Portland State
UNIVERSITY

ECE – 579

PORTLAND STATE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Intelligent Robotics II

Project: INTELLIGENT BIPED ANIMATION

Guidance: Dr. Marek Perkowski

Adel Alkharraz (adel5@pdx.edu)
Aniruddha Shahapurkar (anirud@pdx.edu)

Date: 03/16/2020

INTRODUCTION:

- The HR-OS5 Robot is for researchers and educators who are looking for a solid, high-performance, open-source humanoid robot platform to develop on.
- Each HR-OS5 Research Humanoid is assembled with a pre-loaded and configured operating system so that developers can dive right in to getting their robot running.

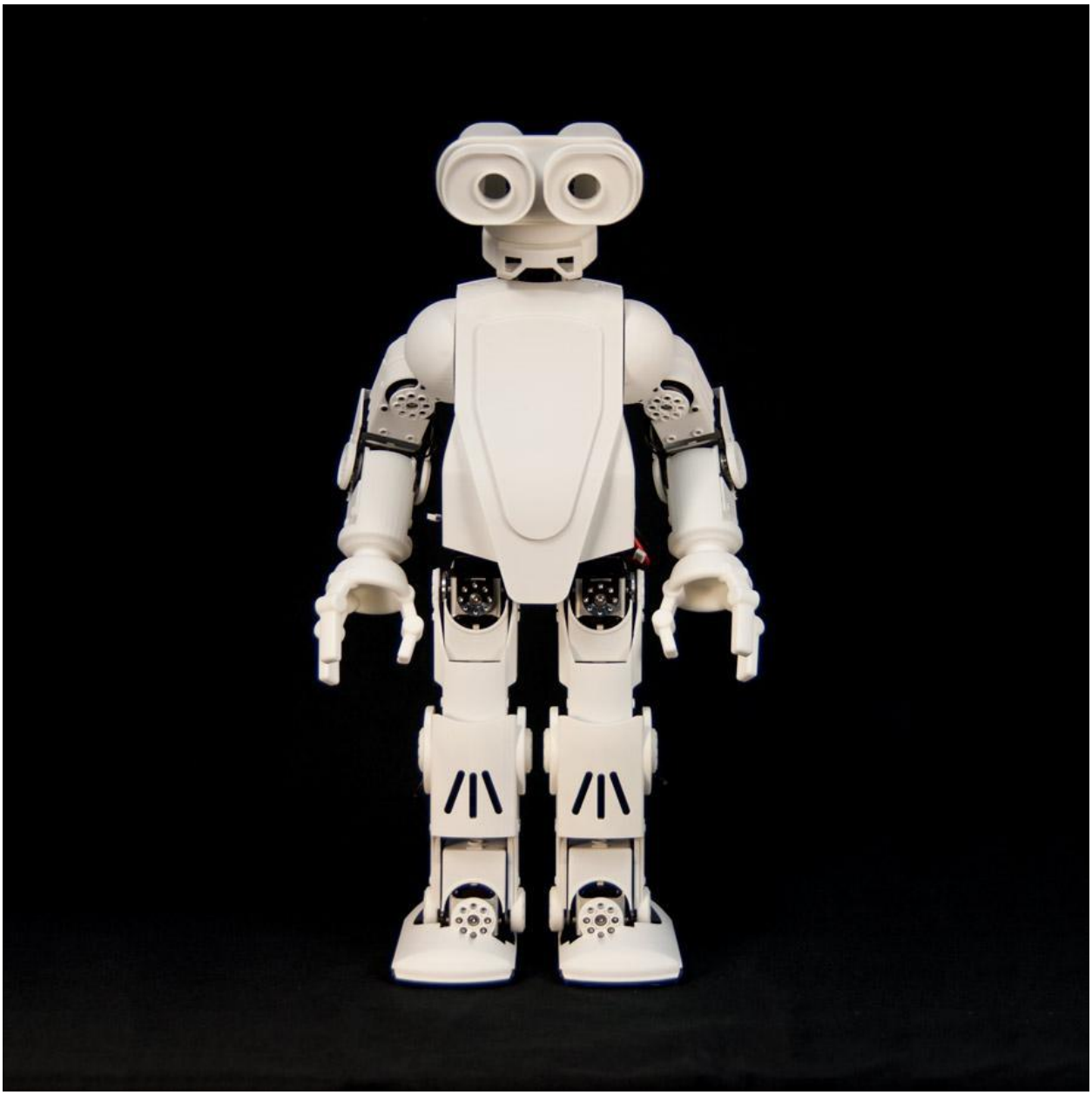


Fig.1 HR-OS5

Block Diagram

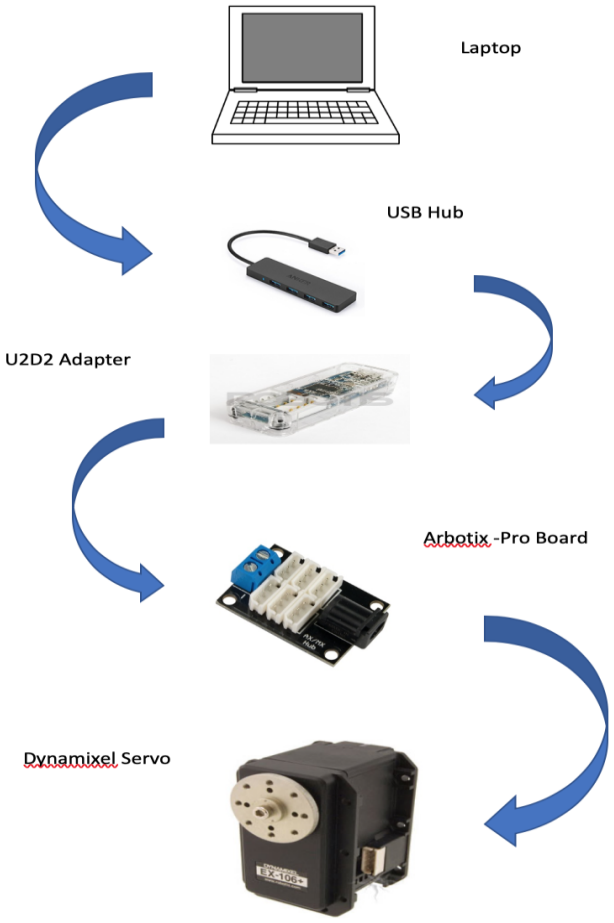


Fig. Block Diagram

- Figure shows the block diagram for our project.
- The key components which we need to setup while working on the project are
    - U2D2 adapter,
    - Arbotix-Pro board,
    - A USB hub
    - Dynamixel Servo motors

HARDWARE

- The HR-OS5 Humanoid's internal endoskeleton is manufactured from 5052 aircraft aluminum and is completely modular and expandable.
