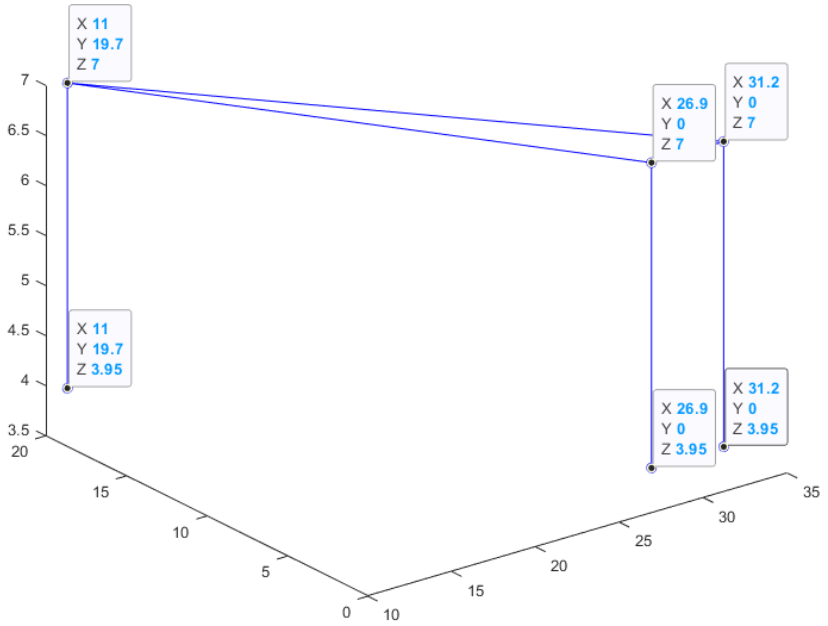

Figure 3: Part 3 - Step 2

destination. After dropping the ball in the first destination, the dobot headed back to the second intermediate position before heading home. This made sure that the ball was not kicked out of place. The path followed is as shown in Fig. 2.

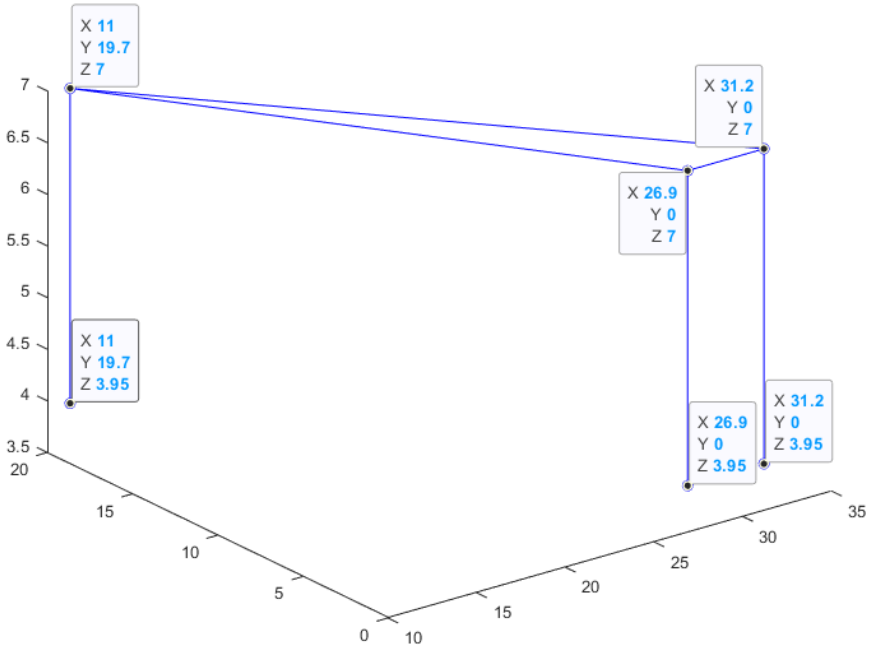

Figure 4: Part 3 - Final Task

In the second part of the experiment, the same ball was moved from the first destination to its second destination, all starting from home position. In picking the ball up from the first destination, all prior intermediate positions were still considered in the path. A third and final intermediate position was inserted into the path; this time, exactly above the second destination. In dropping the ball, the dobot first headed to the third intermediate position before placing the ball in the second destination. And before returning home, it went back straight into the third intermediate position to make sure the ball was not kicked out of place. The path followed is shown in Fig. 3.

