Dynamics and Control

Simulation of Robot Manipulator

Project Details:

Team Members:

1. Rahul Swayampakula - 2018102037
2. Karan Mirakhor - 2018102034
3. Jaideep Teru -2018102001

Simulation Software used : Webots R2020b

Manipulator model: UR3e by Universal robots,

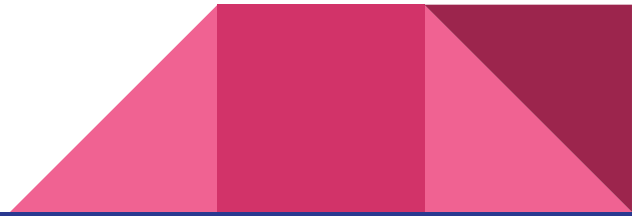
Environment : Picking cans from conveyor belt and dropping them into specific cartons.

Simulation Platform : Webots

- Webots is a free and open-source 3D robot simulator used in industry, education and research developed by cyber robotics.
- Operating system : Windows/ Ubuntu/ MacOS
- programming languages: C , C++, python, MATLAB
- Physics Engine : ODE(Open Dynamics Engine)
- Download link: https://www.cyberbotics.com/
- Git link: https://github.com/cyberbotics/webots

Why webots?

1. It is computable with multiple OS
2. It supports different programming languages like C, C++, Python, etc
3. It has a very good user-system interface.
4. It provides the users a variety of models which are very close to real life scenarios.
5. It has ODE as Physics engine that provides an good estimate of certain physical systems (rigid and soft body dynamics) .
6. We can have several simulated robots in the same world (platform) using several instances of the same controller



Structure of webots:

If we create a project/simulation in the webots it primarily contains 4 directories which determine the project:

1. **Protos:** This directory contains the .proto files. Proto files are basically the representation of the robots in the simulator.
2. **Worlds:** This directory contains .wbt files which are representing the simulated environment.
3. **Libraries:** This contains necessary include files for the project.
4. **Controllers:** This directory contains the controller file for the robot written in desired programming language. The controller files are the key part of the project which helps in spawning the robot and analysis the sensors in the environment

CAD Model of Robot and Gripper

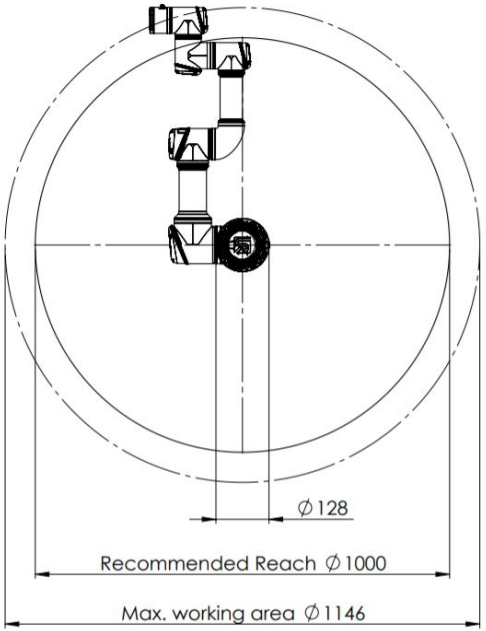
UR-3E Manipulator

- Commonly Used manipulator in factories for Palletizing
- This Manipulator has 6-DOF.
- Developed by Universal Bots.
- Reference link : ref

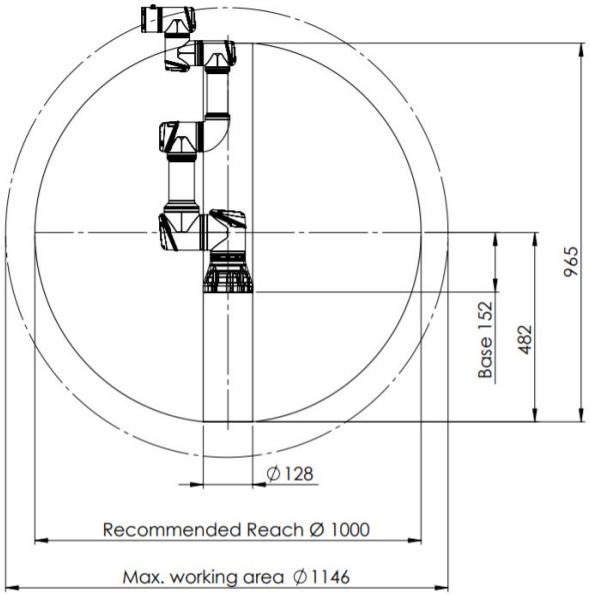
Robotiq 3F Gripper

- Gripper used with the Manipulator
- Each of the three fingers of the gripper can be individually controlled using 3 - Motors.
- Ideal for Picking and placing objects.

Workspace OF UR3e



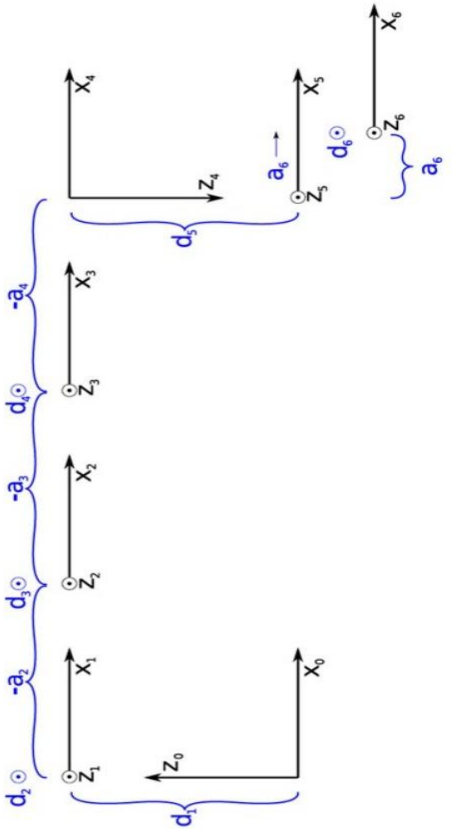
Working area, top view



Working area, side view

DH Parameters:

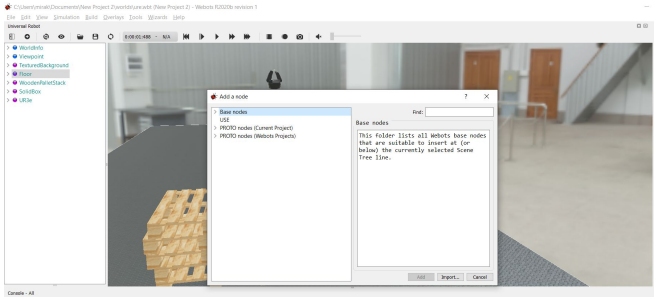
Joint	Θ	d	a	α
1	q1	0.1519	0	π/2
2	q2	0	-0.2436	0
3	q3	0	-0.2132	0
4	q4	0.11235	0	π/2
5	q5	0.08535	0	-π/2
6	q6	0.0819	0	0

