About the “LAB Robot Arm, LRA-01”

This robot manipulator is the first official robot arm from EAMS LAB, it was designed for AI development purpose in general application such as object detection or object tracking by using additional camera, simple pick and place task, running a complicated motion by motion recording and running repeatable sequence motion by teaching point. According to that the configuration of the robot is 6DOF (Degree of freedom) motion as translation in X-Y-Z direction and rotation about X-Y-Z hand’s coordinate. This robot is an Open Source project and still in development.

Hardware

This robot was designed mainly with aluminum parts, carbon fiber, and standard parts from ROBOTIS. The actuator are Dynamixel robotic servos as **XM540-W270-R** and **XM430-W350-R** which be able to provide feedback signal of present angle / velocity / temperature / and torque (current) during the operation. An overall dimension of the robot is shown in Fig.1.1

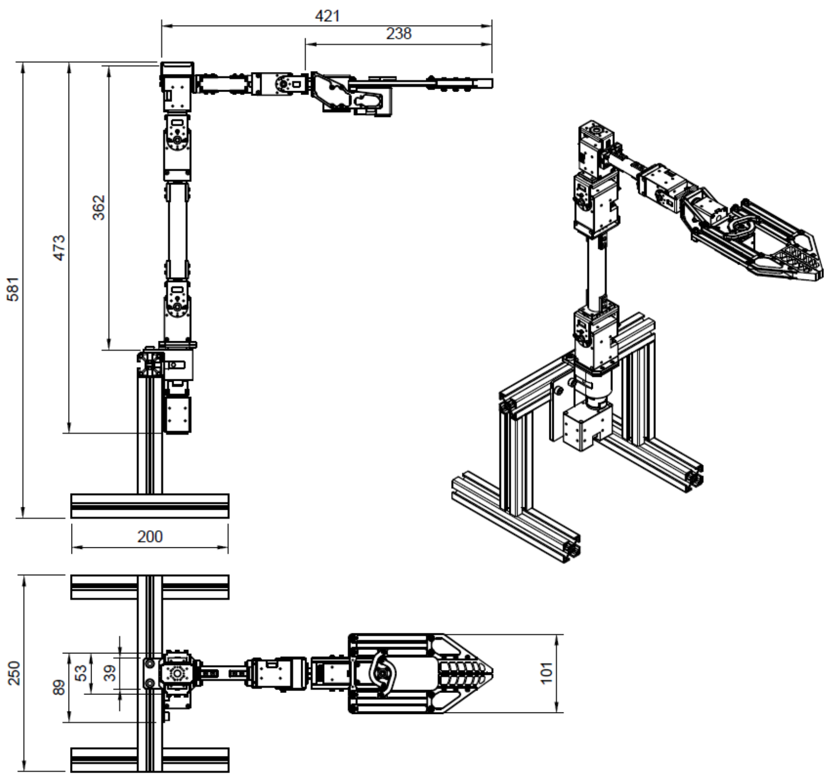


FIgure 1.1 Overall size of the robot arm LRA-01

For more detail of the Dynamixel robotic servos, please check the following link.

http://support.robotis.com/en/product/actuator/dynamixel_x/xm_series/xm430-w350.htm#bookmark26

http://support.robotis.com/en/product/actuator/dynamixel_x/xm_series/xm540-w270.htm#bookmark33

http://support.robotis.com/en/product/auxdevice/interface/u2d2.htm

- ## Mechanical Set Up

The whole set of LAB Robot Arm has included the Robot Arm itself, aluminum profile frame, cables, and some additional parts such bolts as shown in Fig. 1.2. First, let's assemble an aluminum profile frame for a robot stand, you would find 5 pieces of aluminum frame which was marked as A, B and C. A is used as a beam, B is a pole and C is a ground frame. After assemble the frame, assembling the robot arm to the frame as shown in Fig.2. Before attaching the servo no.1 to the frame make sure the robot is in position as shown in Fig.2, and make sure that the servo horn of no.1 is pointing same direction as Fig.3. You can easily notice on small dot on the horn.

IMPORTANT: If the position of servo no.1 is not correctly attached, you would get the wrong kinematics value (X,Y,Z hand's coordinate, and joint angles)

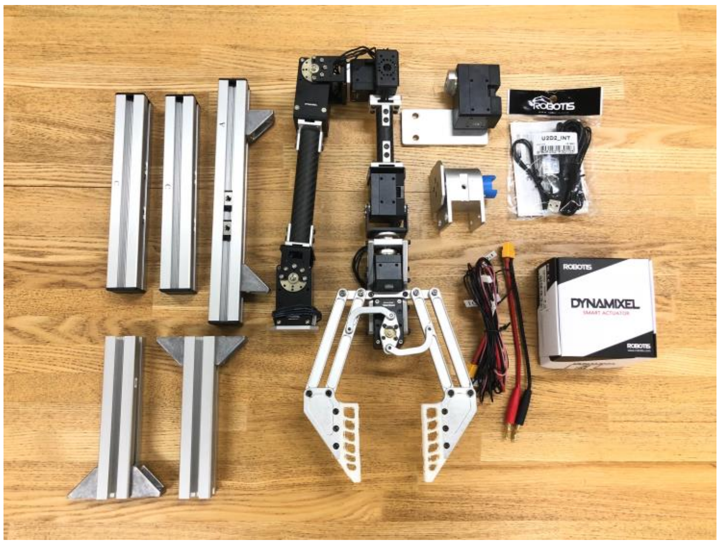
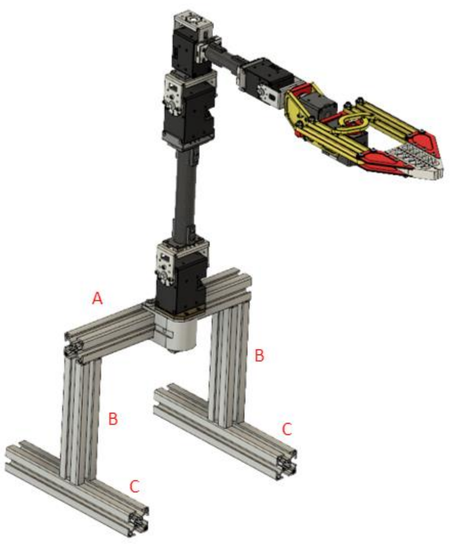


