TAMPERE UNIVERSITY OF TECHNOLOGY

ASE-9406

Robot Manipulators: Modelling, Control & Programming

ASSIGNMENT 2

Lego Manipulator

Group 12

Salman Azim 245038

Antti Hietala 264340

Shambhu Mishra 267600

Table of Contents

Introduction:

This assignment is done for partial completion of the course Robot Manipulators: Modelling, Control & Programming. We were three members in the group for this assignment. We were asked to design and build a Robot manipulator with Lego Mindstorms EV3 kit which can draw different types of mobile phone according to the given configuration. This report will reflect the design process of the robot manipulator, the decisions taken for design purpose, technologies used, programming of the robot, problems faced while doing this assignment and limitations of this robot manipulator.

Provided Kits:

For this assignment we were provided with some kits. The list of the kits is given below:

1. Lego MINDSTORM EV3 Kit Box
2. Raspberry Pi 3 Model B with power brick and MicroSD card and casing
3. Brick Pi 3
4. Ethernet cable

In the MicroSD card the image was provided. We flashed and verified the image with Etcher software and it was ready to insert in the Raspberry Pi. Raspbian operating system for Raspberry Pi was pre-installed. Also, VNC server was installed to connect remotely with the raspberry pi without any monitor, keyboard or mouse. ROS was also installed in the system.

Technologies & Process:

There were several technologies & programming languages used for this assignment. The list is given below:

1. ROS for making communication between planner and task files
2. C++ for programming the trajectory planner
3. Python 3.6 for programming the task of the robot
4. Matlab script for requesting mobile phone type.

At first the ROS master will be initiated. Matlab, planner and the task file will be the slave. All the communication will be done by using the publish and subscribe method of ROS. At first, the request for the type of mobile will be send through the matlab to the task.py file which will send the 3D coordinates according to the request to the planner.cpp file. The planner file will calculate the trajectory of the joints of the robot and send the JointTrajectory to the task.py file. According the trajectory received by the task.py file it will tell the robot to move it joints to each of the points needed to draw. After all the requests are done, it will stop moving and the process can be killed by using cntrl+c on the console.

Setting up the Pi:

First, we needed to mount the Brick Pi3 on top of the Raspberry Pi and set up the casing, so that it would be sturdy enough and does not get broken. After that we connected the Raspberry Pi with a laptop using the Ethernet cable and we used Wireshark software to get the IP of our Raspberry Pi. Although there are many ways to get the IP address, however we used Wireshark before that is why it was easy for us. In Wireshark we started capturing data packet with only switching on the ethernet option. Then from the captured packet we searched and found the IP address of the Pi and it was 169.254.173.214.

Then, we installed VNC viewer in our laptop to access the Pi remotely. After setting up the IP address and id password provided previously we got the screen of the Raspberry Pi OS easily in our PC. Then in Raspberry Pi console we tried running the $ roslaunch script. For that, first we run the $ roscore command. In a different console window, we run the $ cd ~/catkin_ws, which will then move into the catkin workspace. After that we run the setup.bash file provided with the ROS installation to add the environment variables to the path. It was done with the command $ source devel/setup.bash. The planner and the task file were provided with maximum of the coding done for some other robot. We connected the 3 Lego motors with the BrickPi3 and turned on the Pi with the 9V adapter with 8*AA battery pack. The programs were done for four motors initially. We commented out the fourth motor setting from the code and tried running the roslaunch command $ roslaunch trajectoryplanner start.launch. We observed that the motors had some movement and we were good to proceed with the robot design.

Designing the Manipulator

At first, we tried to brainstorm about how we will build the robot. After having many ideas, we built a 3 degrees of freedom manipulator all having revolute joints. The base of the robot was attached on top of the casing of the Pi to give some additional height and so that it remains fixed. Initially the robot looked like the picture below. Initially, our idea was to draw on the floor. Our design decisions were taken based on that.