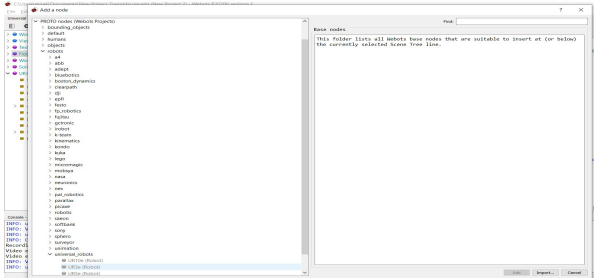


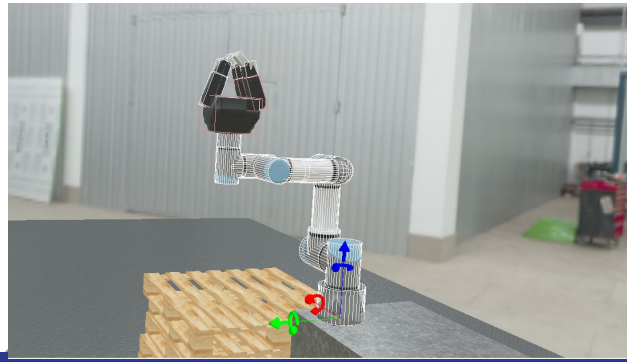
- *All joints are revolute.*
- *Assigning frames with proper dimensions.*
- *Dimensions are in cm.*

Importing the CAD:

- UR3e is one of the inbuilt Robot in the webots simulator.
- The ‘+’ option on the taskbar in Webots project environment can be used to simply import this robot using the GUI.
- Similarly environment can be construed by importing required CAD models.
- The position and orientation can be chosen by moving the models using GUI.

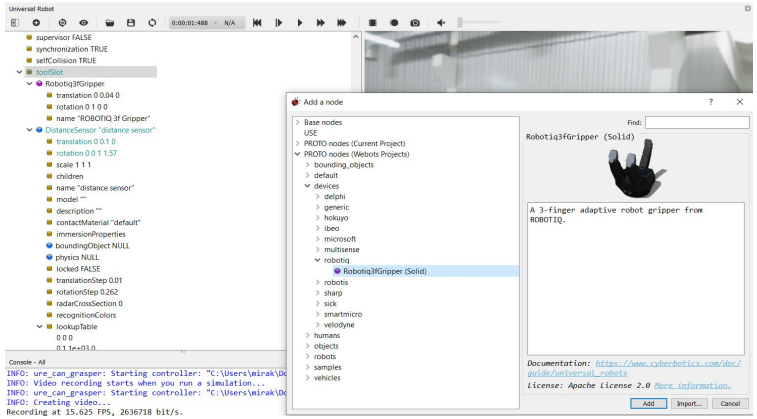


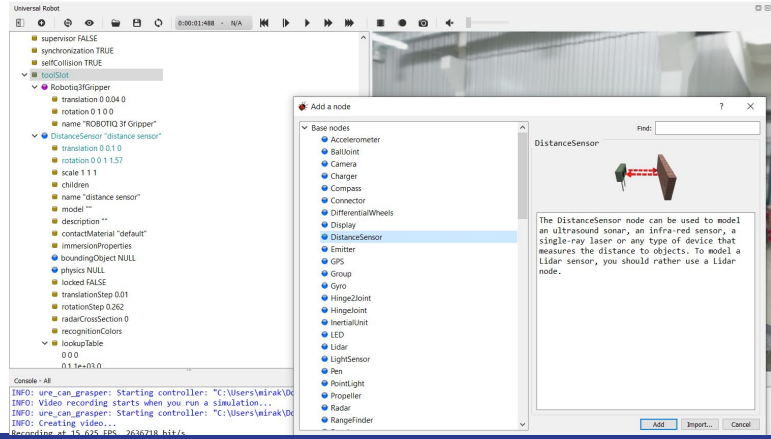




Attaching the gripper

- We used Robotiq 3F Gripper which is an inbuilt gripper in the webots.
- The gripper can be attached to the bot by selecting the appropriate proto file of gripper in the tool slot.
- The sensors are also attached in similar fashion.
- The GUI of the simulator helps in making this easier this way!!

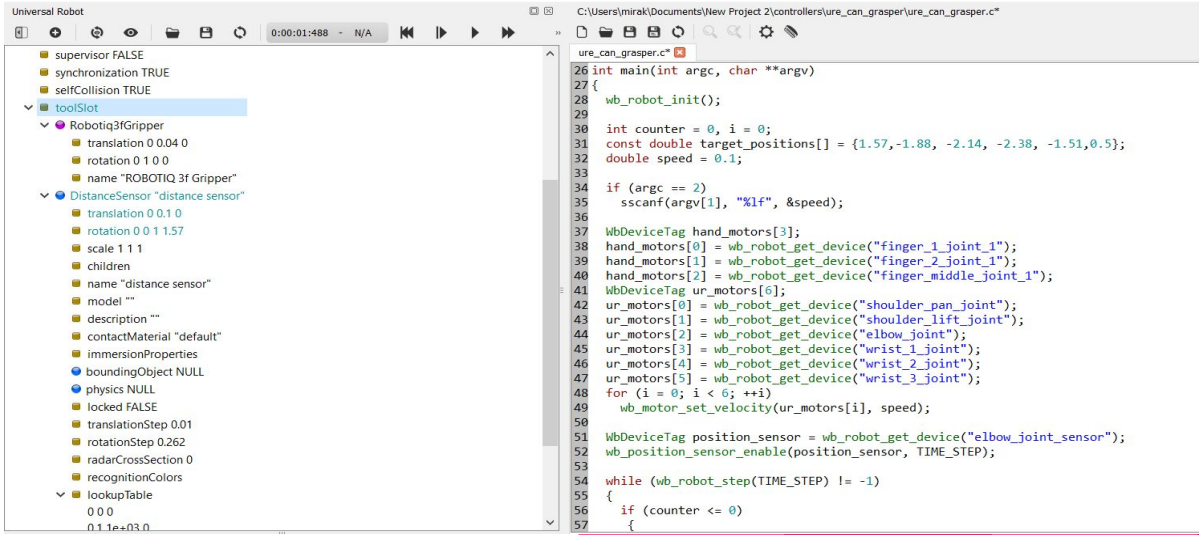




Accessing the joints:

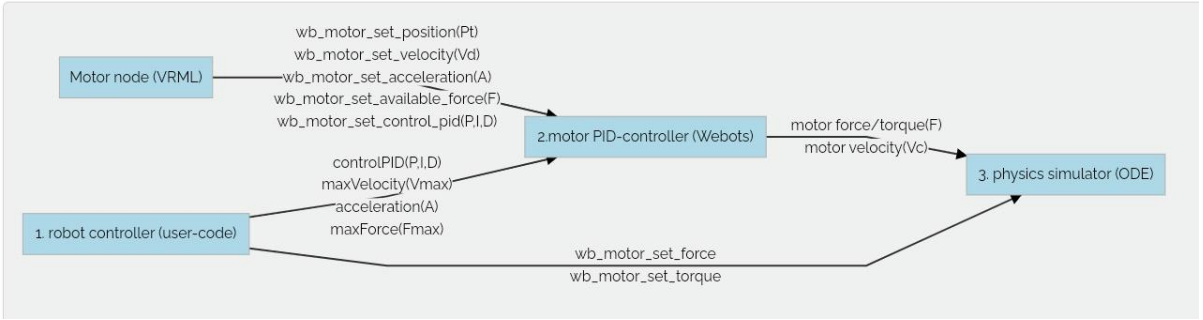
- Each joint of the robot has a unique name
- These names can be known by looking in the proto file (file modelling robot) of the robot
- Once we know the names of the joints we can access the things of the joint in the controller file

```
HingeJoint {
  jointParameters HingeJointParameters {
    axis 0 0 1
    anchor 0 0 0.152
  }
  device [
    RotationalMotor {
      name "shoulder_pan_joint"
      maxVelocity 3.14
      minPosition -6.28318530718
      maxPosition 6.28318530718
      maxTorque 330
    }
    PositionSensor {
      name "shoulder_pan_joint_sensor"
    }
  ]
```



Controlling the joints:

- At every joint there will be a motor placed , the type of motor depends on the type of joint we take .
- So if we want to move a joint it can be done through giving the appropriate command to the motor
- Here all our joints are revolute joints so a rotational motor can be at the joints to control the position.
- Here we can give inputs as the required angle but the motor needs the input as torque / current, this done by an inbuilt controller loop in the webots. This can also be tuned to our desired values by overwriting existing controller values.

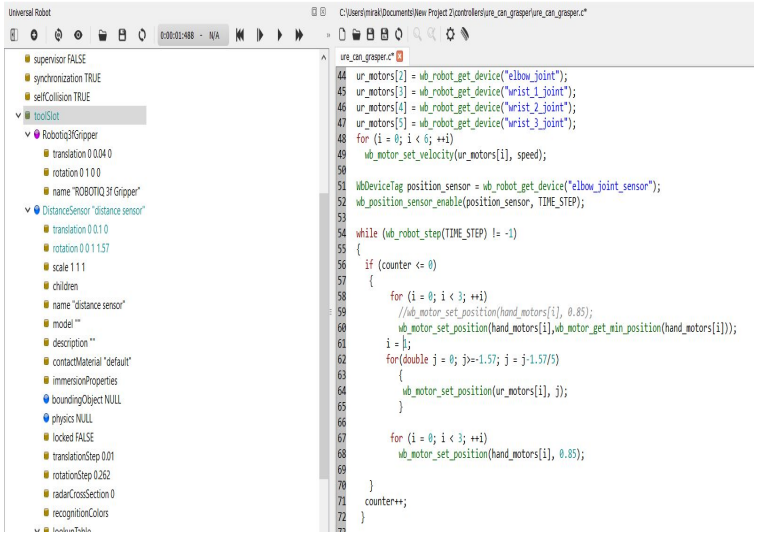


Writing controller

The controller can be written in desired programming language (used C) . As discussed in earlier slides by using the names of joints we access them and we set the desired position using the inbuilt libraries.

The documentation of webots libraries helps in programming this.

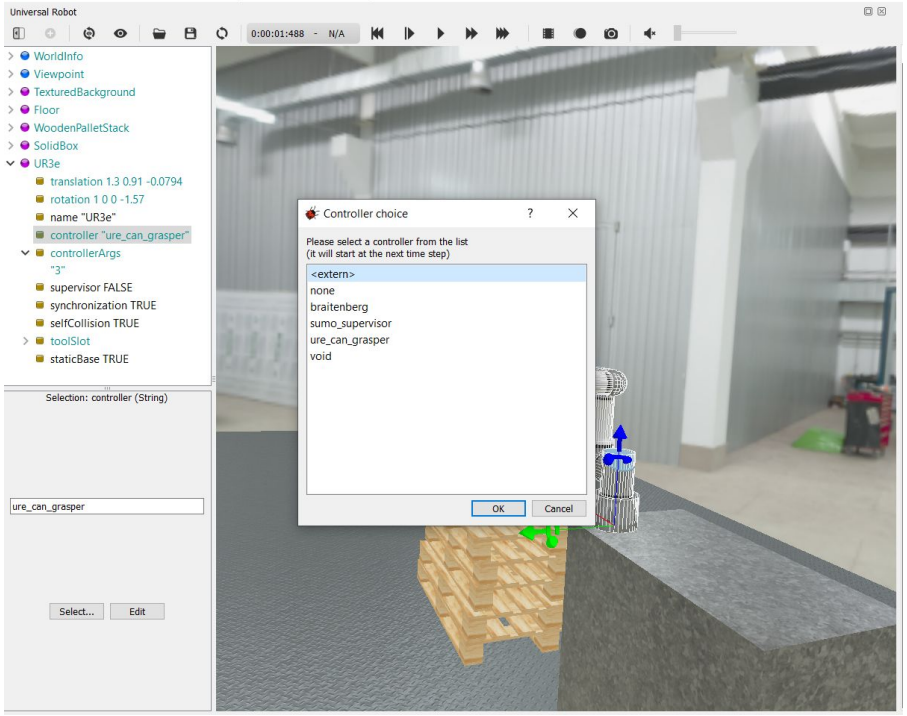
https://www.cyberbotics.com/doc/guide/tutorials



Attaching controller

We can build multiple controllers in same project and we can select a particular controller according to our requirements.

Choosing of controller is done using GUI .