Figure 1.2 LAB Robot Arm Set



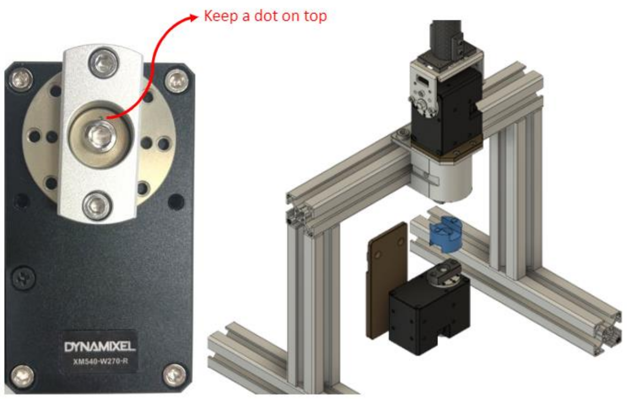


Figure 2 Assemble a frame with robot arm

Figure 3 Set up servo1 and assemble

- ## Electrical Wiring and Plug

Hardware part is almost done. Next, we will move to the electrical part, just a simple wiring, and all you need are laboratory power supply or LiPo battery 3S or 4S and Raspberry Pi (or Linux machine).

The official robot arm from EAMS LAB includes a necessary cable and some electrical part which ready for plug and play. In the box, you will see a cable and stuff as shown in Fig.4

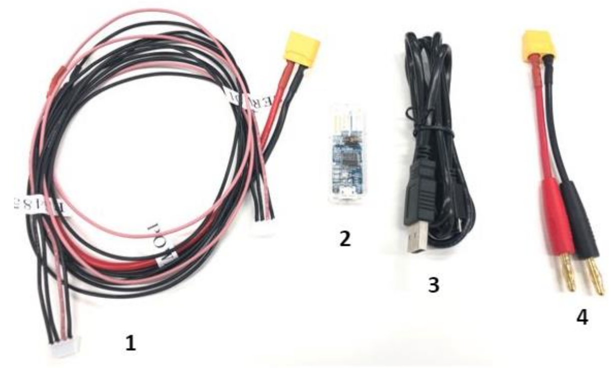


Figure 4 Electrical parts in the box

1. Cable for connecting 1st servo to U2D2 and power supply
2. U2D2 USB communication converter
3. Micro USB cable
4. Banana jack adapter to laboratory power supply

First, we plug the RS485 cable (1) to U2D2 RS485 socket, then another end connects to servo1. If you are using laboratory power supply, you can use a banana jack adapter that come with and plug the male side to the power supply. The servos that used in this robot is using 11.1V - 14.8V (12V is recommend). So make sure that the input voltage is not lower or higher than the specified range. Plug the micro USB to U2D2 and plug USB side to your Raspberry Pi, now your robot is almost ready to go.

Software

ROBOTIS is developing a SDK to easily control the servo with your familiar language and OS, you can check the detail from the link below. And you can find the tutorial of how to set up the computer before running the servo from ROBOTIS YouTube Channel as following.

https://github.com/ROBOTIS-GIT/DynamixelSDK

https://www.youtube.com/channel/UC0M8G5T34THKEgQbdtj1aug/videos

For the LAB Robot Arm, we are using python with Raspberry pi to run all of the script. You can download a class and some example in this GitHub link.

https://github.com/rasheeddo/LabRobotArmOfficial

All of the necessary function is made as a Python class which named as “LabRobotArm.py”. You will find, for instance, Inverse Kinematics, Forward Kinematics, Trajectory Generation, GetXYZ, GoHome, Stand by position, Jog Joint/Linear and many useful functions in there.

• Setting and Update Dynamixel Servo

Before going too far on the code, let make sure that our mechanical parts are ready.

One software that would make sure your servo is working well or not is "R+ Manager", you can download it from this link

http://www.robotis.us/roboplus2/

To check the servo, first plug the USB cable from U2D2 to your computer, turn on a power supply for the servo. Open R+ Manager, if it has an update, please update everything. Then you will see a window of the application shown in Fig.5