- The outer shell of the robot is designed to be 3D printed, which means modifications and aesthetic tweaks are very easy and affordable.
- The robot features:
    - 12 MX-106T DYNAMIXEL servos,
    - 6 MX-64T DYNAMIXEL servos, and
    - 2 MX-28T DYNAMIXEL servos from ROBOTIS.
- The on-board CPU is comprised of an Intel NUC D54250WYK, which features an Intel Core i5-4250U 4th Generation Haswell chipset, 4GB of DDR3 RAM (up to 16gb expansion), and a 32GB SSD for internal storage.
- Wireless connectivity options are available in:
    - Xbee,
    - Wifi 802.11N, and
    - Bluetooth.
- Onboard IO for expansion and development include:
    - 4x USB 3.0 & 2x USB 2.0 ports,
    - HDMI & Display Port video output,
    - Gigabit Ethernet, SATA port,
    - 2x mini-PCIe slots (used for SSD & Wifi/Bluetooth), and
    - Up to 8-channel audio.

- The sub-controller used to communicate with the Dynamixels is the ROBOTIS CM-730 Cortex-M3 based microcontroller, which will eventually be replaced by our own Arbotix-PRO.
- Onboard LiPo batteries provide 4S 14.8v 4000mAh of power, which produces approximately 30-60 minutes of runtime per charge.