For the final task in part 3, a combination of steps 1 and 2 has to be tested on the dobot. The dobot starts at the home position by picking up the ball and move to first position and move back to home position. And then it moves to the first position again to pick the ball up and move to second position and reset back to home position. As part of the intermediate positions, there will only be a vertical displacement observed(z-axis) so as to avoid dragging the ball along the surface. The path followed is shown in Fig. 4. The results and video links are discussed in the Results section.

Part 4: Obstacle Avoidance using PRM methods

For this part, the provided ME598_Lab2_randMap.m MATLAB script was used to generate aa probabilistic roadmap from the x0-z0 plane of the dobot workspace. y0 was always set zero throughout the motion along the path. First step is to generate a random sample of points in the X0-Z0 plane. The sample was generated in the range of 10 to 40 for the x coordinates and 0 to 20 for the z0 coordinates. These limits are considered based on the workspace limitations of the dobot.

All points below the z=3.9 in the roadmap are considered an obstacle and were therefore not considered part of the path. The generated sample can be seen in Fig 2a. The points are connected with their nearest neighbors as shown in Fig. 2b. This was

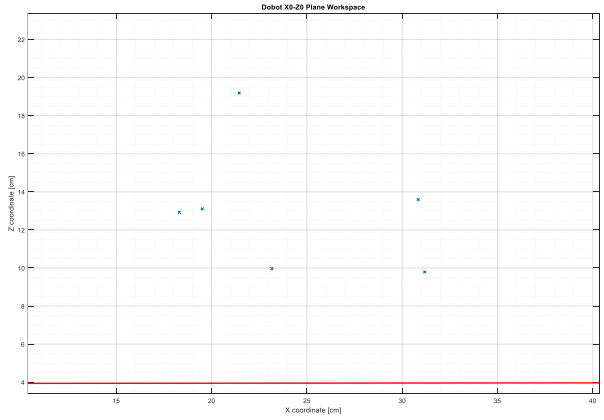

Figure 5a: Random Sample Generated

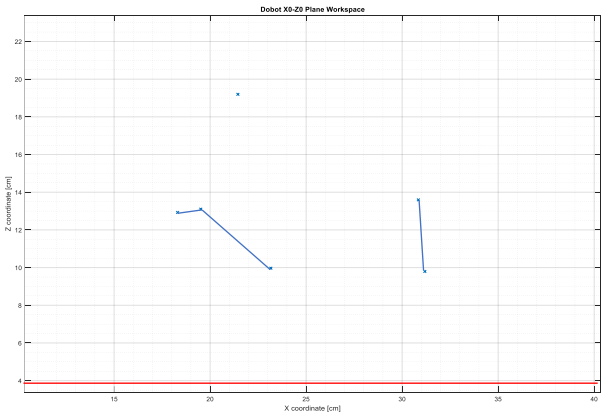

Figure 5b: Nearest Neighbors Connected

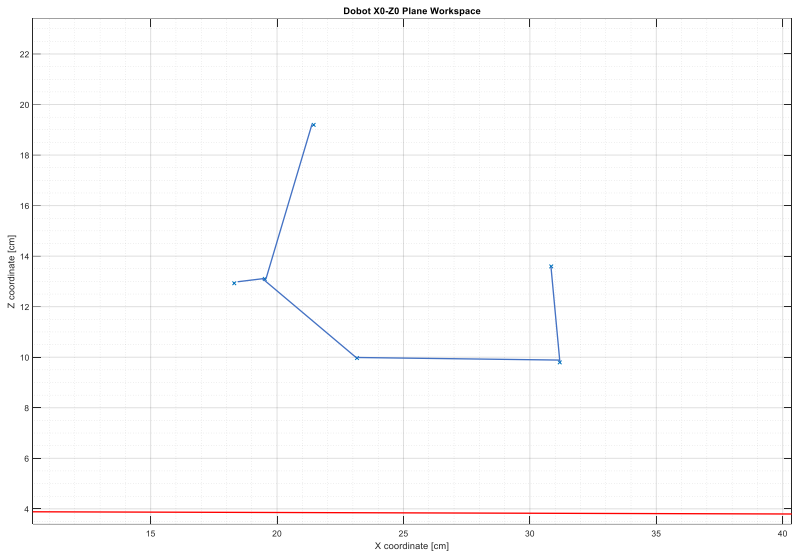

Figure 6: Enhanced Path connecting all the points

done manually. Once the connections were made, they were enhanced, and a continuous path was created. Smoothing was not needed here as the points that would be chosen to be part of the dobot path will not require it.