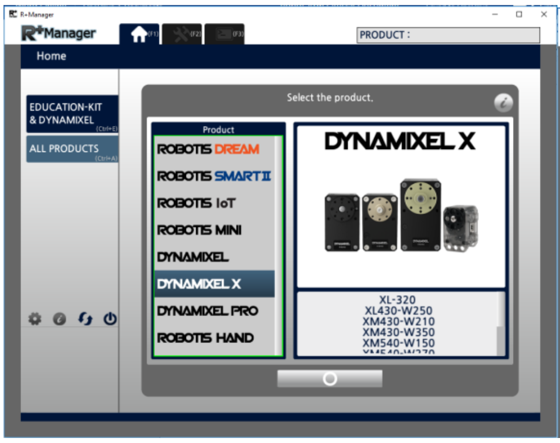


Figure 5

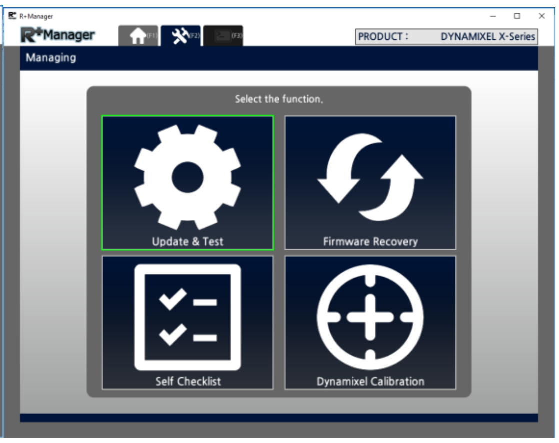
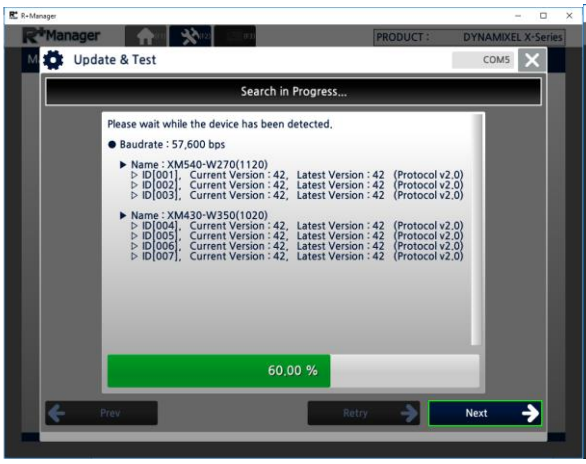
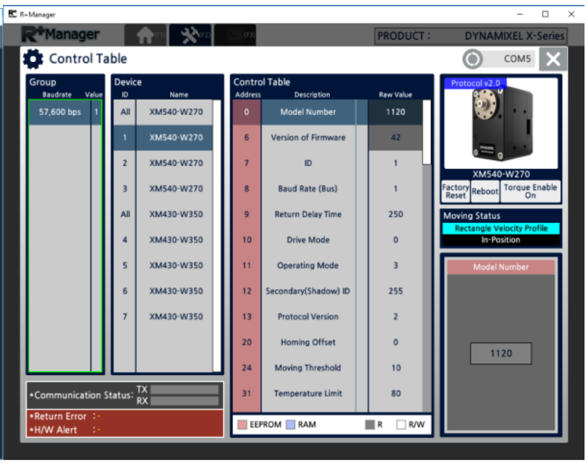


Figure 6

Select DYNAMIXEL X, and click O. Go to the first menu "Update & Test", keep clicking on Next and it will try detecting devices as show in Fig.6 and 7. If it was found, it will show you all of the servo that's connecting. If there is an update, just go ahead. Finally, you will end up at Control Table window as shown in Fig.8. If you want to test driving servo, don't forget to click Torque Enable On first, then find Goal Position address, click on it, and you can manually drive servo by playing on the value in the right lower box.



```
Please wait while the device has been detected.
● Baudrate : 57,600 bps
  ▶ Name : XM540-W270(1120)
    ▷ ID[001],  Current Version : 42,  Latest Version : 42  (Protocol v2.0)
    ▷ ID[002],  Current Version : 42,  Latest Version : 42  (Protocol v2.0)
    ▷ ID[003],  Current Version : 42,  Latest Version : 42  (Protocol v2.0)
  ▶ Name : XM430-W350(1020)
    ▷ ID[004],  Current Version : 42,  Latest Version : 42  (Protocol v2.0)
    ▷ ID[005],  Current Version : 42,  Latest Version : 42  (Protocol v2.0)
    ▷ ID[006],  Current Version : 42,  Latest Version : 42  (Protocol v2.0)
    ▷ ID[007],  Current Version : 42,  Latest Version : 42  (Protocol v2.0)
```



Group Baudrate	Value
57,600 bps	1

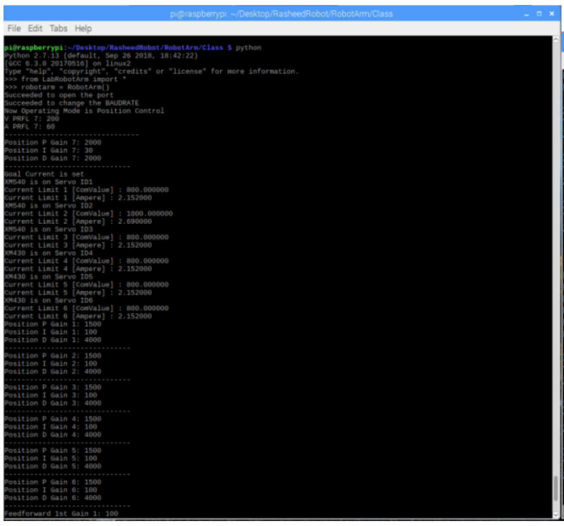
Device ID	Name
All	XM540-W270
1	XM540-W270
2	XM540-W270
3	XM540-W270
All	XM430-W350
4	XM430-W350
5	XM430-W350
6	XM430-W350
7	XM430-W350

Address	Description	Raw Value
0	Model Number	1120
6	Version of Firmware	42
7	ID	1
8	Baud Rate (Bus)	1
9	Return Delay Time	250
10	Drive Mode	0
11	Operating Mode	3
12	Secondary(Shadow) ID	255
13	Protocol Version	2
20	Homing Offset	0
24	Moving Threshold	10
31	Temperature Limit	80