SOFTWARE

- The robot is offered in two flavors of Linux- Ubuntu 14.04 LTS for developers who wish to take advantage of a full-featured desktop OS, or Yocto Project Poky distribution OpenEmbedded Linux which has a custom 21C Robots layer to allow for unified support across many different CPU configurations and hardware.
- This framework is being actively developed and is based upon the open-source and highly featured Darwin-OP framework from ROBOTIS and Virginia Tech University.
- We will continue to refine and expand this software, providing multiple control interface solutions as well as a REST based API so that the robot can be connected to ROS & the Intel XDK for cross-platform application development, as well as a variety of different software environments.
- The goal of the API is to easily expose higher level functions of the robots, so that developers can bring the robots to life without having an in-depth knowledge of the more advanced lower level functions that make the robot move and walk.

Like the hardware, all of the software will be available completely open-source and available for anyone to download.

- The HR-OS5 is a robotic research development platform aimed at researchers, university programs, and high-level software developers.
- An advanced working of knowledge of Linux and C++ is recommended to do development on the software framework.
- We are working to bring a high level API to developers so that the higher level functions of the robot are more easily accessible.
- In the coming weeks & months, we will be publishing all of our current and available documentation on the platform, as well as continuing to document and develop more information and applications.
- As this is a very advanced research platform, we encourage potential developers to contact us to answer any questions they may have.
- We hope to continue to forge an ever expanding relationship with users, developers, and Makers.

Important Tutorials Successfully Completed

ROS WIKI Tutorials

A. Beginner Level:

1) Installing and Configuring ROS environment:

- This tutorial walks us through installing ROS and setting up the ROS environment on our computer.
- There is more than one ROS distribution supported at a time. We got two options which are shown below.

Fig. Ros Distributions

- Out of these two we used ROS KINETIC as it provides better support while working with Dynamixell servos.
- Managing ROS environment:
- Environment setup files are generated for us, but can come from different places:

  - ROS packages installed with package managers provide setup.*sh files
  - rosbuild workspaces provide setup.*sh files using tools like rosws
  - Setup.*sh files are created as a by-product of building or installing catkin packages

- Create a ROS Workspace:
  We created the ROS workspace with the help of following commands

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

- Once the ROS environment was setup, we continued with ROS filesystem tutorial.

2) Navigating the ROS filesystem:

- This tutorial introduces ROS filesystem concepts, and covers using various important tools like:
    - roscd,
    - rosls, and
    - rospack.
- rospack allows us to get information about packages. In this tutorial, we are only going to cover the find option, which returns the path to package.
- roscd is part of the rosbash suite. It allows us to change directory (cd) directly to a package or a stack.
- roscd can also move to a subdirectory of a package or stack.

3) Creating a ROS Package

- This tutorial covered usage of roscreate-pkg or catkin to create a new package, and rospack to list package dependencies.
- For a package to be considered a catkin package it must meet a few requirements:
- The package must contain a catkin compliant package.xml file.
- That package.xml file provides meta information about the package.
- The package must contain a CMakeLists.txt which uses catkin.
- If it is a catkin metapackage it must have the relevant boilerplate CMakeLists.txt file.
- Each package must have its own folder
- This means no nested packages nor multiple packages sharing the same directory.
- The simplest package might have a package which looks as follows:

```
my_package/
  CMakeLists.txt
  package.xml
```

4) Building a ROS Package:

- This tutorial covers the toolchain to build a package.
- For this we need the CMake workflow which breaks down as follows:

```
# In a CMake project
$ mkdir build
$ cd build
$ cmake ..
$ make
$ make install  # (optionally)
```

- If your source code is in a different place, say my_src then you would call catkin_make like this:

```
# In a catkin workspace
$ catkin_make --source my_src
$ catkin_make install --source my_src  # (optionally)
```

- Once all of this is done, we can build the package using the command:
  catkin_make.