Forward kinematics:

- a, b, c, d, e, f are q1, q2……q6 respectively.
- R_{ij} are the elements of the Forward Kinematics Transformation matrix ($^{G}T_{E}$) where ‘G’ is the Global frame and ‘E’ is the end effector frame i.e. the Gripper frame.

```
r11 = - sin(f)*(cos(d + pi/
2)*(cos(c)*cos(a + pi/2)*sin(b -
pi/2) + cos(a + pi/2)*cos(b - pi/
2)*sin(c)) + sin(d + pi/
2)*(cos(c)*cos(a + pi/2)*cos(b -
pi/2) - cos(a + pi/2)*sin(c)*sin(b
- pi/2))) - cos(f)*(sin(e)*sin(a +
pi/2) - cos(e)*(cos(d + pi/
2)*(cos(c)*cos(a + pi/2)*cos(b -
pi/2) - cos(a + pi/2)*sin(c)*sin(b
- pi/2)) - sin(d + pi/
2)*(cos(c)*cos(a + pi/2)*sin(b -
pi/2) + cos(a + pi/2)*cos(b - pi/
2)*sin(c))))
```
11:17 PM

```
r12 = sin(f)*(sin(e)*sin(a + pi/2)
- cos(e)*(cos(d + pi/
2)*(cos(c)*cos(a + pi/2)*cos(b -
pi/2) - cos(a + pi/2)*sin(c)*sin(b
- pi/2)) - sin(d + pi/
2)*(cos(c)*cos(a + pi/2)*sin(b -
pi/2) + cos(a + pi/2)*cos(b - pi/
2)*sin(c)))) - cos(f)*(cos(d + pi/
2)*(cos(c)*cos(a + pi/2)*sin(b -
pi/2) + cos(a + pi/2)*cos(b - pi/
2)*sin(c)) + sin(d + pi/
2)*(cos(c)*cos(a + pi/2)*cos(b -
pi/2) - cos(a + pi/2)*sin(c)*sin(b
- pi/2)))
```
11:17 PM

```
r13 = - cos(e)*sin(a + pi/2) -
sin(e)*(cos(d + pi/
2)*(cos(c)*cos(a + pi/2)*cos(b -
pi/2) - cos(a + pi/2)*sin(c)*sin(b
- pi/2)) - sin(d + pi/
2)*(cos(c)*cos(a + pi/2)*sin(b -
pi/2) + cos(a + pi/2)*cos(b - pi/
2)*sin(c)))
```
11:18 PM

```
r14 = (122*cos(a + pi/2)*cos(b -
pi/2))/5 - (131*sin(a + pi/2))/10
- (46*cos(e)*sin(a + pi/2))/5 -
(46*sin(e)*(cos(d + pi/
2)*(cos(c)*cos(a + pi/2)*cos(b -
pi/2) - cos(a + pi/2)*sin(c)*sin(b
- pi/2)) - sin(d + pi/
2)*(cos(c)*cos(a + pi/2)*sin(b -
pi/2) + cos(a + pi/2)*cos(b - pi/
2)*sin(c))))/5 + (87*cos(d + pi/
2)*(cos(c)*cos(a + pi/2)*sin(b -
pi/2) + cos(a + pi/2)*cos(b - pi/
2)*sin(c)))/10 + (87*sin(d + pi/
2)*(cos(c)*cos(a + pi/2)*cos(b -
pi/2) - cos(a + pi/2)*sin(c)*sin(b
- pi/2))))/10 + (213*cos(c)*cos(a +
pi/2)*cos(b - pi/2))/10 -
(213*cos(a + pi/2)*sin(c)*sin(b -
pi/2))/10
```
11:19 PM

```
r21 = cos(f)*(cos(a + pi/2)*sin(e)
+ cos(e)*(cos(d + pi/
2)*(cos(c)*cos(b - pi/2)*sin(a +
pi/2) - sin(c)*sin(a + pi/2)*sin(b
- pi/2)) - sin(d + pi/
2)*(cos(c)*sin(a + pi/2)*sin(b -
pi/2) + cos(b - pi/2)*sin(c)*sin(a
+ pi/2)))) - sin(f)*(cos(d + pi/
2)*(cos(c)*sin(a + pi/2)*sin(b -
pi/2) + cos(b - pi/2)*sin(c)*sin(a
+ pi/2)) + sin(d + pi/
2)*(cos(c)*cos(b - pi/2)*sin(a +
pi/2) - sin(c)*sin(a + pi/2)*sin(b
- pi/2)))
```
11:19 PM

```
r22 = - sin(f)*(cos(a + pi/
2)*sin(e) + cos(e)*(cos(d + pi/
2)*(cos(c)*cos(b - pi/2)*sin(a +
pi/2) - sin(c)*sin(a + pi/2)*sin(b
- pi/2)) - sin(d + pi/
2)*(cos(c)*sin(a + pi/2)*sin(b -
pi/2) + cos(b - pi/2)*sin(c)*sin(a
+ pi/2)))) - cos(f)*(cos(d + pi/
2)*(cos(c)*sin(a + pi/2)*sin(b -
pi/2) + cos(b - pi/2)*sin(c)*sin(a
+ pi/2)) + sin(d + pi/
2)*(cos(c)*cos(b - pi/2)*sin(a +
pi/2) - sin(c)*sin(a + pi/2)*sin(b
- pi/2)))
```
11:20 PM

```
r23 = cos(e)*cos(a + pi/2) -
sin(e)*(cos(d + pi/
2)*(cos(c)*cos(b - pi/2)*sin(a +
pi/2) - sin(c)*sin(a + pi/2)*sin(b
- pi/2)) - sin(d + pi/
2)*(cos(c)*sin(a + pi/2)*sin(b -
pi/2) + cos(b - pi/2)*sin(c)*sin(a
+ pi/2)))
```
11:20 PM

```
r24 = (131*cos(a + pi/2))/10 +
(122*cos(b - pi/2)*sin(a + pi/2))/
5 + (87*cos(d + pi/
2)*(cos(c)*sin(a + pi/2)*sin(b -
pi/2) + cos(b - pi/2)*sin(c)*sin(a
+ pi/2)))/10 + (87*sin(d + pi/
2)*(cos(c)*cos(b - pi/2)*sin(a +
pi/2) - sin(c)*sin(a + pi/2)*sin(b
- pi/2)))/10 - (46*sin(e)*(cos(d +
pi/2)*(cos(c)*cos(b - pi/2)*sin(a
+ pi/2) - sin(c)*sin(a + pi/
2)*sin(b - pi/2)) - sin(d + pi/
2)*(cos(c)*sin(a + pi/2)*sin(b -
pi/2) + cos(b - pi/2)*sin(c)*sin(a
+ pi/2))))/5 + (46*cos(e)*cos(a +
pi/2))/5 + (213*cos(c)*cos(b - pi/
2)*sin(a + pi/2))/10 -
(213*sin(c)*sin(a + pi/2)*sin(b -
pi/2))/10
```
11:20 PM