🟥 EEPROM 🟦 RAM ■ R □ R/W

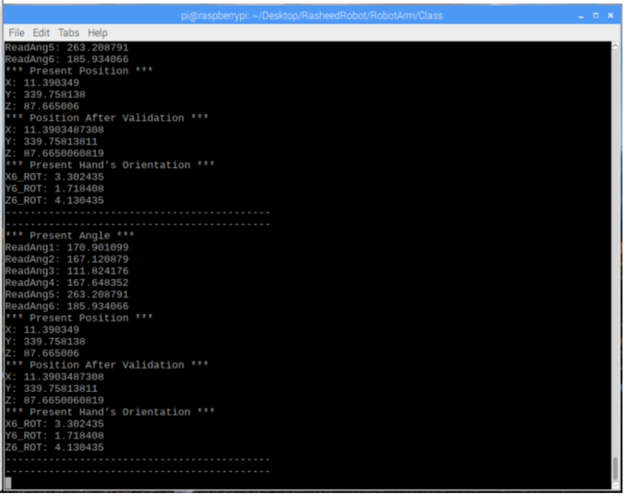
Figure 8

Now, let’s go back to the Python code. I will start with a LabRobotArm.py python class itself. Turn on the power to the robot and go to your Raspberry Pi. Opening the terminal and go to the directory that you downloaded the script. Then, you just type *python* and enter. Next, just import the class by type *from LabRobotArt import ** , and create a class by typing *robotarm = RobotArm()* .The terminal screen should look like the Fig. 9.1. It will initialize the paramter and setup. So you can easily run a function in class by typing *robotarm.”function name”*, for example you can try run *robotarm.RobotArmKinematics()* and you will see that the screen will print all of the necessary variable such as X, Y, Z, gripper coordinate and joint angle in each servo. To terminate the program just press Ctrl+C. Normally, you will have your own program to make the robot move whatever you want. So you can create a new python script which import LabRobotArm class at header like you did in the terminal.



```
pi@raspberrypi:~/Desktop/RasheedRobot/RobotArm/Class $ python
Python 2.7.13 (default, Sep 26 2018, 18:42:22)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from LabRobotArm import *
>>> robotarm = RobotArm()
Succeeded to open the port
Succeeded to change the BAUDRATE
Now Operating Mode is Position Control
V PRFL 7: 200
A PRFL 7: 60
--------------------------------
Position P Gain 7: 2000
Position I Gain 7: 30
Position D Gain 7: 2000
------------------------------
Goal Current is set
XM540 is on Servo ID1
Current Limit 1 [ComValue] : 800.000000
Current Limit 1 [Ampere] : 2.152000
XM540 is on Servo ID2
Current Limit 2 [ComValue] : 1000.000000
Current Limit 2 [Ampere] : 2.690000
XM540 is on Servo ID3
Current Limit 3 [ComValue] : 800.000000
Current Limit 3 [Ampere] : 2.152000
XM430 is on Servo ID4
Current Limit 4 [ComValue] : 800.000000
Current Limit 4 [Ampere] : 2.152000
XM430 is on Servo ID5
Current Limit 5 [ComValue] : 800.000000
Current Limit 5 [Ampere] : 2.152000
XM430 is on Servo ID6
Current Limit 6 [ComValue] : 800.000000
Current Limit 6 [Ampere] : 2.152000
Position P Gain 1: 1500
Position I Gain 1: 100
Position D Gain 1: 4000
------------------------------
Position P Gain 2: 1500
Position I Gain 2: 100
Position D Gain 2: 4000
------------------------------
Position P Gain 3: 1500
Position I Gain 3: 100
Position D Gain 3: 4000
------------------------------
Position P Gain 4: 1500
Position I Gain 4: 100
Position D Gain 4: 4000
------------------------------
Position P Gain 5: 1500
Position I Gain 5: 100
Position D Gain 5: 4000
------------------------------
Position P Gain 6: 1500
Position I Gain 6: 100
Position D Gain 6: 4000
------------------------------
Feedforward 1st Gain 1: 100
```

Figure 9.1



```
ReadAng5: 263.208791
ReadAng6: 185.934066
*** Present Position ***
X: 11.390349
Y: 339.758138
Z: 87.665006
*** Position After Validation ***
X: 11.3903487308
Y: 339.75813811
Z: 87.6650060819
*** Present Hand's Orientation ***
X6_ROT: 3.302435
Y6_ROT: 1.718408
Z6_ROT: 4.130435
--------------------------------------------
--------------------------------------------
*** Present Angle ***
ReadAng1: 170.901099
ReadAng2: 167.120879
ReadAng3: 111.824176
ReadAng4: 167.648352
ReadAng5: 263.208791
ReadAng6: 185.934066
*** Present Position ***
X: 11.390349
Y: 339.758138
Z: 87.665006
*** Position After Validation ***
X: 11.3903487308
Y: 339.75813811
Z: 87.6650060819
*** Present Hand's Orientation ***
X6_ROT: 3.302435
Y6_ROT: 1.718408
Z6_ROT: 4.130435
--------------------------------------------
--------------------------------------------
```