Our goal was to move the ball from one circle behind of home (first position) to the position that is one circle ahead of home (second position); this was set exactly along the y-axis. Another ball was placed in the home position and was considered the obstacle to be avoided using the generated roadmap.

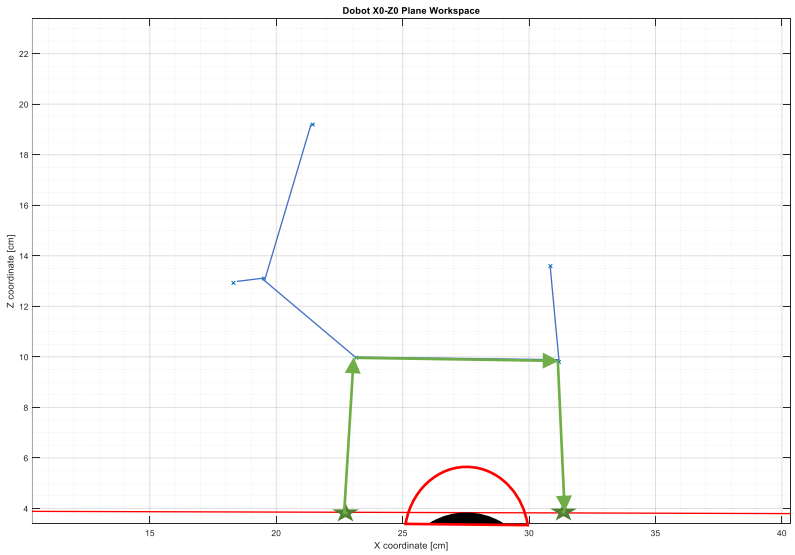

Figure 7: Final Path of the dobot

First, the coordinates of the first and second positions were physically measured using a ruler. Considering some discrepancies between ideal and actual values, a trial-and-error technique was used to re-calibrate the measured coordinates that was later used to

determine the joint configurations for the dobot to land the ball exactly on the desired circles (positions).

Next, two points along the generated roadmap path were chosen as intermediate points that would allow the ball to be lifted from first position and dropped at the second position, respectively, while avoiding the obstacle. By doing so, a total of four points were considered for the path of the motion, namely: first position (initial point), first intermediate position (point above and nearest to first position), second position (destination point), and second intermediate position (point above and nearest to second position).

Results

Testing Dobot Manipulator System

A practical demonstration of the Dobot under several conditions has been analyzed by analyzing computationally and validating the Dobot manipulator. To test the System, a home ball is placed at the center of the workspace with specific coordinates and the Dobot arm is moved at different positions around the home ball. A suction pipe attached to the end-effector is used to pick and place the ball at different positions.