5) Understanding ROS nodes:

- This tutorial introduces ROS graph concepts and discusses the use of roscore, rosnode, and rosrun commandline tools.
- Overview of Graph Concepts:

  - Nodes: A node is an executable that uses ROS to communicate with other nodes.
  - Messages: ROS data type used when subscribing or publishing to a topic.
  - Topics: Nodes can publish messages to a topic as well as subscribe to a topic to receive messages.
  - Master: Name service for ROS (i.e. helps nodes find each other)
  - rosout: ROS equivalent of stdout/stderr
  - roscore: Master + rosout + parameter server.

- Nodes:
  A node really isn't much more than an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service.

- Client Libraries

  ROS client libraries allow nodes written in different programming languages to communicate:

  - rospy = python client library
  - roscpp = c++ client library

- roscore is the first thing we should run using ROS.
- Using rosnode:
  Open up a **new terminal**, and let's use **rosnode** to see what running roscore did... Bear in mind to keep the previous terminal open either by opening a new tab or simply minimizing it.

- rosrun:
  rosrun allows you to use the package name to directly run a node within a package (without having to know the package path).

6) Understanding ROS Topics

- This tutorial introduces ROS topics as well as using the rostopic and rqt_plot_commandline tools.
- rostopic contains the rostopic command-line tool for displaying debug information about ROS Topics, including publishers, subscribers, publishing rate, and ROS Messages. It also contains an experimental Python library for getting information about and interacting with topics dynamically.
- ROS messages:
  Communication on topics happens by sending ROS **messages** between nodes. For the publisher (turtle_teleop_key) and subscriber (turtlesim_node) to communicate, the publisher and subscriber must send and receive the same **type** of message. This means that a topic **type** is defined by the message **type** published on it. The **type** of the message sent on a topic can be determined using rostopic type.

7) Understanding ROS services and Parameters:

- This tutorial introduces ROS services, and parameters as well as using the rosservice and rosparam commandline tools.
- rosservice can easily attach to ROS's client/service framework with services. rosservice has many commands that can be used on services, as shown below:

```
rosservice list         print information about active services
rosservice call         call the service with the provided args
rosservice type         print service type
rosservice find         find services by service type
rosservice uri          print service ROSRPC uri
```

- Using rosparam:

  rosparam allows you to store and manipulate data on the ROS Parameter Server. The Parameter Server can store integers, floats, boolean, dictionaries, and lists. rosparam uses the YAML markup language for syntax. In simple cases, YAML looks very natural: 1 is an integer, 1.0 is a float, one is a string, true is a boolean, [1, 2, 3] is a list of integers, and {a: b, c: d} is a dictionary. rosparam has many commands that can be used on parameters, as shown below:

- Usage

```
rosparam set             set parameter
rosparam get             get parameter
rosparam load            load parameters from file
rosparam dump            dump parameters to file
rosparam delete          delete parameter
rosparam list            list parameter names
```

- When we use the command rosparam list we get following results:

```
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_57aea0986fef__34309
/rosversion
/run_id
```

ROS Dynamixel Tutorials:

1. Beginner Level:

1) Connecting to Dynamixel Bus

- This tutorial describes how to connect and examine raw feedback from Robotis Dynamixel servos
- The first step involves creating a package.
  We can do this with the help of command:

```
sudo apt-get install ros-<distro>-dynamixel-controllers
```

  In this our distro is Kinetic.
- Second step is creating a launch file for the manager node.
- We assume that the Dynamixel servos are connected to /dev/ttyUSB0 serial port.
- First we need to start up the controller manager that will connect to the motors and publish raw feedback data.
- Next step is to copy paste following code into a controller_manager.launch file

Toggle line numbers

```
<!-- -*- mode: XML -*- -->

<launch>
    <node name="dynamixel_manager" pkg="dynamixel_controllers" type="controller_manager.p
y" required="true" output="screen">
        <rosparam>
            namespace: dxl_manager
            serial_ports:
                pan_tilt_port:
                    port_name: "/dev/ttyUSB0"
                    baud_rate: 1000000
                    min_motor_id: 1
                    max_motor_id: 25
                    update_rate: 20
        </rosparam>
    </node>
</launch>
```