Figure 9.2

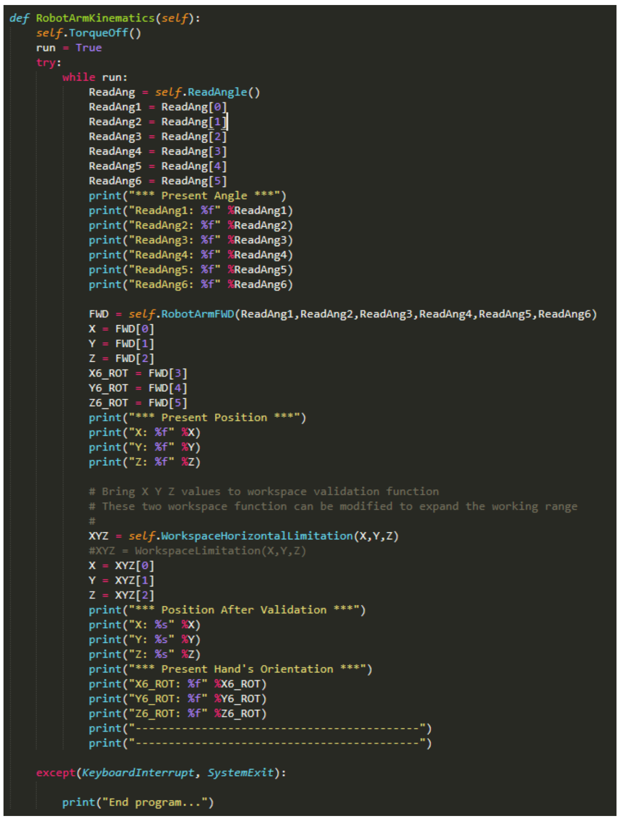
- Kinematics.py

This section I will make another script to run a function in LabRobotArm class. First, try running python code Kinematics.py, if there is no error message and the power already on, you would see a terminal window printing like Fig.9.2, which is same as we did before.

The servo will not be driven in this program, so you can bring a robot arm, move around in the position that you want.

You can open the LabRobotArm script and take a look inside there how it works. The script is shown in Fig. 9.3. The robot is in TorqueOff mode, so that’s why you can easily move the robot by your hand. The main program is running in the while run: loop. The angle of each servo was read and printed out, then those value will be used to calculate a forward kinematics function. The output of this function shows X,Y,Z position and orientation of the gripper coordinate. After that you will see WorkspaceHorizontalLimitation() which is used for checking the X,Y,Z gripper’s position when orientation is keeping parallel to the ground. And WorkspaceLimitation() which is used for checking the all of the reachable point in working envelope. In the example, I used only

WorkspaceHorizontalLimitation() because our AI developer needs a camera to stay horizontal all the time. You can change / comment it out and play with these functions. If the position X,Y,Z is out of the work space, the function will return [None].



```python
def RobotArmKinematics(self):
    self.TorqueOff()
    run = True
    try:
        while run:
            ReadAng = self.ReadAngle()
            ReadAng1 = ReadAng[0]
            ReadAng2 = ReadAng[1]
            ReadAng3 = ReadAng[2]
            ReadAng4 = ReadAng[3]
            ReadAng5 = ReadAng[4]
            ReadAng6 = ReadAng[5]
            print("*** Present Angle ***")
            print("ReadAng1: %f" %ReadAng1)
            print("ReadAng2: %f" %ReadAng2)
            print("ReadAng3: %f" %ReadAng3)
            print("ReadAng4: %f" %ReadAng4)
            print("ReadAng5: %f" %ReadAng5)
            print("ReadAng6: %f" %ReadAng6)

            FWD = self.RobotArmFWD(ReadAng1,ReadAng2,ReadAng3,ReadAng4,ReadAng5,ReadAng6)
            X = FWD[0]
            Y = FWD[1]
            Z = FWD[2]
            X6_ROT = FWD[3]
            Y6_ROT = FWD[4]
            Z6_ROT = FWD[5]
            print("*** Present Position ***")
            print("X: %f" %X)
            print("Y: %f" %Y)
            print("Z: %f" %Z)

            # Bring X Y Z values to workspace validation function
            # These two workspace function can be modified to expand the working range
            #
            XYZ = self.WorkspaceHorizontalLimitation(X,Y,Z)
            #XYZ = WorkspaceLimitation(X,Y,Z)
            X = XYZ[0]
            Y = XYZ[1]
            Z = XYZ[2]
            print("*** Position After Validation ***")
            print("X: %s" %X)
            print("Y: %s" %Y)
            print("Z: %s" %Z)
            print("*** Present Hand's Orientation ***")
            print("X6_ROT: %f" %X6_ROT)
            print("Y6_ROT: %f" %Y6_ROT)
            print("Z6_ROT: %f" %Z6_ROT)
            print("-------------------------------------------")
            print("-------------------------------------------")

    except(KeyboardInterrupt, SystemExit):

        print("End program...")
```

Figure 9.3

When you move the robot to the same position as Figure 2 , ReadAngle1,2,..6 should show around ~180 degree. This is a stand by position for the robot. In the next script you will see that I used a function called StandByPos(), which will drive each servo to 180 degree. If your value doesn't show around 180deg. That's mean you must go and check the servo horn whether it's correctly assembled or not. In this position (stand by position) all of the servo horns must be similar as Fig.3. And that's it for the first script.

• HomePositionSetting.py

Next, before letting the robot move, there is something that you need to consider about. First, this robot doesn't have a big inertia of the gear or any lock mechanism, so during the motion if the power is suddenly off, the robot will fall down. Same as a starting and ending position, this robot needs some place to sleep (shutdown), before and after doing its job. For example, I usually set the robot arm to sleep on its side frame as shown in Fig. 10.1, and it would be placed on soft foam as shown in Fig. 10.2. The home position to sleep can be set anywhere you want, if you have an external part / frame, you can also set that position to let him calmly sleep there.