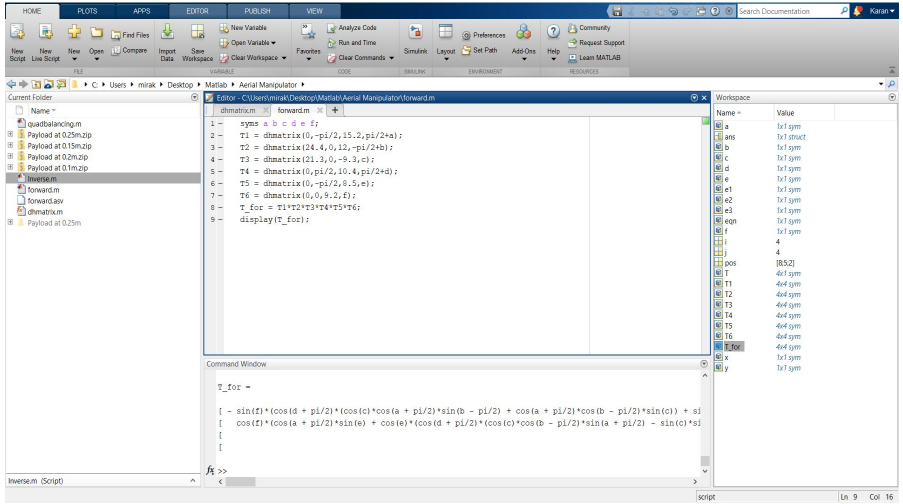
```
r31 = sin(f)*(cos(d + pi/
2)*(sin(c)*sin(b - pi/2) -
cos(c)*cos(b - pi/2)) + sin(d +
pi/2)*(cos(c)*sin(b - pi/2) +
cos(b - pi/2)*sin(c))) -
cos(e)*cos(f)*(cos(d + pi/
2)*(cos(c)*sin(b - pi/2) + cos(b -
pi/2)*sin(c)) - sin(d + pi/
2)*(sin(c)*sin(b - pi/2) -
cos(c)*cos(b - pi/2)))
```
11:21 PM

```
r32 = cos(f)*(cos(d + pi/
2)*(sin(c)*sin(b - pi/2) -
cos(c)*cos(b - pi/2)) + sin(d +
pi/2)*(cos(c)*sin(b - pi/2) +
cos(b - pi/2)*sin(c))) +
cos(e)*sin(f)*(cos(d + pi/
2)*(cos(c)*sin(b - pi/2) + cos(b -
pi/2)*sin(c)) - sin(d + pi/
2)*(sin(c)*sin(b - pi/2) -
cos(c)*cos(b - pi/2)))
```
11:21 PM

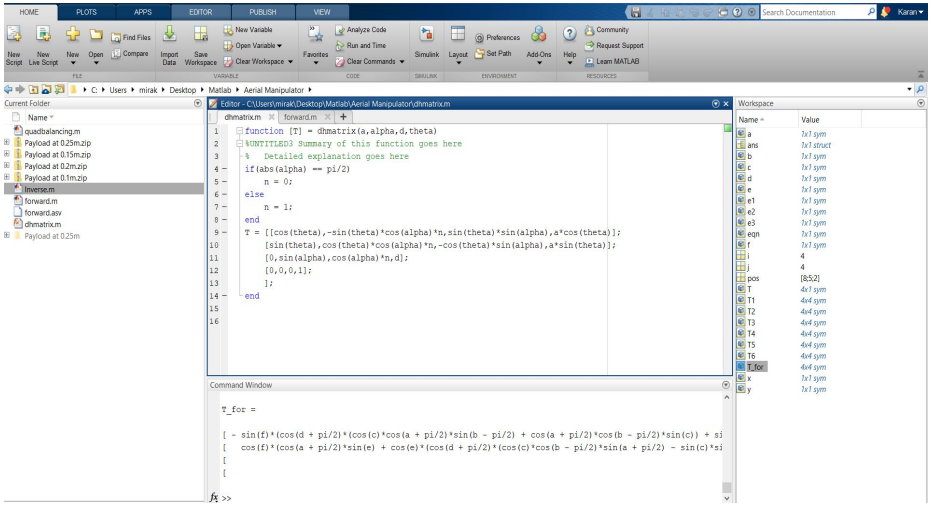
```
r33 = sin(e)*(cos(d + pi/
2)*(cos(c)*sin(b - pi/2) + cos(b -
pi/2)*sin(c)) - sin(d + pi/
2)*(sin(c)*sin(b - pi/2) -
cos(c)*cos(b - pi/2)))
```
11:22 PM

```
r34 = (46*sin(e)*(cos(d + pi/
2)*(cos(c)*sin(b - pi/2) + cos(b -
pi/2)*sin(c)) - sin(d + pi/
2)*(sin(c)*sin(b - pi/2) -
cos(c)*cos(b - pi/2))))/5 -
(213*cos(c)*sin(b - pi/2))/10 -
(213*cos(b - pi/2)*sin(c))/10 -
(87*cos(d + pi/2)*(sin(c)*sin(b -
pi/2) - cos(c)*cos(b - pi/2)))/10
- (87*sin(d + pi/2)*(cos(c)*sin(b
- pi/2) + cos(b - pi/2)*sin(c)))/
10 - (122*sin(b - pi/2))/5 + 76/5
```
11:22 PM

Code in MATLAB:



This script calculates the Forward kinematics Matrix in the variables as mentioned in previous slide.



This Function calculates the DH matrix for any given DH parameters.

Example of forward kinematics:

1. [xi,yi,zi] = [-0.223 , 0,0.694] → [0,-0.223,0.694] by rotating q1 by 90 degrees. (In the frame of Robot like mentioned in DH slide) - Simulation Video
2. [xi,yi,zi] = [-0.223 , 0,0.69.4] → [0.244,-0.223,-0.146] by rotating q1by 90 deg and q2,q3 by -90 degrees. (In the frame of Robot like mentioned in DH slide) - Simulation Video

By comparing theoretical and simulated outputs we got both values same. This verifies the calculations of the DH parameters.

Links for more simulation: here

Inverse Kinematics :

$$T_6^0 = \begin{bmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For our Task our θ_1 and θ_6 are not needed.

