Robot Module 1: Signs of Life

Amir Samani*

***Abstract*— In Robot Module 1, we build the basic tools required for real life system to test various applications of reinforcement learning (RL). In this module we discuss how to assemble a simple robot, initialize robot's servo motors, how to communicate with servo motors (sending instructions and receiving sensorimotor data), and plotting the sensorimotor data stream (both real-time and offline plotting).**

I. Assemble the Robot

As the first step we need to assemble the robot. Figure 1 shows the a simple sketch of our robot and its connection to power and a computer. While Figure 2 shows the end result after all steps are done.

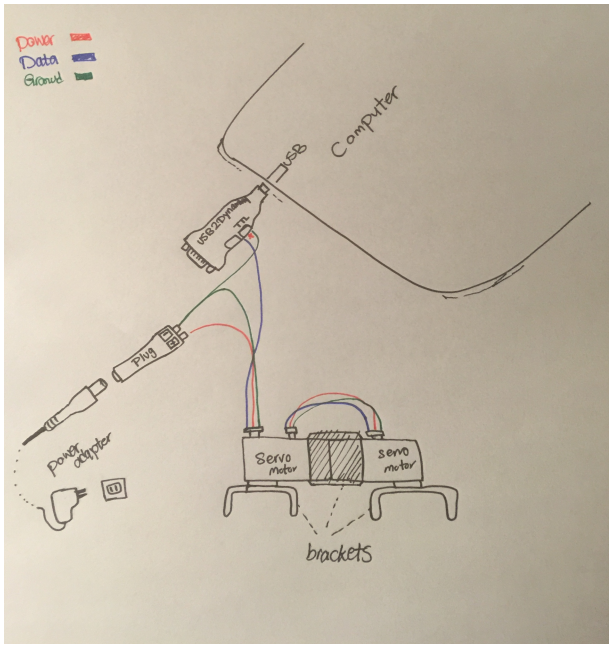

Fig. 1. Simple sketch of how to assemble the robot.

Now we can dive into the details we need to create the robot. We used two AX-12A Servo Motors from Robotis [1]. We then used OF-12S bracket to connect the two servo motors. Figure 3 [2] shows the OF-12S bracket with PHS M2*6 bolts [3] and N1 NUT M2 [4] which are required for assembling the two servo motors together.

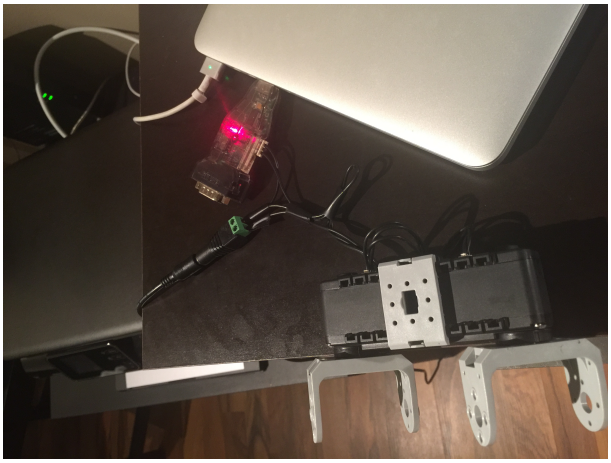

Fig. 2. The final result after the robot is assembled and it is connected to both power and a computer.

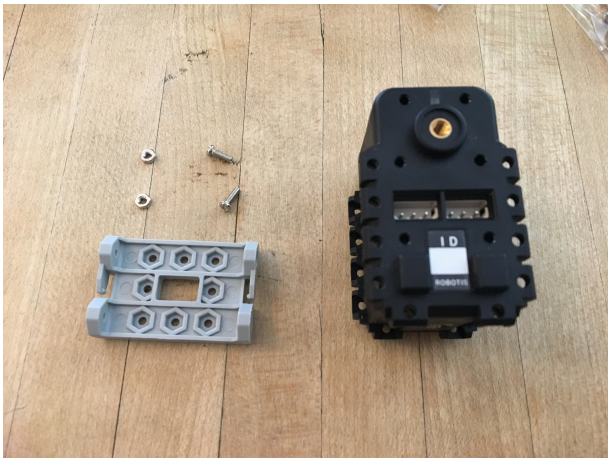

Fig. 3. Servo motor and OF-12S bracket with required nuts and bolts for assembly.