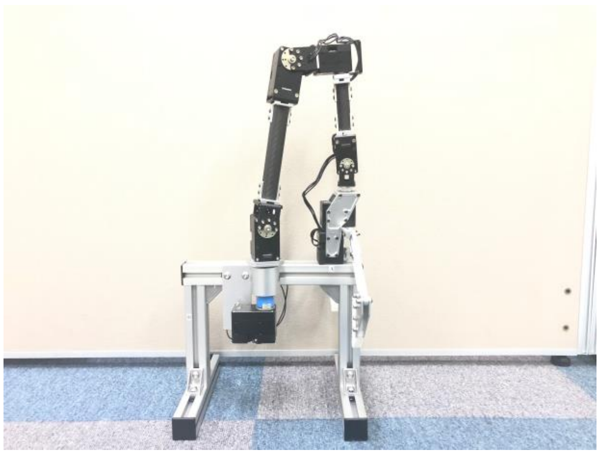


Figure 10.1 Home Position

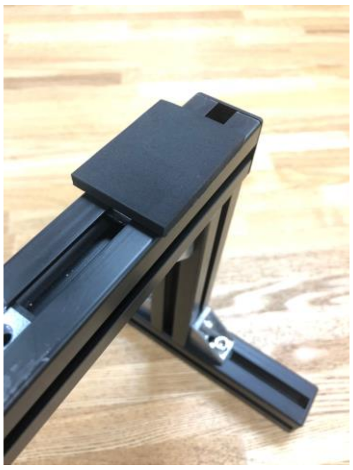


Figure 10.2

To set up a desired home position, go to Terminal window, and run “HomePositionSetting.py”. You may need a game joy stick to press a button for teaching function. I am using one from Logicool as shown in Fig.11. Following the instruction that printed in terminal window finally just copy the new home position and replace on the previous home position it in the __init__() block of LabRobotArm.py class as shown in Fig. 12 and 13.

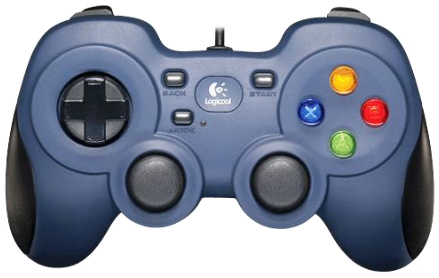
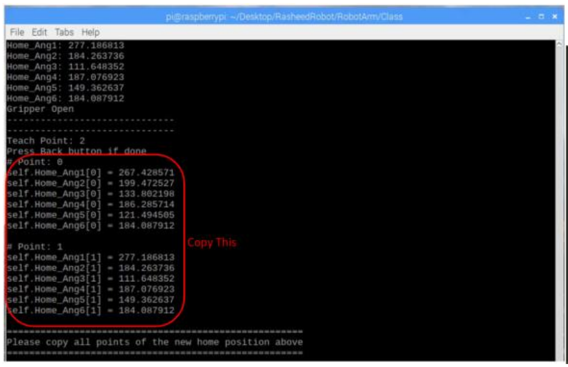
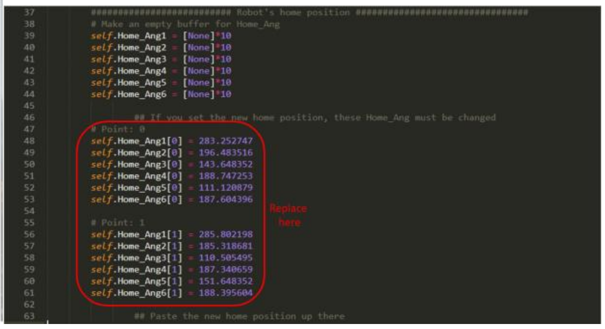


Figure 11 Joy stick



```
Home_Ang1: 277.186813
Home_Ang2: 184.263736
Home_Ang3: 111.648352
Home_Ang4: 187.076923
Home_Ang5: 149.362637
Home_Ang6: 184.087912
Gripper Open
------------------------------
------------------------------
Teach Point: 2
Press Back button if done
# Point: 0
self.Home_Ang1[0] = 267.428571
self.Home_Ang2[0] = 199.472527
self.Home_Ang3[0] = 133.802198
self.Home_Ang4[0] = 186.285714
self.Home_Ang5[0] = 121.494505
self.Home_Ang6[0] = 184.087912

# Point: 1
self.Home_Ang1[1] = 277.186813
self.Home_Ang2[1] = 184.263736
self.Home_Ang3[1] = 111.648352
self.Home_Ang4[1] = 187.076923
self.Home_Ang5[1] = 149.362637
self.Home_Ang6[1] = 184.087912

=====================================================
Please copy all points of the new home position above
=====================================================
```