$\therefore \theta_1 = 0$ and $\theta_6 = 0$

$\theta_5 = tan^{-1}(T_6^0(2,1)/T_6^0(2,3))$

$\phi = \theta_2 + \theta_3 + \theta_4 = tan^{-1}(T_6^0(3,1)/T_6^0(1,1))$

$w_1 = d_5 sin(\phi) - d_6 sin(\theta_5) cos(\phi)$

$w_2 = d_1 - d_5 cos(\phi) - d_6 sin(\theta_5) sin(\phi)$

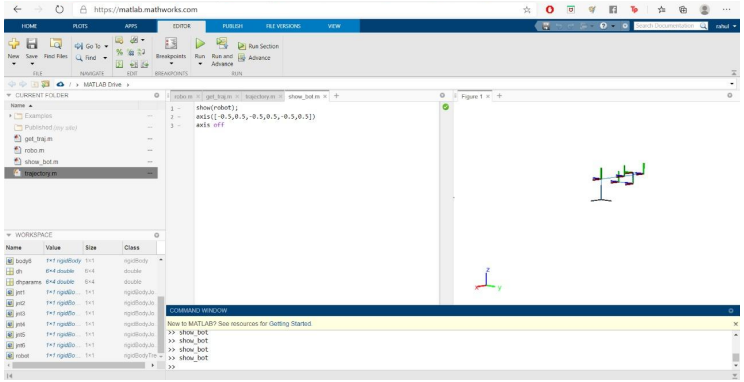
$p_1 = T_6^0(1,4) - w_1$

$p_2 = T_6^0(3,4) - w_2$

$\theta_3 = cos^{-1}[(p_1^2 + p_2^2 - a_2^2 - a_3^2)/(2a_2a_3)]$

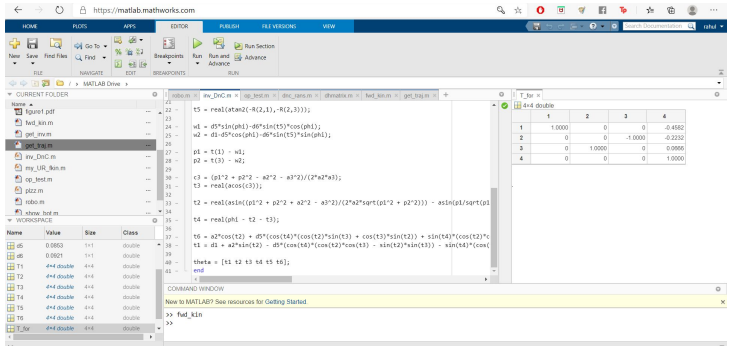
$\theta_2 = sin^{-1}1[(p_1^2 + p_2^2 + a_2^2 - a_3^2)/(2a_2\sqrt{p_1^2 + p_2^2})] - sin^{-1}(p_1/\sqrt{p_1^2 + p_2^2})$

$\theta_4 = \phi - \theta_2 - \theta_3$



Tested the results using inverse kinematics of Rigid body tree in the robotics toolkit of Matlab

Inverse Kinematics MATLAB code for testing

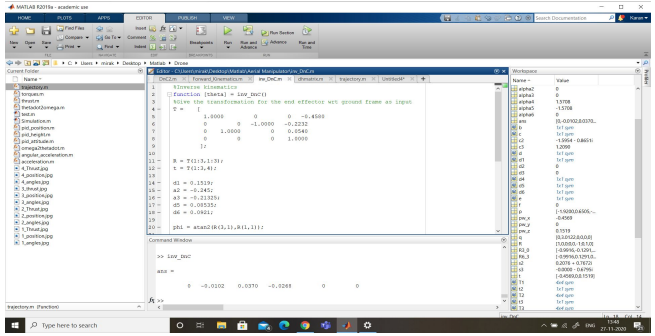


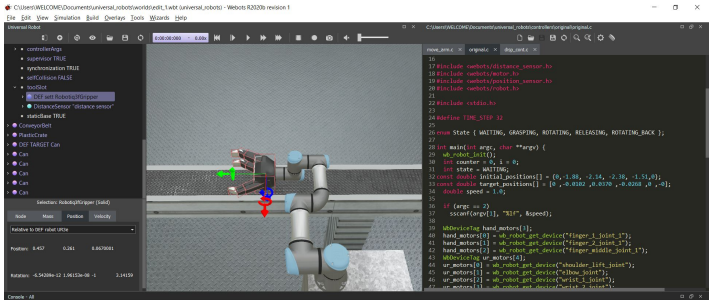
The above equations were implemented and tested in Matlab and values were compared with Simulation results

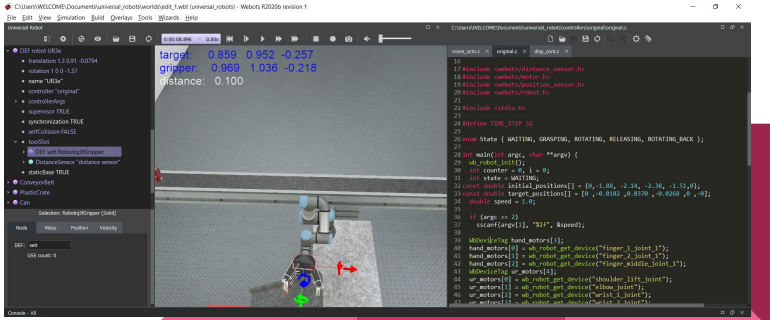
Inverse kinematics: (verification)

- To test the code a position and orientation input was given and the obtained output of angles are tested in simulation
- The bot started at angles [0,0,0,0,0,0] and it was supposed to attain
  - Pose [x,y,z] = [0.4580, 0.2232, 0.0640]
  - Orientation [[-1,0,0];[0,-1,0];[0,0,1]]
- The output from MATLAB is
  - theta = [0 ,-0.0102 ,0.0370 ,-0.0268 ,0 ,-0].
- With MATLAB output, in simulation the robot gripper achieved the
  - pose xyz=[0.4580, 0.2632, 0.0660]]
  - Orientation [[-1,0,0];[0,-1,0];[0,0,1]]