The details to connect the two servo motors using the bracket are shown in Figures 4, 5, and 6. In order to make sure two servo motors are connected body-to-body with minimal room for unwanted movement, we connect the other side of the servo motors together using the same bracket as shown in Figure 7. Now we can connect one OF-12SH bracket to each servo's horn using the PHS M2*6 bolt. The result is shown in Figure 8. Now we have the robot assembled. In the next section we discuss how we can connect the robot to the power and also to a computer for controlling the robot.

*Amir Samani is with Faculty of Computing Science, University of Alberta, Edmonton, Canada samani@ualberta.ca

[1] available at: http://www.robotis.us/ax-12a/

[2] all the following figures about robot assembly and power connection are made by Patrick Pilarski and are available in CMPUT 607 Dropbox.

[3] available at: http://www.robotis.us/bolt-phs-m2-6-20-200pcs/

[4] available at: http://www.robotis.us/n1-nut-m2-400-pcs/

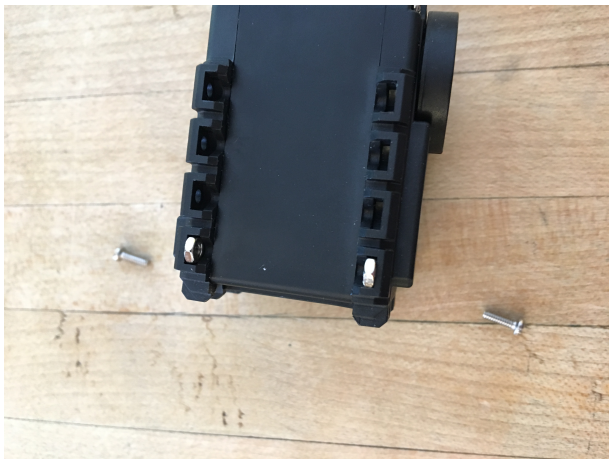

Fig. 4. Pusing the nuts in the designated part of the servo motor.

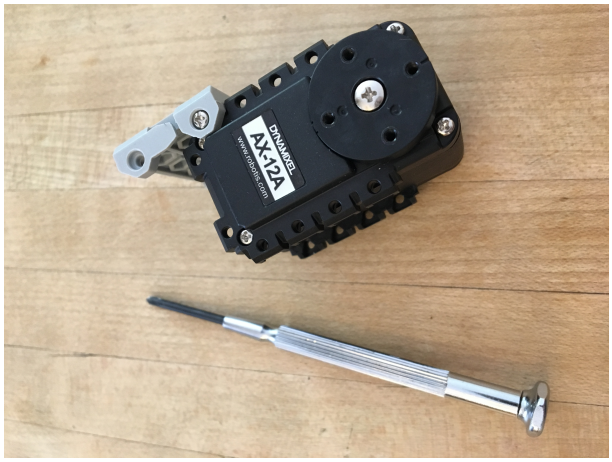

Fig. 5. Connecting the bracket to the servo motor using the bolts.

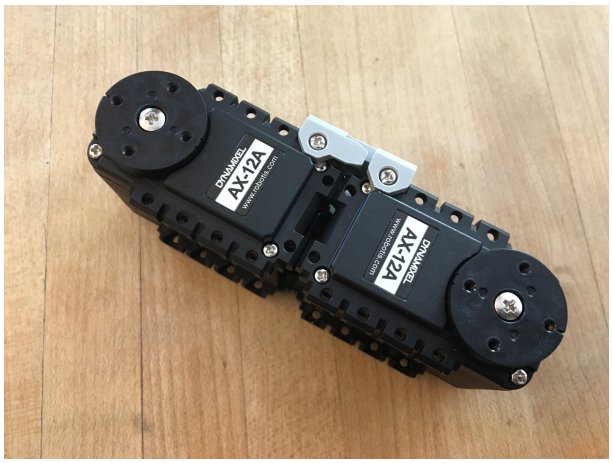

Fig. 6. Connecting the second servo motor to the bracket using similar nuts and bolts.