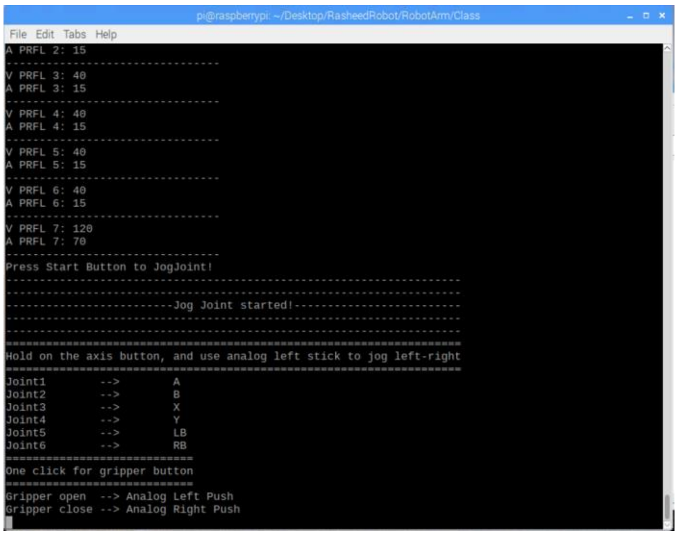
```
        ########################## Robot's home position ###################################
        # Make an empty buffer for Home_Ang
        self.Home_Ang1 = [None]*10
        self.Home_Ang2 = [None]*10
        self.Home_Ang3 = [None]*10
        self.Home_Ang4 = [None]*10
        self.Home_Ang5 = [None]*10
        self.Home_Ang6 = [None]*10

                ## If you set the new home position, these Home_Ang must be changed
        # Point: 0
        self.Home_Ang1[0] = 283.252747
        self.Home_Ang2[0] = 196.483516
        self.Home_Ang3[0] = 143.648352
        self.Home_Ang4[0] = 188.747253
        self.Home_Ang5[0] = 111.120879
        self.Home_Ang6[0] = 187.604396

        # Point: 1
        self.Home_Ang1[1] = 285.802198
        self.Home_Ang2[1] = 185.318681
        self.Home_Ang3[1] = 110.505495
        self.Home_Ang4[1] = 187.340659
        self.Home_Ang5[1] = 151.648352
        self.Home_Ang6[1] = 188.395604

                ## Paste the new home position up there
```

Figure 12

Figure 13

• JogJoint.py

Next, we are going to control the robot by running “JogJoint.py” . Following the instruction on the terminal window, then you would see the text shows which button you are going to use as shown in Fig.14. You would use a joy stick same as Fig.11 to jog the robot. In the script, you will find that the main loop is while startJog:. It takes a signal from joy stick and if there is any axis button pressed, it will start reading angle of that axis and incrementally increase/decrease by a DegIncrement x analog signal (-1 to 1). That means if you fully pushed the stick, the robot would move fastest speed as we set at first. This is just an idea for a jogging algorithm, you can add more advance and complicated calculation to make the robot more stable / precise as you want.

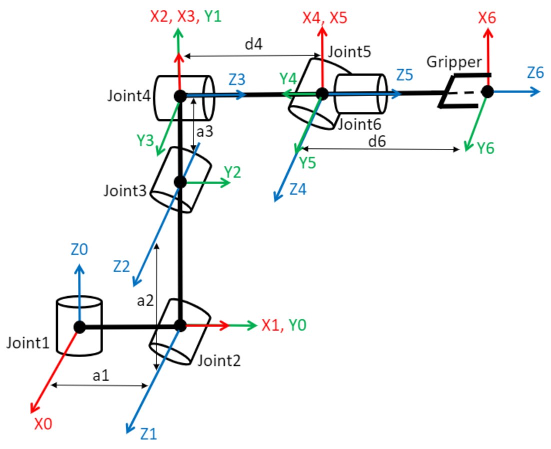


```
A PRFL 2: 15
-------------------------------
V PRFL 3: 40
A PRFL 3: 15
-------------------------------
V PRFL 4: 40
A PRFL 4: 15
-------------------------------
V PRFL 5: 40
A PRFL 5: 15
-------------------------------
V PRFL 6: 40
A PRFL 6: 15
-------------------------------
V PRFL 7: 120
A PRFL 7: 70
-------------------------------
Press Start Button to JogJoint!
-------------------------------------------------------------------
-------------------------------------------------------------------
------------------------Jog Joint started!-------------------------
-------------------------------------------------------------------
-------------------------------------------------------------------
===================================================================
Hold on the axis button, and use analog left stick to jog left-right
===================================================================
Joint1        -->        A
Joint2        -->        B
Joint3        -->        X
Joint4        -->        Y
Joint5        -->        LB
Joint6        -->        RB
============================
One click for gripper button
============================
Gripper open  --> Analog Left Push
Gripper close --> Analog Right Push
```

Figure 14

• JogLinear.py

For an industrial robot, it consists the function of Jog Linear to make the robot translates in each axis as straight line. This is a very useful function to apply in welding, drawing, 3D printing, sorting stuff, translational tracking with camera and more. In order to reach the same level as that robot, I also developed a JogLinear mode which using Inverse and Forward Kinematics calculations in the algorithm. The idea of making it move in straight line is to make it slightly changes the position only in one axis while the other axes are constant. For example, if we were able to find the hand X,Y,Z position and X,Y,Z orientation respect to the ground, then we will be able to increase incrementally just only X position while the other keep constant. If so, the robot would be able to move in that direction only. Some simple right?, the idea is similarly with JogJoint. But the most difficult problem that we are going to face is to find closed-form inverse kinematic equations of 6DOF robot arm. If you are a mechanical or mechatronics guy, you might be able to understand and familiar with the linkage / DH parameters / transformation matrix. Anyway, if you are a computer science / AI developer or more about computer guy, it might not too much difficult for you to understand, because it was just a matrices calculation. The Lab Robot Arm's configuration is based on the same configuration of Fanuc S-900W in the text book of "Robot Analysis The Mechanics of Serial and Parallel Manipulator" by Lung Wen Tsai. The inverse kinematics of this robot's configuration was derived thoroughly there, so you can check it for more detail and see how comes the equation in my code. For a fast explanation, I was re-writing this kinematics diagram from the text book as shown in Fig. 15. The robot has 6 servos refer to 6 joints and one gripper. Each joint has its own coordinate frame, but I suggest you to just focus only on frame 0 (X0,Y0,Z0) or joint 1 and gripper's frame (X6,Y6,Z6).