Part 1. Validate Forward Kinematics Model

The Forward-Kinematics code developed in Lab 1 was used to verify the actual positions of the end-effector in the given workspace. Three different configuration points were added in the Dobot program to move the arm. After testing the Dobot arm, it was found that the position achieved was almost equivalent to the simulation positions of the Dobot. Positions of the ball were measured using a reference scale. To repeat the process, the armwas moved before every different trial to adjust the arm to its home position. Minor discrepancies were found dueto human error, parallax error and center axis alignment due to the base of the bot and the frame of the workspace. These can beminimized by firmly setting the base of the bot and the frame of workspace and by adjusting the error margin in the measured scale.

Serial Number	Input parameter (Joint Angles) (deg)	Simulation Results (End Effector Position) (cm)	Measured Values (End Effector Position) (cm)	Difference between measured and simulation results (cm)
1	[-20, 80, -100, 20, 0]	[21.5, -7.83, 11.3]	[22, -7.5, 11.5]	[0.5, 0.33, 0.2]
2	[50, 70, -120, 50, 90]	[13.1, 15.6, 3.93]	[12.5, 15.5, 3]	[0.6, 0.1, 0.93]
3	[30, 45, -75, 30, 60]	[25, 14.5, 5.05]	[24, 14, 4.5]	[1, 0.5, 0.55]

Table 1: Simulation results versus measured end effector position

Part 2. Validate Inverse Kinematics Model

Like the Forward-Kinematics validation model, same steps were carried out to verify the joint angles of the Dobot arm. Three different configurations were added in the Dobot

program to move the arm. Using the codedeveloped in Lab 1, three different set of joint angles were found using the different configurations. The achieved results were measured in the form of end effector positions with respect to the ball. Like the forward-kinematic results, similar discrepancies were found and could be adjusted by fixing the base of the bot and the frame of the workspace, by the adjusting the error margin in the measured scale. Parallax error stood out while measuring the joint angles. The measurement error could be reduced by using a digital protractor over a manual or printed paper protractor.

Serial Number	Input parameter (End Effector Position) (cm)	Simulation Results (Joint Angles) (deg)	Measured Values (Joint Angles) (deg)	Difference between measured and simulation results (deg)
1	[26, 11, 5]	[22.9, 47.4, -79.2, 31.8, 0]	[30, 45, -75, 30, 0]	[7.1, 2.4, 4.2, 1.8, 0]
2	[19, -16, 7]	[-40.1, 64.17, -96.91, 32.73, 0]	[-45, 65, -90, 30, 0]	[4.9, 0.83, 6.91, 2.73, 0]
3	[12, 24, 10]	[63.43, 62.23, -82.13, 19.9, 0]	[60, 62, -89, 11, 0]	[3.43, 0.23, 6.87, 8.9, 0]

Table 2: Simulation results versus measured joint angle

Part 3. Basic Path Planning

In an obstacle free workspace, the Dobot arm is moved from the home position to position 1 and then to position2 until it reaches back to the home position. Every time the ball reaches a position, the ball is dropped onto the workspace, record a delay and then pick up the ball again. In theory, the constant motion of the arm creates a paththat defines its trajectory motion. The same path is defined in the simulation analysis of the Dobot using the MATLAB code. While creating the path, it was made sure that the ball is picked up from the same location as theone it was dropped on. The performance of the Dobot arm is repeatable as it functions based on the positionsprovided. The Dobot arm will create the same path during each run. It is essential to set the home position of the Dobot within decimal accuracies for the vacuum pipe to pick up the ball.

End effector Path for Part 1: {[26.9;0;3.95], [26.9;0;7], [31.2;0;7], [31.2;0;3.95], [31.2;0;7], [26.9;0;7], [26.9;0;3.95]}

Dobot's motion in obstacle free workspace - part 1: Lab 2 - Basic Path Execution 1_ Group R3_Step 1

End effector Path for Part 2: {[26.9;0;3.95], [26.9;0;7], [31.2;0;7], [31;0;3.95], [31;0;7], [11;19.7;7], [11;19.7;3.95], [11;19.7;7], [26.9;0;7], [26.9;0;3.95]}

Dobot's motion in obstacle free workspace - part 2: Lab 2 - Basic Path Execution 2_Group R3_Step 2