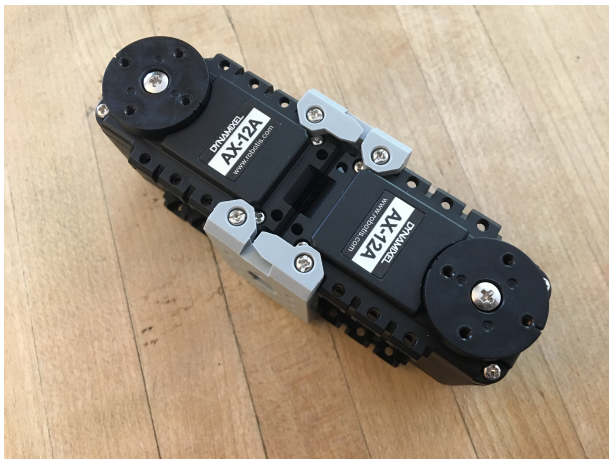

Fig. 7. Connecting the other side of the two servos with the same bracket to minimize the unwanted movements.

II. Connection to Power and Computer

We have one serial connection between the servos, power, and computer. We use 3P cables [5] to make this connection. As shown in Figure 1 and 2, we have a connection between two servos and a connection shared between computer, power, and one of the servos. The connection between two servos are simply made by using a 3P cable. However, for connecting one of the servos to power and computer using a single 3P we need to modify the cable. Figure 9 shows a the plug and a 3P cable. As shown, if we keep the flat side of 3P connector facing up and the pins facing towards point of view, the most right pin is for carrying the data, the middle one is for the power, and the most left pin is for the ground connection. We can mark these pins for convenience. In our case we used blue for the data, red for the power, and black for the ground connection. As 3P cable has 2 ends, one of them are connected to the servo with no modification. The other end is connected to the computer through a USB2Dynamixel [6]. However, for this end we need to make sure the we cut the power line since it should not be connected to the USB2Dynamixel (it is important to note that the USB2Dynamixel devices are very elegant and can be simply damaged or killed by wrong usage). While the data line only needs to be between the servo and USB2Dynamixel, the ground line is the only line shared between the servo, the USB2Dynamixel, and the plug. In order to modify the 3P cable for our use, as shown in Figure 10, we need to remove a part of ground line's skin. As shown in Figure 11, now we attach the power line to the positive side of the plug and the ground line to the negative side of the plug by twisting it. We make sure these connections are secure with the bolts provided in the plug. To summarize, the 3P cable should be similar to Figure 12. The end with power (middle) line detached is the one which is connected to the USB2Dynamixel, the plug is connected to a power adapter as shown in Figures 1 and 2 and the unmodified end of the 3P cable is connected to the servo motor. In the next part we discuss how to initialize the servo motors after they are connected to the computer.

[5] available at: http://www.robotis.us/robot-cable-3p-140mm-10pcs/
[6] available at: http://www.ca.diigiit.com/robotis-usb2dynamixel

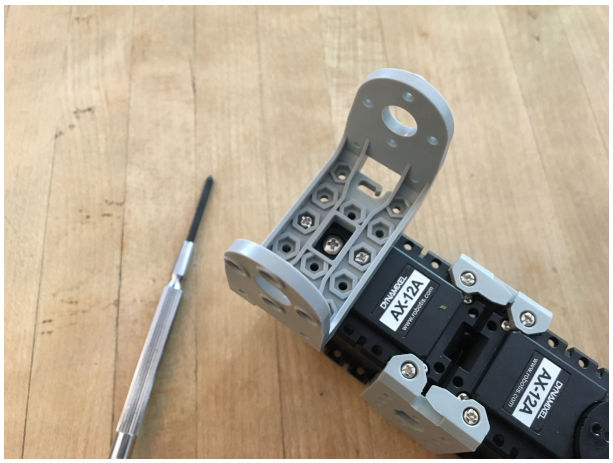

Fig. 8. Attach a OF-12SH bracket to a servo's horn.

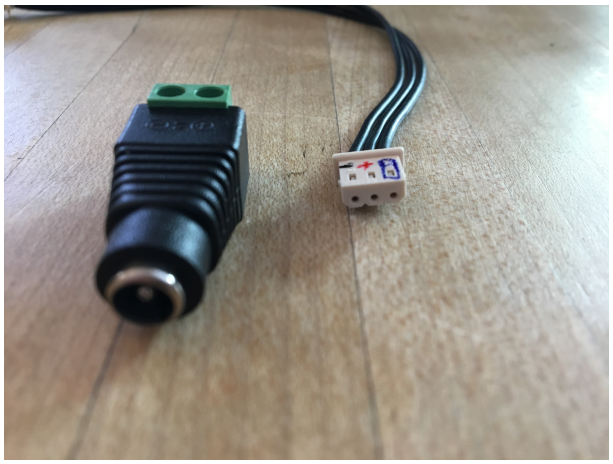

Fig. 9. A plug for the power connection and a 3P cable.