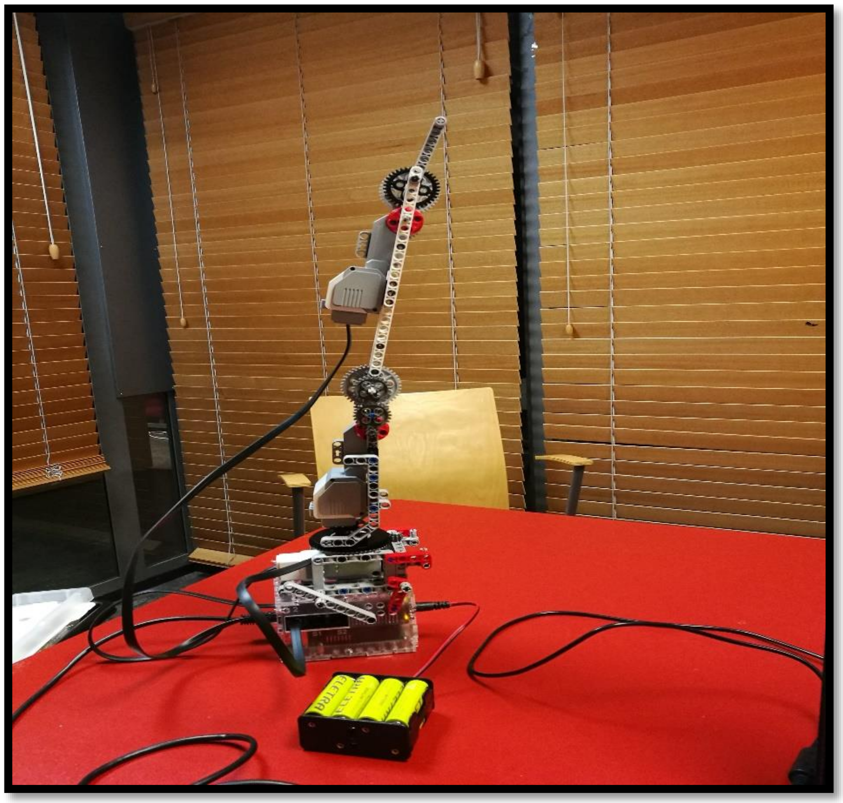


Fig: Initial Robot Design

After that we had some design changes for making the robot more flexible. Initially the motors were not able to withstand the weight of the arm. We were free to choose the end effector type, that's why we decided to design the end effector ourselves.

End effector design

It was decided, that pencils would be used to draw the cellphone, due to the good supply of them and because they can be easily shortened, so they can be placed in smaller space. Because of the robot's design, the pen would have to be aligned along the last link and to minimize the offset in coordinates, the end effector extends directly from the last link.

Since the end effector needed to be attached to the Lego blocks, it was necessary to take in to consideration the standard measurements of the Lego parts, i.e. the hole sizes and distances, dimensions of the blocks, and so on. The design of the end effector can be seen in figure 2.

The issue with the design was how it would be possible to hold the pencil firmly enough in place, so that it does not slip out of the tool when lifting the end effector, or slide deeper when the pen is pressed against the paper. For that reason, additional whole was made to the side of the end effector, so that a bolt could be used to tighten the pencil in place. However, this was later found to be unnecessary, as the pencil fitted very tightly to the hole designed for it. As a precaution, an additional bar was placed behind the pencil, which would prevent the slipping in case the robot pressed the pencil too hard against the paper. With the bar it was also possible to modify how far the pencil comes out of the end effector.