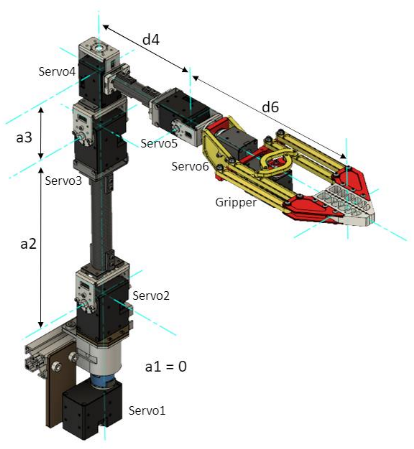


Figure 15 Figure 16

You can see clearly what is the parameters (a1,a2,a3,d4,d6) when compare between the real robot in Fig.16 and kinematics diagram. When we say the robot is moving in X-axis, it means that the gripper's frame (X6,Y6,Z6) is now translating along X0 direction. And when it moves in Y or Z axis, that means it is translating along Y0 and Z0 axes of ground frame. When the robot rotates in X-axis, it means the robot try to keep the position and rotate the whole body around X6 axis, similar as rotation in Y6 and Z6 axis. Fig. 17 shows the axes of ground and gripper coordinate frame.

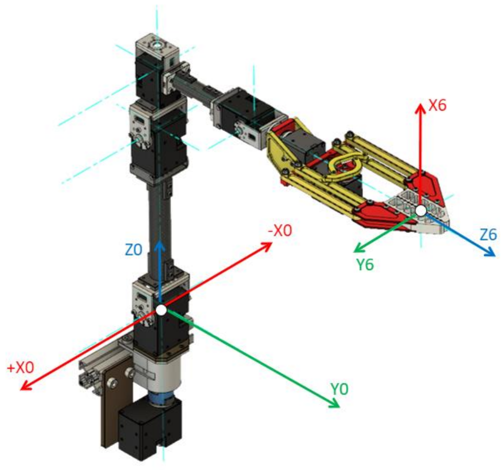


Figure 17

Now you may understand what you are going to do if you need the robot to move in straight line or changing gripper's orientation. You can check on the script of "JogLinear.py", the idea to drive servo is same as JogJoint but we just need to know the present coordinate that why we need a RobotArmFWD() function and to incrementally drive each axis, we use RobotArmINV() to calculate how much angle of the servo needs. You can run the script and follow the instruction on the terminal window.

- ## Dynamics.py

To make the robot run automatically, there are three script that you may need to check on 1.) Dynamics.py 2.) TeachPoint.py and 3.) TeachRecord.py. In Dynamics.py, I have developed a function which called TrajectoryGeneration3(). The idea is if you move the robot with same joint velocity, some longer distance on one servo would make the robot looks not smooth not nature. So, to make this robot looks more natural movement, all of the servos should stop almost at time same time, or another meaning, each servo needs to be adjusted the speed depends on its travel distance. Dynamixel servo has a Velocity / Acceleration profile to make a smooth motion and to avoid suddenly accelerated motion. Firstly, each servo needs to be checked how much distance it needs from current position to goal position. Then, we choose a standard velocity and acceleration, so the idea is the longest distance servo will use this standard vel/acc for profile and the other servos which has shorter distance would be adjusted the speed from standard vel/acc to become slower. You can see the detail of profile equation in ROBOTIS's website below

http://support.robotis.com/en/product/actuator/dynamixel_x/xm_series/xm430-w350.htm#bookmark26

We expect the servo to move with Trapezoid and Triangle profile for long distance, but you can see if the travel distance is too short, it would use Step profile. Each profile would be automatically selected according to which value of velocity / acceleration we picked. In the script of Dynamics.py, in the main loop of while run: , I want the robot to move from 250mm to 0mm to –250mm and so on in X direction. Then come to for loop, you would see the angle was read before to calculate the first travel distance, goal position of each servo were calculated by using RobotArmINV() then find the travel distance, and after that bring those travel distance and vel/acc standard to TrajectoryGeneration3(), it would calculate how much velocity and acceleration for each servo to make it finishes with the same time. After running the servo, MovingStatus() was used to show which kind of profile in each servo. Then to make sure that the servo must completely reach first goal position and not overlapping with new goal position, IsMoving() was used to check whether it was stop or still running. Lastly, just update the current position and run next goal position further. If you run the script, you will see the robot is moving in smooth/natural movement.

- ### TeachPoint.py & TeachRecord.py

To become such a useful robot, it should repeatably do a task from user teaching motion. If you run the script of "TeachPoint.py", this will allow you to teach the robot and because this is not such a giant robot so you can easily teach it by using your hand (no need to jog). The script allow you to let the robot memorizes the position, and play it back repeatably with smooth motion. Moreover, if you need the robot to continuously move, you can run the script of "TeachRecord.py", and it will record all of the motion you give to the robot and run continuously.

Conclusion

For conclusion, this manual is a guide line for LAB Robot Arm kit start from unboxing / setup / and run an example script. The software is developed from Dynamixel SDK and includes a necessary function for running a robot arm. This is an Open Source project, so feel free to add your idea / comment / and update your developed version with us. I would be happy if this robot is getting more useful and getting smarter. Hope you enjoy the LAB Robot Arm.