End effector Path for Final Task: {[26.9;0;3.95], [26.9;0;7], [31.2;0;7], [31;0;3.95], [31;0;7], [31;0;3.95], [31;0;7], [11;19.7;7], [11;19.7;3.95], [11;19.7;7], [26.9;0;7], [26.9;0;3.95]}

Dobot's motion in obstacle free workspace - Final Task: Lab 2 - Basic Path Execution 2_Group R3_Final Task

Part 4. Obstacle Avoidance Using PRM Methods

The final path of the dobot involved 4 set of coordinates. Among these, first and last were the start and end positions for the ball. And the two points in between were chosen from the enhanced path of the Probabilistic Road Map and in a way to avoid the obstacle while the dobot moves the ball. Suction state of the vacuum cup also needs to be configured as the dobot has to know when to pick the ball up and drop it.

Path Followed: {[22.8;0;3.95], [23.16;0;9.96], [31.18;0;9.795], [31.2;0;3.95]}

Dobot's motion with an obstacle in place: Lab 2 - Obstacle Motion Planning part 1_Group R3

Conclusion

In this lab, we have successfully tested and validated the practical application of the Dobot with respect to the simulation. Understanding Forward and Inverse Kinematics of a manipulator is an essential aspect of each task provided in this lab. After successful trials, the Dobot functioned almost accurately keeping in mind the error adjustments that were observed due to human error and axis alignment of the Dobot base w.r.t the plane of the workspace. Well scripted MATLAB codes are presented in this report to justify the given analysis.

References

[1] Mark W Spong, Seth Hutchinson, M. Vidyasagar – Robot Modeling and Control – 1st Edition, 2006, Tehseen F. Abaas, Ali A. Khlief, Mohanad Q. Abbood," Inverse Kinematics Analysis and Simulation of a 5 DOF Robotic Arm using MATLAB , AL-Khwarizmi Engineering Journal, Vol. 16, No.1, (2020)

[2] MATLAB Examples and API Reference: https://www.mathworks.com/help/matlab/examples.html?s_tid=CRUX_topnav

Appendices

- MATLAB code for Forward Kinematics (Model Validation)

```
function coordis = ME598_GrpR3_FwdKin(confs)
disp(confs)
%Error throw1 : the confs size must satisfy 5x1, check matrix dimension
    if length(confs) < 5 || length(confs) > 5
        error('The number of degrees input in array must be equal to 5 (base,
shoulder, elbow, wrist, twist)');
    end
%Error throw2 : check 1: Base, Shoulder, and Twist joints are within ranges
```

```
    if confs(1) < -130 || confs(1) > 130 %base joint beyond range
        error('Warning! Base angle beyond robot workspace!');
    elseif confs(2) < 5 || confs(2) > 95 %shoulder joint beyond range
        error('Warning! Shoulder angle beyond robot workspace!');
    elseif confs(4) < -5 || confs(4) > 85 %wrist joint beyond range
        error('Warning! Wrist angle beyond robot workspace!');
    elseif confs(5) < -90 || confs(5) > 90 %twist joint beyond range
        error('Warning! Twist angle beyond robot workspace!');
    elseif confs(2) + confs(3) + confs(4) ~= 0
        error('Warning! Joint angles violate mechanical constraints on wrist.')
    else %within robot workspace
        [x,y,z,cup_dir] = fwd_kin(confs);
        if cup_dir < 0
            coordis = [x;y;z];
        else
            error('cup is not in downward direction');
        end

    end

end

%% Function involves in part3
function T = trans_f(d,theta,a,alpha) %input unit in cm and rad
    R_z = [cos(theta)  -sin(theta)  0 0;
           sin(theta)  cos(theta)   0 0;
           0            0           1 0;
           0            0           0 1];
    T_z = [1 0 0 0;
           0 1 0 0;
           0 0 1 d;
           0 0 0 1];
    T_x = [1 0 0 a;
           0 1 0 0;
           0 0 1 0;
           0 0 0 1];
    R_x = [1        0       0        0;
           0 cos(alpha) -sin(alpha) 0;
           0 sin(alpha) cos(alpha)  0;
           0       0       0        1];

    T = R_z*T_z*T_x*R_x;
end


function [x,y,z,cup_dir] = fwd_kin(confs) %input unit in cm and degrees
    q_base = deg2rad(confs(1));
    q_shoulder = deg2rad(confs(2));
    q_elbow = deg2rad(confs(3));
    q_wrist = deg2rad(confs(4));
    q_twist = deg2rad(confs(5));

    %transforming frames
    T_01 = trans_f(10.5,q_base,0,deg2rad(90));
    T_12 = trans_f(0,q_shoulder,13.5,0);
```

```
    T_23 = trans_f(0,q_elbow,16,0);
    T_34 = trans_f(0,q_wrist,5.5,deg2rad(90));
    T_45 = trans_f(7,q_twist,0,0);

    T_05 = T_01 * T_12 * T_23 * T_34 * T_45;
    x = T_05(1,4);
    y = T_05(2,4);
    z = T_05(3,4);
    cup_dir = dot([0;0;1],[T_05(1,3);T_05(2,3);T_05(3,3)]);

end
```