  https://youtu.be/-gpVdDCpUCI - Simulation check

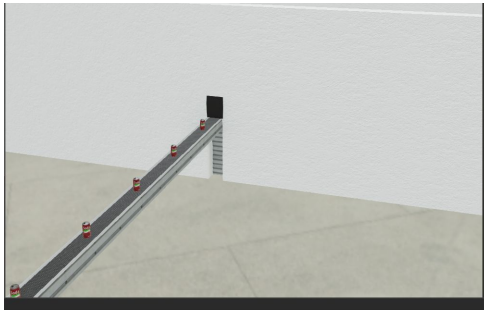


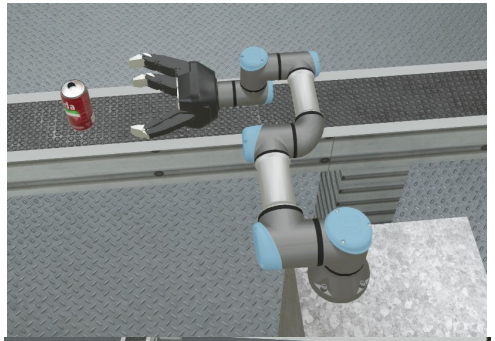




Targeted Task

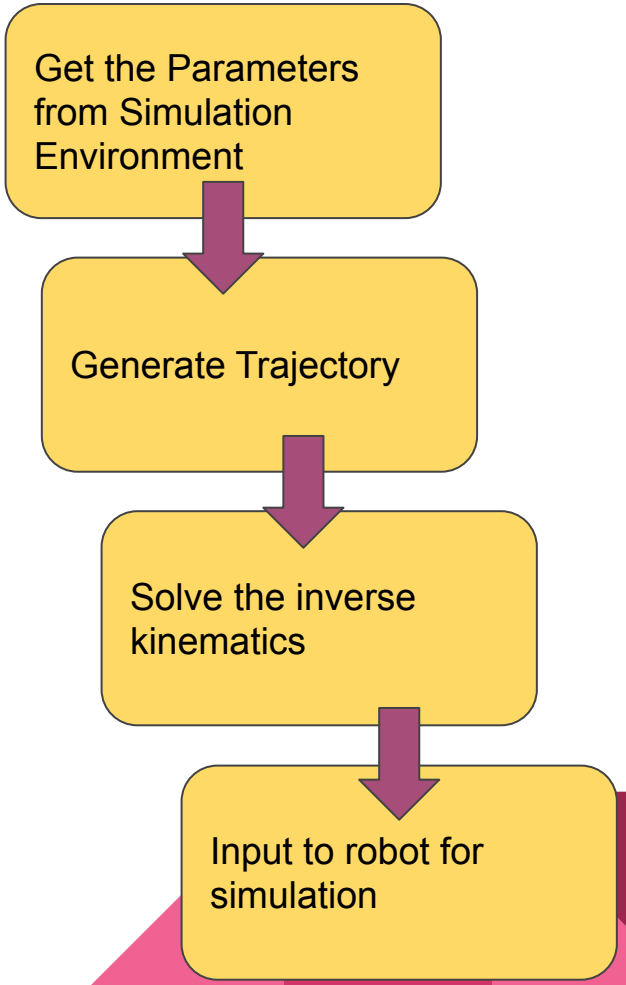
- A set of cans were placed on the Conveyor belt at a distance of 1.14 meter gap.
- The manipulator is mounted on the Slab next to the belt.
- As the conveyor belt moves the cans start approaching the manipulator one by one.
- The task for the manipulator is to pick up the can and making get in to the packing box placed below the slab through the channel.
- So we need to control conveyor belt motor and the motors of the robots to achieve the task in sync.
- Final result Link is given in *simulation results* slide.





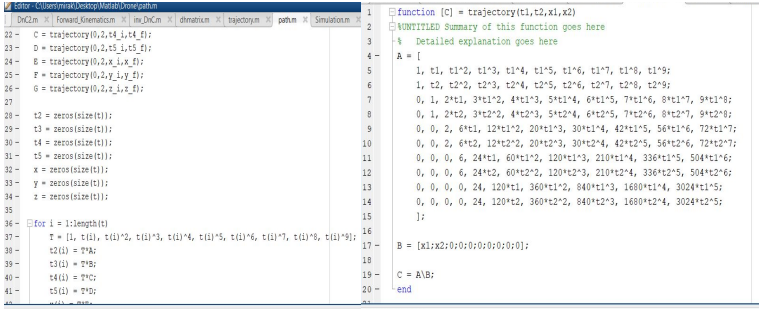
Environment for the Task:

- The motor of conveyor belt was set to a speed such that object on top of the belt move at a constant speed of 0.2m/s.
- In order to pick the next can perfectly, the entire process of picking and placing of each can needs to be done within 5.7 seconds at least.
- The initial and final positions (in the robot frame) are obtained from the simulation environment.
- The respective trajectory was generated by MATLAB code written.
- These positions given to inverse kinematics code written and respective angles are generated.

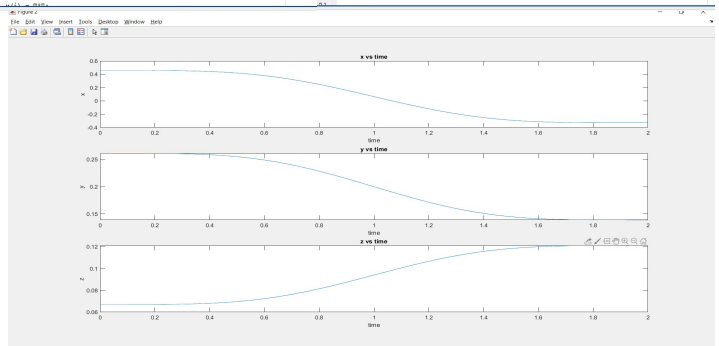


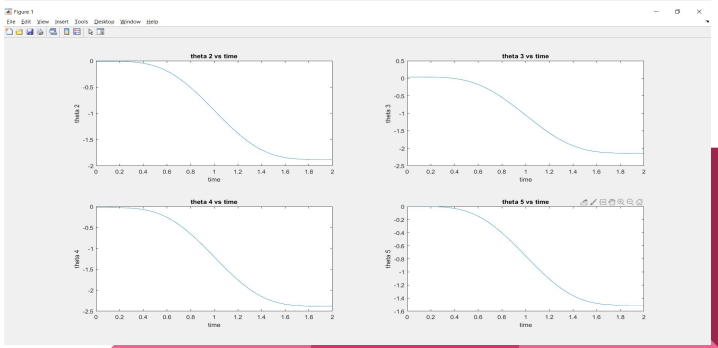
Path generation:

- We used *Minimum Snap Trajectory* for all the path equations to ensure smooth operation of Robot.
- Our whole task has 4 divisions namely :
  - *Initial position to Grabbing can position (in 2 seconds)*
    - xi = 0.4570; yi = 0.2610; zi = 0.0670;
    - xf = -0.3290;   yf = 0.1391; zf = 0.1210;
    - Theta_i = [0,0,0,0,0,0]
    - Theta_f = [0,-1.88,-2.14,-2.38,-1.51,0]
  - *Grabbing the can*
  - *Grabbing can position to initial position (in 2 seconds)*
    - xi = -0.3290;   yi = 0.1391; zi = 0.1210;
    - xf = 0.4570; yf = 0.2610; zf = 0.0670;
    - Theta_i = [0,-1.88,-2.14,-2.38,-1.51,0]
    - Theta_f = [0,0,0,0,0,0]
  - *Releasing the can*
- All these codes were tested using MATLAB before simulating