- In order to examine the motor feedback, we have to type the following commands:
  - rostopic list
  - rostopic echo /motor_states/pan_tilt_port
- As we were having four AX -12+ motors connected, the screenshot of the expected results is shown below.

motor_states:


```
motor_states:
 -
   timestamp: 1351555461.28
   id: 4
   goal: 517
   position: 527
   error: 10
   speed: 0
   load: 0.3125
   voltage: 12.4
   temperature: 39
   moving: False
 -
   timestamp: 1351555461.28
   id: 5
   goal: 512
   position: 483
   error: -29
   speed: 0
   load: 0.0
   voltage: 12.6
   temperature: 40
   moving: False
 -
   timestamp: 1351555461.28
   id: 6
   goal: 511
   position: 516
   error: 5
   speed: 0
   load: 0.0
   voltage: 12.3
```

```
  temperature: 41
  moving: False
-
  timestamp: 1351555461.28
  id: 13
  goal: 256
  position: 256
  error: 0
  speed: 0
  load: 0.0
  voltage: 12.3
  temperature: 38
  moving: False
---
```

2) Creating a meta controller:

- Meta controller is an action server that allows you to group up any number of motors and control it by an action client.
- First, we need to create a configuration file that will contain all parameters that are necessary for our controller.
- Then, We need to create a configuration file that group up all controllers and made it an action server. Paste the text below following into joints_trajectory_controller.yaml file

```
f_arm_controller:
  controller:
    package: dynamixel_controllers
    module: joint_trajectory_action_controller
    type: JointTrajectoryActionController
  joint_trajectory_action_node:
    min_velocity: 0.1
    constraints:
      goal_time: 0.25
```

- Its important that the motor id matches the id assigned to your dynamixel actuator. The goal_time parameter in the .yaml file changes only the time period between each action and does not interfere in the motor's velocity.

- Next we need to create a launch file that will load controller parameters to the parameter server and start up the controller. Paste the following text into start_meta_controller.launch file.

```
<launch>
<!-- Start tilt joint controller -->
   <rosparam file="$(find my_dynamixel_tutorial)/tilt.yaml" command="load"/>
   <node name="controller_spawner" pkg="dynamixel_controllers" type="controller_spawner.py"
         args="--manager=dxl_manager
               --port dxl_USB0
               joint3_controller
               joint4_controller
               joint5_controller
               "
         output="screen"/>

  <!-- Start joints trajectory controller controller -->
   <rosparam file="$(find my_dynamixel_tutorial)/joints_trajectory_controller.yaml" command="load"/>
   <node name="controller_spawner_meta" pkg="dynamixel_controllers" type="controller_spawner.py"
         args="--manager=dxl_manager
               --type=meta
               f_arm_controller
               joint3_controller
               joint4_controller
               joint5_controller
              "
         output="screen"/>
</launch>
```

- After the controller manager is up and running we finally load our controller, with the help of following command:

```
roslaunch my_dynamixel_tutorial start_meta_controller.launch
```

- After that, using “rostopic list”command, we can list the topics and services that are provided by our controller.
- The relevant topics which we get are:

```
/f_arm_controller/command
/f_arm_controller/follow_joint_trajectory/cancel
/f_arm_controller/follow_joint_trajectory/feedback
/f_arm_controller/follow_joint_trajectory/goal
/f_arm_controller/follow_joint_trajectory/result
/f_arm_controller/follow_joint_trajectory/status
/f_arm_controller/state
/joint3_controller/command
/joint3_controller/state
/joint4_controller/command
/joint4_controller/state
/joint5_controller/command
/joint5_controller/state
```

3) Setting up Dynamixel:

- This tutorial should help us to get used to Dynamixel and how to set it's parameters in order to control it.
- First of all, we must understand the basic concepts of Dynamixel. It is a servomotor in which the user can easily use it by giving certain paramaters through ROS, which is going to be covered up later. Also, there are three ways we can control our Dynamixel:
  - Joint Mode
  - Wheel Mode
  - Multi-turn Mode
- It's very important to say that every time we want to configure any mode of our Dynamixel we always must run our controller manager.
- Pinging the motor:
  - This can be done with the help of a Python script.
- Joint Mode:
  - To control the arm of a robot, we should rotate a certain angle and then the arm should freeze at the given angle, that's what a joint does. Configurating the Dynamixel to joint mode is the simplest configuration that we can do.
- Wheel Mode:
  - To use Wheel Mode, we must look up for the following package in our terminal

```
$ rosrun dynamixel_drivers set_servo_config.py
```