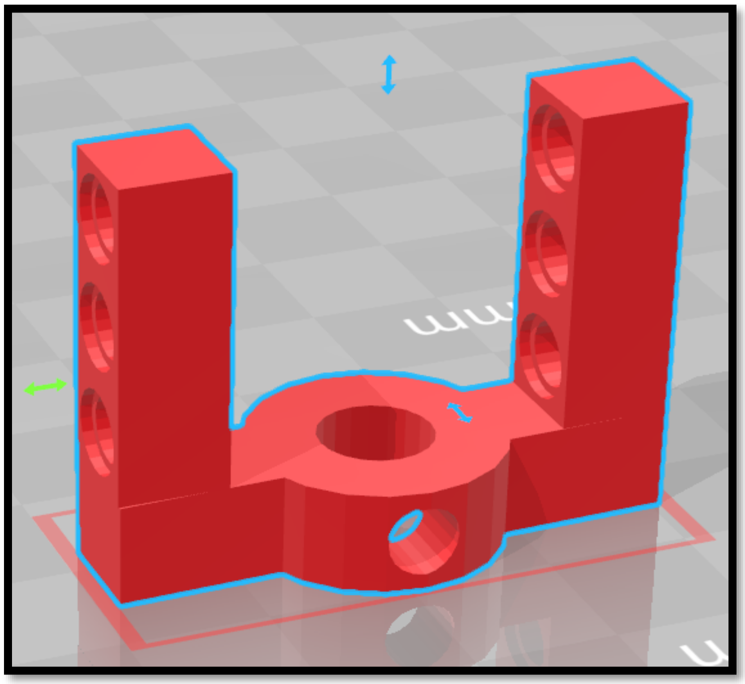


Figure 2. Design of the end effector

The end effector was produced in TUTLab with 3D-printer and it proved to serve its purpose well. The end effector is attached to the robot in a way displayed in figure 3. With one additional hole on either side of the end effector, it would be possible to adjust the position of it afterwards.

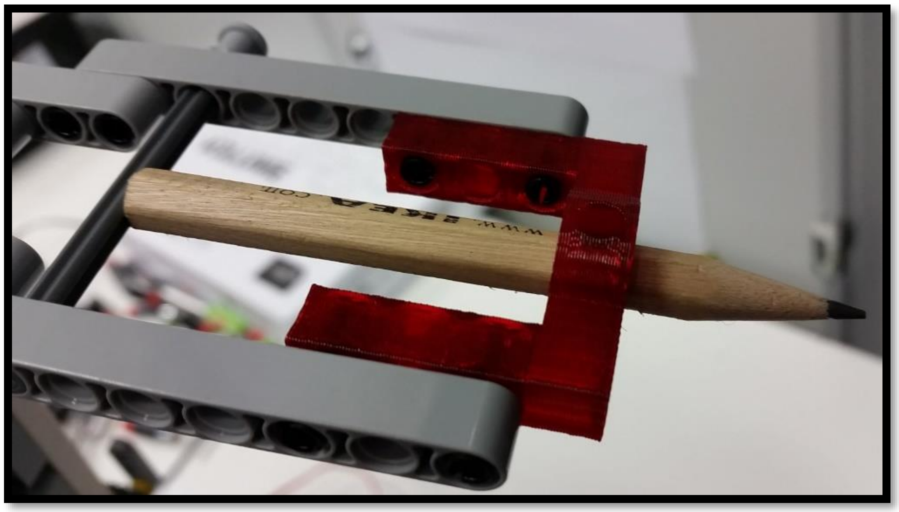


Figure 3. Printed end effector attached to the robot.

Finally, some pictures of the robot manipulator in working positions are given below:

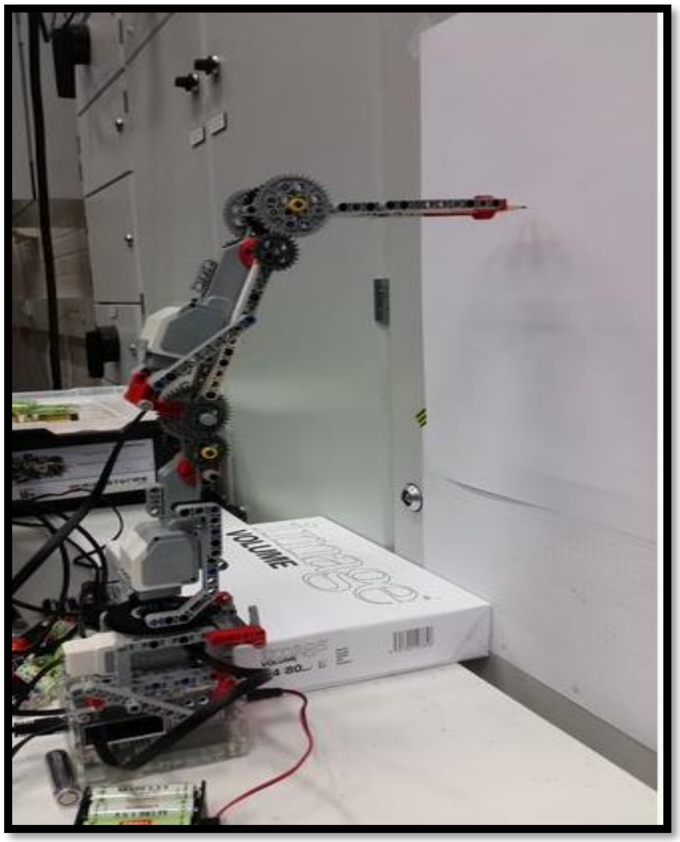


Fig 4: Robot in drawing position1

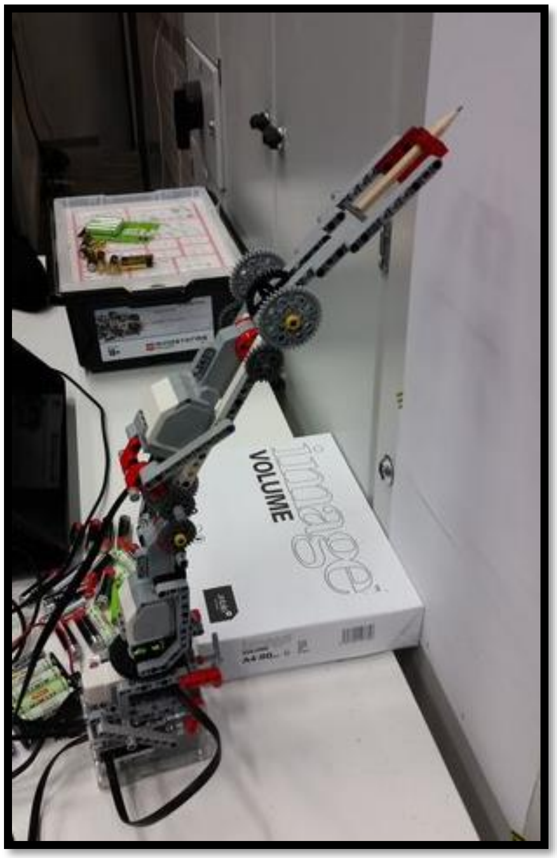


Fig 5: Robot in drawing position 2

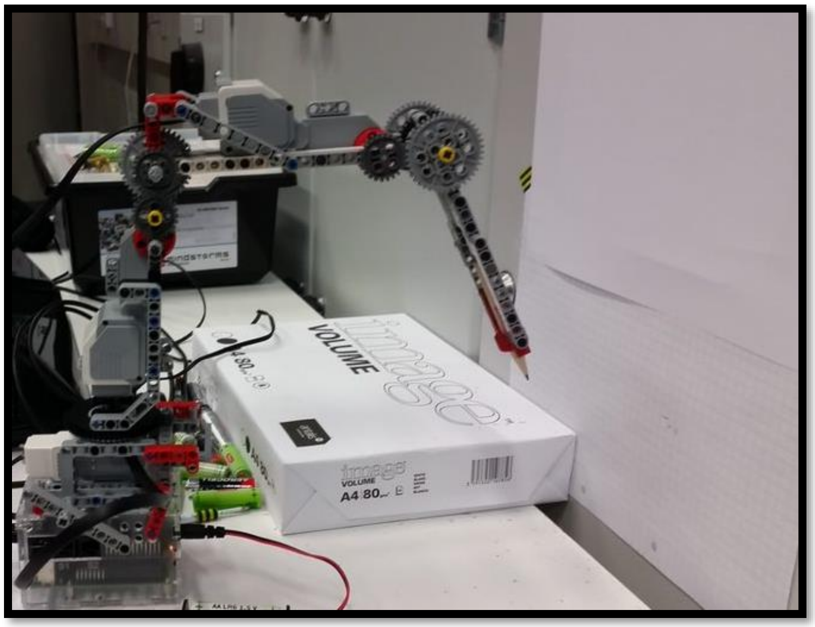
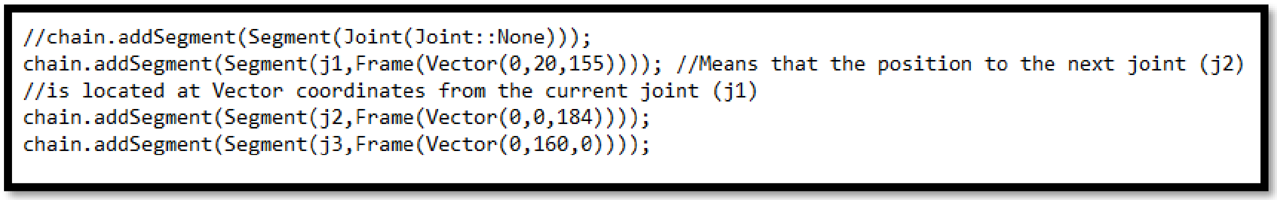