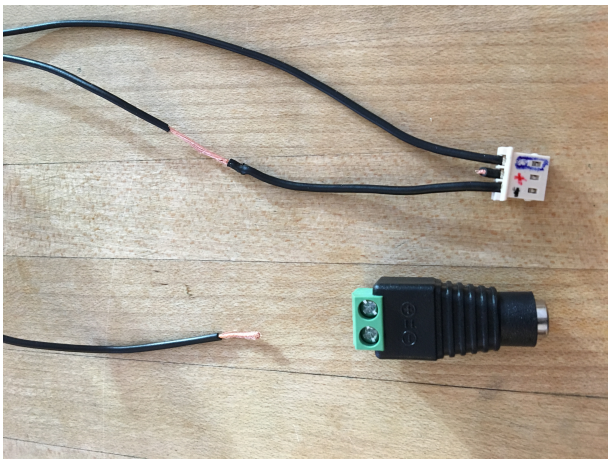

Fig. 10. Removing the middle part of the ground line's skin to connect it the plug.

Fig. 11. Connecting the 3P cable to the plug.

III. Initialize the Servo Motors

Now we have everything ready to connect the robot to the computer. It is important to first connect the robot to the power and then to the USB2Dynamixel since voltage fluctuation can damage the USB2Dynamixel. After we connect the robot to the computer we can use "lib_robotis_hack" [7] to communicate with it. We need to find the USB2Dynamixel name by looking at the "dev" directory. In order to give the servos proper numbers we first connect one of the to the computer. Then in a python 2.7 console we run the following code (one by one, that is why we use python console instead of a script). The "dev_name" argument should be assigned by the address to the USB2Dynamixel. We configured "baudrate" with the value of 1000000 in our settings.

```
from lib_robotis_hack import *
D = USB2Dynamixel_Device(dev_name,baudrate)
s_list = find_servos(D)
s1 = Robotis_Servo(D, s_list[0])
s1.write_id(2)
# then we connect the second servo
s_list = find_servos(D)
s2 = Robotis_Servo(D,s_list[1])
s2.write_id(3)
```

[7] by Travis Deyle, Advait Jain & Marc Killpack (Healthcare Robotics Lab, Georgia Tech.)

Now the servos are configured and we can communicate with them according to their id numbers (2 and 3). The reason behind this configuration is, as both servo motors share the same serial bus they need to have unique id numbers to be distinguishable when receiving instructions or sending back sensorimotor data). In the next step we demonstrate how you can send instructions to the servos and also read their sensorimotor data stream.

IV. Communicating with Servo Motors

After all the configuration, now we are able to send instructions to the servos and read their their sensorimotor data stream in order to visualize it or choose the next action based on these streams of data. However, in this section we are going to implement a simple policy for our servo motors and visualize the sensorimotor data stream in real time. Assuming we have "s1" and "s2" configured as our

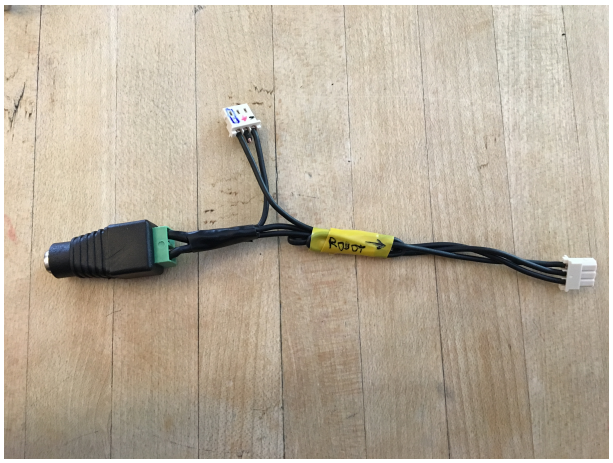

Fig. 12. Final result after modifying the 3P cable.

servos in the code we can simply move them by two methods from "lib_robotis_hack", *move_angle* and *move_encoder*. *Move_angle* is for changing the angle of a servo's horn to reach a certain angle (radians), while has the option to be blocking or non-blocking. By blocking, we freeze the current thread until the required movement is achieved by the servo. On the other hand, *move_encoder* moves to fixed registered points from 0 to 1023 (1024) points. The *move_angle* method is using *move_encoder* and basically maps the angle to a specific encoder. The complete code for this project is available in our Github [8]. A simple policy for our servo 1 with servo id of 2 is as following.

```
def servo1_policy( threadName, D):
  servo = Robotis_Servo(D,2)
  ang = 0.0
  dir = 1
  while (True):
    servo.move_angle(ang, blocking=False)
    ang += dir * 0.1
    if ang >= 1.0 or ang <= -1.0:
      dir *= -1
        time.sleep(0.05)
```