- MATLAB Code to move the dobot along a set of coordinates with suction states at each point

```
function moveDobotAlong(ps, suctionStates, simulate, dobotPlatform)

%%

% Clean up the workspace etc. so that variables are clear, as is the
% instrument (COM) configuration
clc;
instrreset

% Configure Dobot parameters for MATLAB communication
serport = 'COM6';
dobot = serial(serport, 'BaudRate' ,9600);

% Establish serial (USB) connection
fopen(dobot);

% Read current Dobot state
% Output the measured configuration and vacuum condition
[~, ~] = dobotReadDH(dobot);

% Settings for simulator
multiplot   = 1;
fignummulti = 1;
fignumXZ    = 2;

figure(fignummulti)
clf

figure(fignumXZ)
clf

for k = 1:length(ps)

    %disp(k)
    %disp(ps{k})

    q = ME598_GrpR3_InvKin(ps{k});

    if simulate
```

```
        % Simulate behavior
        dobotPlotXZ(q,fignumXZ);
        dobotPlot(q,fignummulti,multiplot);
    end

    if dobotPlatform
        dobotWriteDH(dobot, q, suctionStates(k));
    end

    pause(1)

end

% Disable Dobot object and release COM port (important)
fclose(dobot);
instrreset

end
```

- MATLAB code to move verify forward kinematics model

```
% Clean up the workspace etc. so that variables are clear, as is the
% instrument (COM) configuration
clear all;
close all;
clc;
instrreset

% Configure Dobot parameters for MATLAB communication
serport = 'COM6';
dobot = serial(serport, 'BaudRate' ,9600);

% Establish serial (USB) connection
fopen(dobot)

% Read current Dobot state
% Output the measured configuration and vacuum condition
[q_act, pump] = dobotReadDH(dobot)

%%

% Settings for simulator
multiplot   = 1;
fignummulti = 1;
fignumXZ    = 2;

figure(fignummulti)
clf

figure(fignumXZ)
clf
```

```
% Vacuum parameter: 0=OFF, 1=ON
desSuction = 0;

% Collecting angles of joints from the inverse Kinematics function
q_des = [-20, 80, -100, 20, 0];

% Some arbitrary configuration
%q_des = [26.9306;59.5060;-70.7972;11.2911;0]%[-20; 5; 0; -5; 90]



% Simulate behavior
dobotPlotXZ(q_des,fignumXZ);
dobotPlot(q_des,fignummulti,multiplot);


% Make Dobot move to desired configuration + vacuum state
dobotWriteDH(dobot, q_des, desSuction);

% Pause to give Dobot time to reach commanded configuration
pause(3)

% Read current Dobot state
[q_act, pump] = dobotReadDH(dobot)

% Check configuration error
err_q = q_des - q_act

%%

% Disable Dobot object and release COM port (important)
fclose(dobot)
clear all;
instrreset
```