Fig 6: Robot in resting position

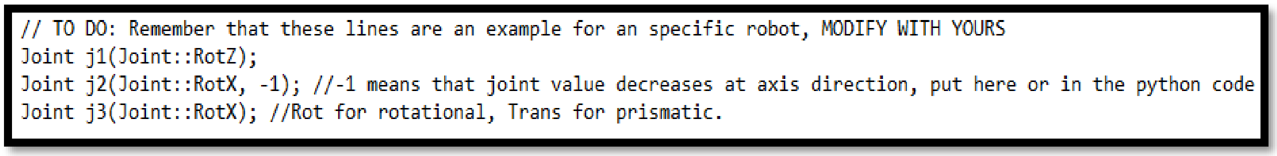
Programming the Robot Manipulator

After building the robot we started to program our manipulator. First, we calculated the length of each link. We selected our base of the robot to be on the first joint which will be the world frame. After calculating the links, we designed our kinematic model and added the measurements in the planner.cpp file. The code is given below.



```
//chain.addSegment(Segment(Joint(Joint::None)));
chain.addSegment(Segment(j1,Frame(Vector(0,20,155)))); //Means that the position to the next joint (j2)
//is located at Vector coordinates from the current joint (j1)
chain.addSegment(Segment(j2,Frame(Vector(0,0,184))));
chain.addSegment(Segment(j3,Frame(Vector(0,160,0))));
```

We also defined the rotation of the axis. According to our design, our first joint rotates around Z axis and the second and the third one around X axis. The second motor rotation is -1 as it decrements with the joint value.



```
// TO DO: Remember that these lines are an example for an specific robot, MODIFY WITH YOURS
Joint j1(Joint::RotZ);
Joint j2(Joint::RotX, -1); //-1 means that joint value decreases at axis direction, put here or in the python code
Joint j3(Joint::RotX); //Rot for rotational, Trans for prismatic.
```

As our robot manipulator is defined now, we set the nrOfJoints and jointOrder according to our design and motors connected to the ports of the BrickPi3. After that we tried to draw a line in a paper by defining xyz value in the task.py file.

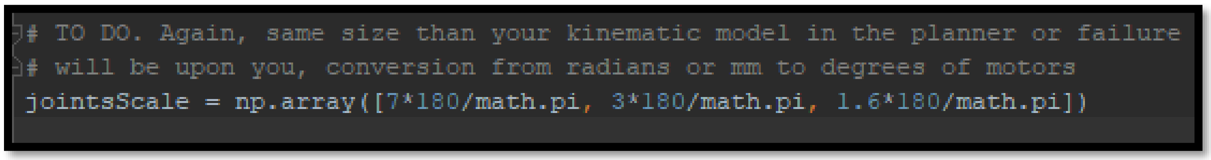
At first the base motor was moving very slowly and the second and the third motor was moving very fast. So, we had to change the jointScale of our robot in the task file.