- Realize that inside the package "dynamixel_drivers" there is a python script called "set_servo_config.py", which is the script to set parameters to Dynamixel. When giving the previous command we should see something like this in the terminal:

```
Options:
  -h, --help            show this help message and exit
  -p PORT, --port=PORT  motors of specified controllers are connected to PORT
                        [default: /dev/ttyUSB0]
  -b BAUD, --baud=BAUD  connection to serial port will be established at BAUD
                        bps [default: 1000000]
  -r RATE, --baud-rate=RATE
                        set servo motor communication speed
  -d DELAY, --return-delay=DELAY
                        set servo motor return packet delay time
  --cw-angle-limit=CW_ANGLE
                        set servo motor CW angle limit
  --ccw-angle-limit=CCW_ANGLE
                        set servo motor CCW angle limit
  --min-voltage-limit=MIN_VOLTAGE
                        set servo motor minimum voltage limit
  --max-voltage-limit=MAX_VOLTAGE
                        set servo motor maximum voltage limit
```

- There are plenty of parameters, in this case we just want to put the Dynamixel in wheel mode. Now we give the following command

```
$ rosrun dynamixel_drivers set_servo_config.py 5 --baud=57600 --cw-angle-limit=0
--ccw-angle-limit=0
```

- The first parameter, which is "5" is related to our Dynamixel's ID, in this case you should see your Dynamixel's ID before.
- The second parameter is the baud rate, there are plenty of baud rates available, you should also check it before.
- The two new parameters here are "--cw-angle-limit" and "--ccw-angle-limit".
- CW means clockwise and CCW means counterclockwise, so it makes sense to say that if we want our motor to fully rotate, then both CW and CCW limits should be 0.

- Multi-turn mode:
    - This mode allows the dynamixel to receive a goal bigger than 4095 (4095 beign a full rotation). It's important to say that the Multi-turn mode is only available in some servos from Dynamixel, you should check the documentation before trying anything.

2. Intermediate Level

1) Creating a Joint Controller

- We learnt how to create a joint controller with one or more Robotis Dynamixel motors.

2) Creating a Joint-Torque Controller

- We learnt how to create a joint torque controller with the help of a Robotis Dynamixel motor.

3) Creating a Dynamixel Action Client Controller

- We learnt how to create a joint controller with one or more Robotis Dynamixel motors.

Setting up Multiple Dynamixle Servos together:

U2D2 Adapter:

Figure U2D2

- U2D2 is a small size USB communication converter that enables to control and to operate the DYNAMIXEL with the PC.
- It is smaller than the previous model and has mount holes to make it easier to install on robots.
- It uses the USB cable to connect to the PC and prevents damage of the USB terminals. It has both 3Pin connectors for TTL communication and 4Pin connectors for RS-485 communication embedded for easier control and access for the Dynamixel X series.
- Supports UART as well.
- Requires the convertible cable for Molex connector using Dynamixels.
- **U2D2** does not supply power to the DYNAMIXEL.
- Therefore, an external power supply should be used to provide power to the DYNAMIXEL servo.

Power Hub:

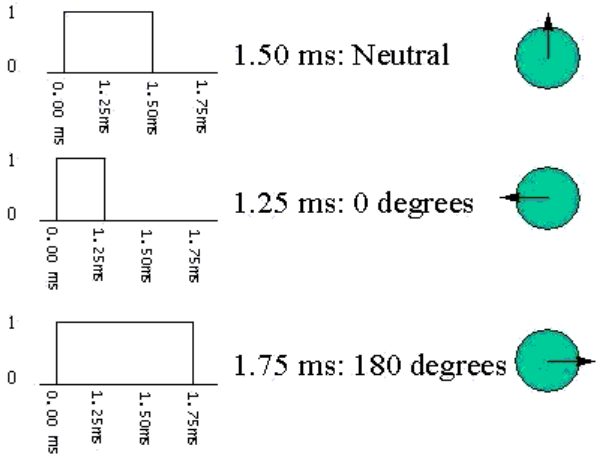
Dynamixel Servos:

- Dynamixel actuators have been used by every major university, research lab, military & government research lab, and robotic competition worldwide.