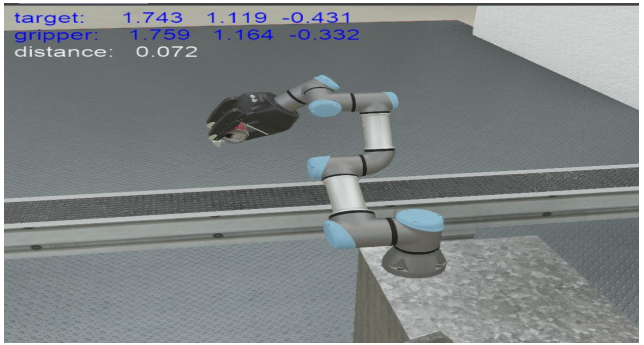
```
function [C] = trajectory(t1,t2,x1,x2)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
A = [
    1, t1, t1^2, t1^3, t1^4, t1^5, t1^6, t1^7, t1^8, t1^9;
    1, t2, t2^2, t2^3, t2^4, t2^5, t2^6, t2^7, t2^8, t2^9;
    0, 1, 2*t1, 3*t1^2, 4*t1^3, 5*t1^4, 6*t1^5, 7*t1^6, 8*t1^7, 9*t1^8;
    0, 1, 2*t2, 3*t2^2, 4*t2^3, 5*t2^4, 6*t2^5, 7*t2^6, 8*t2^7, 9*t2^8;
    0, 0, 2, 6*t1, 12*t1^2, 20*t1^3, 30*t1^4, 42*t1^5, 56*t1^6, 72*t1^7;
    0, 0, 2, 6*t2, 12*t2^2, 20*t2^3, 30*t2^4, 42*t2^5, 56*t2^6, 72*t2^7;
    0, 0, 0, 6, 24*t1, 60*t1^2, 120*t1^3, 210*t1^4, 336*t1^5, 504*t1^6;
    0, 0, 0, 6, 24*t2, 60*t2^2, 120*t2^3, 210*t2^4, 336*t2^5, 504*t2^6;
    0, 0, 0, 0, 24, 120*t1, 360*t1^2, 840*t1^3, 1680*t1^4, 3024*t1^5;
    0, 0, 0, 0, 24, 120*t2, 360*t2^2, 840*t2^3, 1680*t2^4, 3024*t2^5;
    ];

B = [x1;x2;0;0;0;0;0;0;0;0];

C = A\B;
end
```

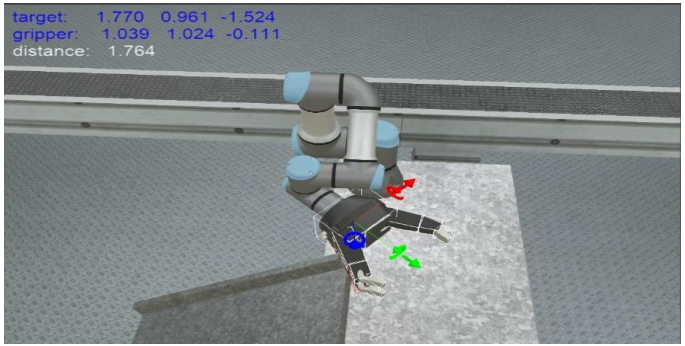


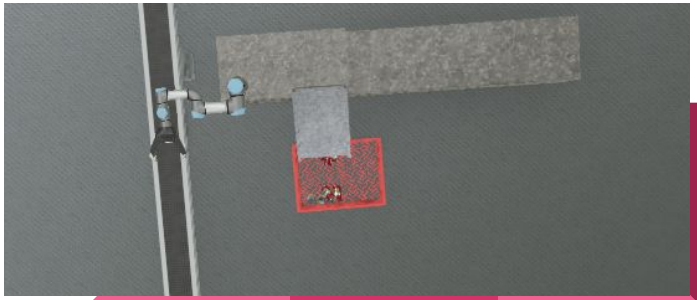


Simulation:

- The mentioned task in the *targeted tasks* slide was done using the inverse kinematics and path generation as mentioned in the previous slides.
- A distance sensor was placed in the gripper slot of the robot to detect the approach of the target to hold it properly.
- The respective controllers for the conveyor belt and UR3e robot were coded in cpp to make the process in sync.
- Link for the simulation: **[Link]**
- Link for playlist of all simulations: **[Link]**







Miscellaneous

- **Tuning PID controller:** The inbuilt motor PID controllers really helped in making the output from the motors smooth curved functions. Few values are experimented and it was tuned to achieve required setting.
- **Rigid body Tree(MATLAB):** Rigid body tree for the robot was built in MATLAB for testing the validity of equations. This is built using the DH-parameters.
- **Position Marker:** A controller in webots was written to display the position of the Gripper and the target continuously so that it will be easy to track position of our objects. This can be viewed at the top-left corner in many of the generated simulations. (Note: All the positions generated are w.r.t Global Frame)

Inertial Parameters of the Bot:

Link	mass	radius	length	Ixx	Iyy	Izz
1	1.98	0.75	0.178	0.2837	0.2837	0.5569
2	3.4445	0.75	0.244	0.5015	0.5015	0.9689
3	1.437	0.75	0.213	0.2075	0.2075	0.2075
4	0.871	0.75	0.12	0.1235	0.1235	0.1235
5	0.805	0.75	0.12	0.1145	0.1145	0.1142
6	0.261	0.32	0.042	0.0067	0.0067	0.0067

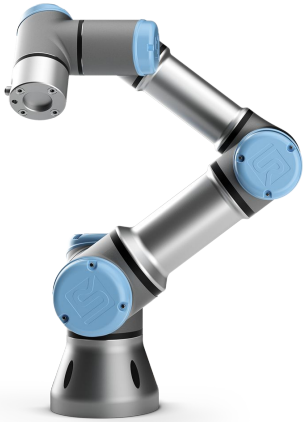


Table: Inertial parameters for all rotating links

Units: Length in metres; mass in kgs; radius in metres; Ixx,Iyy,Izz(inertias) in kg-m^2

- *Maximum Payload : 3Kgs*
- *Maximum Torque: 2 Nm to 330 Nm (varies from joint to joint)*

References:

- Universal Robots - Parameters for calculations of kinematics and dynamics (universal-robots.com)
- universal_robot/ur3e.urdf.xacro at kinetic-devel · ros-industrial/universal_robot (github.com)
- ALANDR-report.pdf (illinois.edu)
- kutzer/URToolbox - File Exchange - MATLAB Central (mathworks.com)
- https://in.mathworks.com/help/robotics/ref/inversekinematics-system-object.html
- https://in.mathworks.com/help/robotics/ref/rigidbodytree.html

Codes for Testing and Simulation :

- Universal Robots - Parameters for calculations of kinematics
- Youtube Playlist
- Github Link for codes and Simulation

Thank you!!