The joint scale was calculated based on the gear ratio, that was moving the joints. The gear ratio is calculated with formula

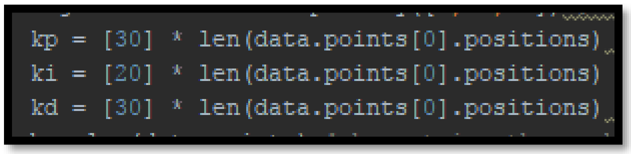
$$R = \frac{N_{out}}{N_{in}}$$

where R is the gear ratio, $Nout$ is the number of teeth in the gear that is driven and Nin the number of teeth in the drive gear. The gear ratio was inserted to the jointScale-array in task.py as is, since it was decided that the direction of the motor is defined in planner.cpp file.

This method worked for the last two joints, but for some reason with the lowest joint, which was rotating around the Z-axis this method did not work, but the jointScale value had to be increased to almost double the calculated value. This might have been caused by the undersized motor, that was used in this joint. The part of the code is given below.


```
# TO DO. Again, same size than your kinematic model in the planner or failure
# will be upon you, conversion from radians or mm to degrees of motors
jointsScale = np.array([7*180/math.pi, 3*180/math.pi, 1.6*180/math.pi])
```

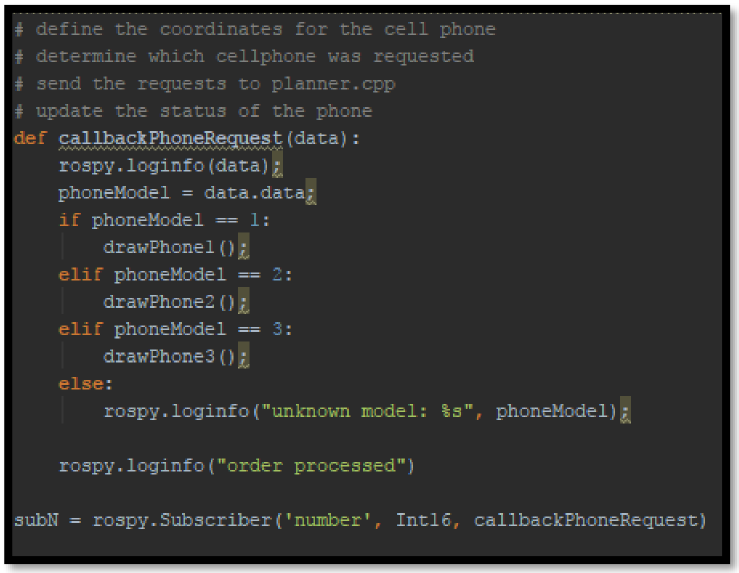
This brought some precision to the robot manipulator and we were able to draw a line. At first the robot was drawing very skewed line. It was not straight at all. Then we changed the speed of the motors and tweaked the values of the PID coefficients Kp, Ki and Kd. After changing some values, we got our robot to draw a solid straight line.


```
kp = [30] * len(data.points[0].positions)
ki = [20] * len(data.points[0].positions)
kd = [30] * len(data.points[0].positions)
```

Then we started drawing the frame of the phone and we gave some coordinates where the robot will move above the point where it will start the drawing from the initial position. From there it will first draw the whole frame and then move to draw the screen and then it will start to draw the keypad. We noticed that when the robot was reaching for the first point, it was missing the exact point. So, it was not reaching the point where it supposed to be because of the change of the coordinate of the first point. Also, the second joint was struggling on the fourth line to move the pencil upwards. Sometimes it even broke the lead of the pencil. After trying many time we shifted our workspace to the wall. It gave better results that the floor. So, we changed our coordinated according to the workspace. It was drawing he horizontal lines were pretty good, but when drawing the vertical lined the arm moved very quickly. Though we changed the speed of the second and third motor, it was of no help. We believe it was due to the weight of the arm.

When we got some satisfactory result, we started to incorporate the matlab code which will publish the request to the task.py file. The requested cellphone was published from matlab script and was caught in ROS by subscriber. The used topic for publishing the number of the phone was called ‘number’, which used datatype std_msgs/Int16 as payload, so it was possible to send only one number at a time to ROS. The subscriber caught the orders and parsed the wanted phone from the message.