Basically, here we have a thread for controlling the servo 1, and we follow a very simple policy. We start by first moving to the angle 0 and each time we move positive 0.1 radians until we get to the angle of 1.0. Then we change our direction and move negative 0.1 radians at each time until we reach the angle of -1.0 and again change our direction. There is no specific reason behind this policy as we only want to experiment communicating with the servo motors and plot the sensorimotor data stream. The next step is to plot the sensorimotor data stream in real time. There are different sensorimotor data that we can ask the servos the get, for instance, the angle, load, voltage, and temperature. The following code is asking this information from servo1 and then plot them. The plotting code is available in our Github but was developed by Jaden Travnik.

[8]https://github.com/Amir-19/M1-Signs_Of_Life

```
d1 = DynamicPlot(window_x = 100,
    title = 'datastream servo 1',
    xlabel = 'time_step', ylabel= 'value')

d1.add_line('s1 Voltage')
d1.add_line('s1 Load')
d1.add_line('s1 Angle * 100')
d1.add_line('s1 Temp')
i = 0
while True:

  #reading data for servo 1
  read_all = [0x02, 0x24, 0x08]
  data = s1.send_instruction(read_all,
          s1.servo_id)
  voltage = data[6] / 10.0
  temperature = data[7]
  load = sum(data[4:6]) / 2.0

  # calculate the angle for servo 1
  ang = (data[1]*256 +data[0] - 0x200)
          * math.radians(300.0) / 1024.0

  # update the plot for servo 1
  d1.update(i, [voltage, load,
          ang*100,temperature])

  # saving for offline plotting or analysis
  servo1_data.append([i, voltage, load,
          ang*100,temperature])
  i += 1
```

As you can see in the code above, in order to read these sensorimotor information, we need to send specific instruction to the desired servo. Since the connection is serial, it is better to minimize the number of usage of this serial bus, this is why we try to read all the information that we want in a single use of instruction. We also save the data that we get in order to offline plotting or further analysis or even for logging.

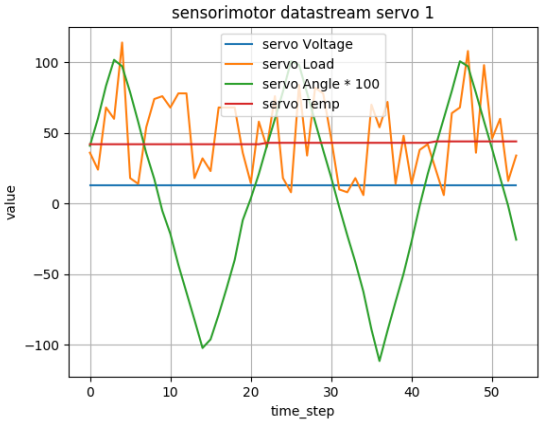

Fig. 13. Offline plot after experiment for the first 50 time steps of mentioned simple policy.

V. Mirroring Policy

The Github repository also contains the code for mirroring policy. In mirroring policy, we basically disable the torque for the servo 1 and let the angle of the horn be changeable by applying third party force (like a human rotating the bracket attached to the horn) with no resistance by the servo. The policy for servo 2 is to mirror the angle of servo 1. The interesting thing about this policy is that, we no longer do a random behavior but we select actions based on the sensorimotor information provided by the robot to the computer. The following code is the policy of servo 2.

```
def servo2_policy( threadName, D):
  servo = Robotis_Servo(D,3)
  s1 = Robotis_Servo(D,2)
  while (True):
    ang = s1.read_angle()
    s2.move_angle(ang, blocking=False)
```

VI. Summary

To summarize, we started by the step-by-step instructions of building our robot and its connection to power and computer. Then we discussed how we initialize the servo motors by connecting them by giving them unique id number to be accessible via the serial bus. Moreover, we demonstrated how to communicate with the servo motors, sending instructions and receiving sensorimotor data stream. Also, we covered the real-time plotting and offline plotting. Last but not least, we discussed implementing the mirroring policy as an extension to our simple policy, in which we include the sensorimotor information to select the next action.