- MATLAB code for Inverse Kinematics (Model Validation)

```
function q = ME598_GrpR3_InvKin(p)

Px = p(1);
Py = p(2);
Pz = p(3);

%Defining the the fixed values (i.e. arm length and theta_5)
d1 = 10.5;
a2 = 13.5;
a3 = 16;
a4 = 5.5;
d5 = 7;
theta_5d = 0;

try
    theta_1r = atan2(Py,Px);                    %theta 1 in radians
```

```
    theta_1d = theta_1r*180/pi;

    x1 = Px/cos(theta_1r) - a4;
    y1 = Pz - d1 + d5;
    x3 = (x1^2+y1^2-a2^2-a3^2) / (2*a2*a3);

    theta_3r = -acos(x3);                   %theta 3 in radians
    theta_3d = theta_3r*180/pi;

    x2 = a2 + a3*cos(theta_3r);
    y2 = a3*sin(theta_3r);
    theta_2r = atan2(y1,x1) - atan2(y2,x2);     %theta 2 in radians
                                                %due to the +5 to +95
                                                %constrain the two angles
                                                %are always subtracted.
    theta_2d = theta_2r*180/pi;

    theta_4d = -(theta_2d + theta_3d);          %theta 4 in degrees.

    if( (130 < theta_1d) || (theta_1d < -130) || (theta_2d < 5) || (theta_2d > 95) ||
(theta_4d < -5) || (theta_4d > 85) )
        error("Warning! End effector beyond robot workspace!");
    end

     q = [theta_1d; theta_2d; theta_3d; theta_4d; theta_5d];

catch
    error("Warning! End effector beyond robot workspace!")
end
```

- MATLAB code for validating inverse kinematics model

```
% Clean up the workspace etc. so that variables are clear, as is the
% instrument (COM) configuration
clear all;
close all;
clc;
instrreset

% Configure Dobot parameters for MATLAB communication
serport = 'COM6';
dobot = serial(serport, 'BaudRate' ,9600);

% Establish serial (USB) connection
fopen(dobot)

% Read current Dobot state
% Output the measured configuration and vacuum condition
[q_act, pump] = dobotReadDH(dobot)

%%

% Settings for simulator
multiplot   = 1;
fignummulti = 1;
```

```
fignumXZ    = 2;

figure(fignummulti)
clf

figure(fignumXZ)
clf



% Vacuum parameter: 0=OFF, 1=ON
desSuction = 0;

% Collecting angles of joints from the inverse Kinematics function
q_des = ME598_GrpR3_InvKin([12; 24; 10]);

% Some arbitrary configuration
%q_des = [26.9306;59.5060;-70.7972;11.2911;0]%[-20; 5; 0; -5; 90]



% Simulate behavior
dobotPlotXZ(q_des,fignumXZ);
dobotPlot(q_des,fignummulti,multiplot);


% Make Dobot move to desired configuration + vacuum state
dobotWriteDH(dobot, q_des, desSuction);

% Pause to give Dobot time to reach commanded configuration
pause(3)

% Read current Dobot state
[q_act, pump] = dobotReadDH(dobot)

% Check configuration error
err_q = q_des - q_act

%%

% Disable Dobot object and release COM port (important)
fclose(dobot)
clear all;
instrreset
```

- MATLAB code for reading the dobot position

```
function q = readDobotPosition()

% Clean up the workspace etc. so that variables are clear, as is the
% instrument (COM) configuration
close all;
clc;
instrreset
```

```
% Configure Dobot parameters for MATLAB communication
serport = 'COM6';
dobot = serial(serport, 'BaudRate' ,9600);

% Establish serial (USB) connection
fopen(dobot);

% Read current Dobot state
% Output the measured configuration and vacuum condition
[q, ~] = dobotReadDH(dobot);

instrreset

end
```