The coordinates for each phone was defined in separate functions, so all the subscriber had to do was to call the correct function and it would take care of the rest. Each function would then send requests for the coordinates to the C++ planner. The code of the callback function for subscribing to the matlab script is given below:



```
# define the coordinates for the cell phone
# determine which cellphone was requested
# send the requests to planner.cpp
# update the status of the phone
def callbackPhoneRequest(data):
    rospy.loginfo(data);
    phoneModel = data.data;
    if phoneModel == 1:
        drawPhone1();
    elif phoneModel == 2:
        drawPhone2();
    elif phoneModel == 3:
        drawPhone3();
    else:
        rospy.loginfo("unknown model: %s", phoneModel);

    rospy.loginfo("order processed")

subN = rospy.Subscriber('number', Int16, callbackPhoneRequest)
```

The drawing made by the robot in the testing phase is given below:

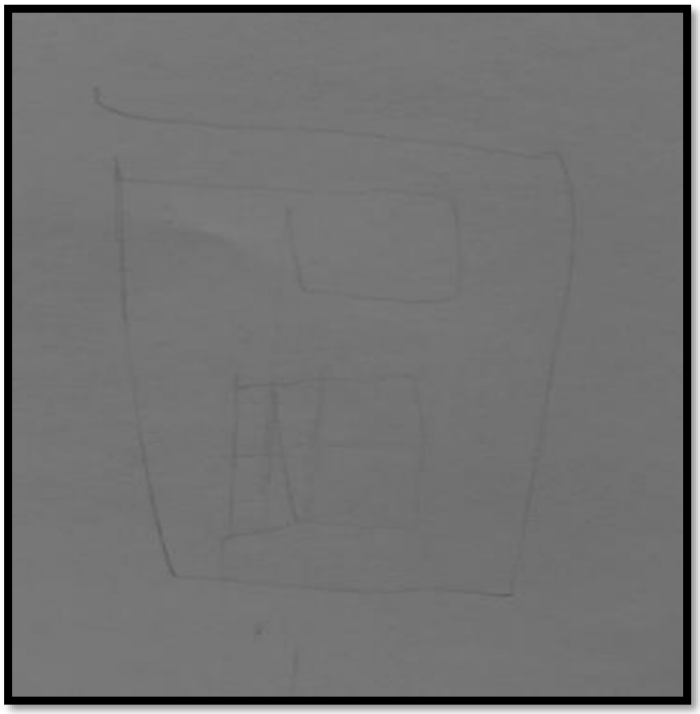


Fig 7: Mobile phone variant 1 drawing by the robot manipulator

Problems Faced

There were certain problems we faced during the assignment. The list of problems is given below:

1. The precision of the Lego motors is not so good. The robot was having hard time reaching the exact position provided by the task file.
2. Calculating the JoinScale was also a problem. According to our calculation we set the jointScale, but the first joint for which we used the weaker motor was struggling to reach any point according to the calculated jointScale. So, we had to almost double the value and then it reached the point, although it was still struggling.
3. We designed our end effector for using pencil. The pencil was attached pretty tightly and was drawing very well. We only had light pencil from ikea which were so light that the drawing was not visible from distance. We could not use any darker pencil, that is why our drawing of the mobile phone looks very light. Also, we decreased the speed of the motor, so that it could reach at least nearby the specified coordinate. That is why the pencil was getting less pressure.
4. Designing the robot with a prismatic joint as the last joint would be helpful to reach the exact point. But at the first stage we were confused that we can use prismatic joint in our robot or not. When the confusion was cleared, it was too late to change the design.
5. We were experimenting so much that it required a lot of battering for the power supply of the Pi. We almost burned 7 set of batteries for this assignment.
6. As we were trying to make the drawing perfect, on the process we could only program the other two variants of mobile phones and could not draw it. But as it drew one type, we are sure that it can also draw the other two variants, but the resultant drawing would be not so perfect.