- MATLAB code for Basic Path Planning – Moving dobot along path selected

```
function ME598_GrpR3_Lab02_Part3()

%listOfPs = {[26.9;0;3.95], [26.9;0;7], [31.2;0;7], [31.2;0;3.95], [31.2;0;7],
[26.9;0;7], [26.9;0;3.95]};
%listOfPs = {[26.9;0;3.95], [26.9;0;7], [31.2;0;7], [31.2;0;3.95], [31.2;0;7],
[11;19.7;7], [11;19.7;3.95], [11;19.7;7], [26.9;0;7], [26.9;0;3.95]};
listOfPs = {[26.9;0;3.95], [26.9;0;7], [31.2;0;7], [31.2;0;3.95], [31.2;0;7],
[31.2;0;3.95], [31.2;0;7], [11;19.7;7], [11;19.7;3.95], [11;19.7;7], [26.9;0;7],
[26.9;0;3.95]};

%suctionStates1 = [1, 1, 1, 0, 0, 0, 0];
%suctionStates2 = [0, 0, 0, 1, 1, 1, 0, 0, 0, 0];
suctionStates3 = [1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0];

moveDobotAlong(listOfPs, suctionStates3, 1, 1)

end
```

- MATLAB code for Moving dobot along Probabilistic Road Map Path Planning

```
function ME598_GrpR3_Lab02_Part4()

listOfPs = {[22.8;0;3.95], [23.16;0;9.96], [31.18;0;9.795], [31;0;3.95]};

suctionStates1 = [1, 1, 1, 0];

moveDobotAlong(listOfPs, suctionStates1, 1, 1)

end
```

- MATLAB code for filtering points from random samples and consider the ones withing dobot workspace

```
%this function take coordinates(x,z) as input
%output: nearest points from the reference point
```

```
function path_new = connect_path(x,z)

    path = zeros(length(x),2);
    for i = 1:length(x)
        path(i,:) = [x(i), z(i)];
    end

    path_new = zeros(length(x),2); %initial path_new var array
    start_pos = path(1,:); %get start_pos
    path_new(1,:) = start_pos; %assign start_pos to path_new
    for i = 1:length(x)
        curr_pos = path_new(i,:); %point to compare cost
        if i == length(x)
            break
        end
        cost = zeros(1,length(x)); %initial cost var

        for j = 1:length(x) %assign cost array
            next_pos = path(j,:);
            if i > 1 && ismember(next_pos(1),path_new(:,1)) &&
ismember(next_pos(2),path_new(:,2))
                cost(j) = 0; %robot stay at the same pos and previous points are
already considered!
            else %calculate distance to all points from point ith
                cost(j) = sqrt((curr_pos(1)-next_pos(1))^2 + (curr_pos(2)-
next_pos(2))^2); %cost function defining absolute distance
            end
        end

        indx_nonzeros = find(cost); %find index of nonzeros cost
        min = cost(indx_nonzeros(1)); %first nonzero cost as min
        n = 1; %initial min and I index
        for k = 1:length(cost)
            if cost(k) == 0 %neglect cost = 0
                continue
            else %consider cost != 0
                if cost(k) <= min
                    min = cost(k);
                    n = k;
                end
            end
        end
        path_new(i+1,:) = path(n,:); %add the minimum distance point to the last
element array
    end

%check points below z=3.9 are obstables
function [x_filtered, z_filtered] = check_obstacles(x,z)

   x_filtered = [];
   z_filtered = [];

   for i = 1:length(x)
       if x(i) > 18 && x(i) < 31.5 && z(i) > 3.9
           x_filtered(end+1) = x(i);
```

```
            z_filtered(end+1) = z(i);
        end
    end
end
```

- MATLAB code used to generate dobot plots

```
function generatePlots(coordinates)
x = [];
y = [];
z = [];
for i=1:length(coordinates)
    point = coordinates{i};
    x(end+1) = point(1);
    y(end+1) = point(2);
    z(end+1) = point(3);
end

figure(10);
clf;
plot3(x, y, z, 'o-blue');

end
```