- The casing of each servo is built specifically with robotics in mind, providing easy to use mounting rails and a comprehensive bracket system available for building robotic limbs.
- TTL and RS-485 serial communication allows for daisy-chainable bus connections at up to 1-3mbps.

How Regular Servos Work

http://www.pyroelectro.com/tutorials/servo_motor/servomotor.html

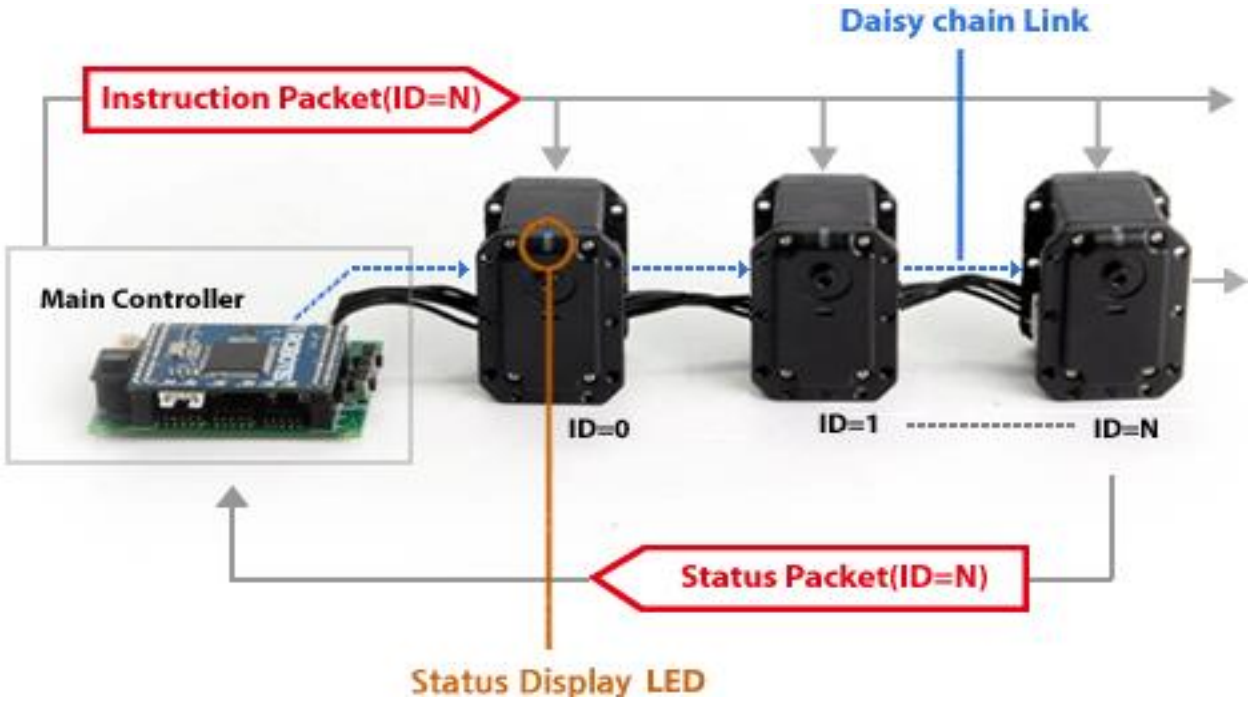


http://www.pitt.edu/~sorc/robotics/handbook/

- Conventional Servo motors are used for positioning
  - every 20ms you need to send a pulse to the servo motor,
  - the width of this pulse determines the position the servo.
- This has to be done for every servo motor in the system => large overhead on main mcu.

How Dynamixel Servos Work:

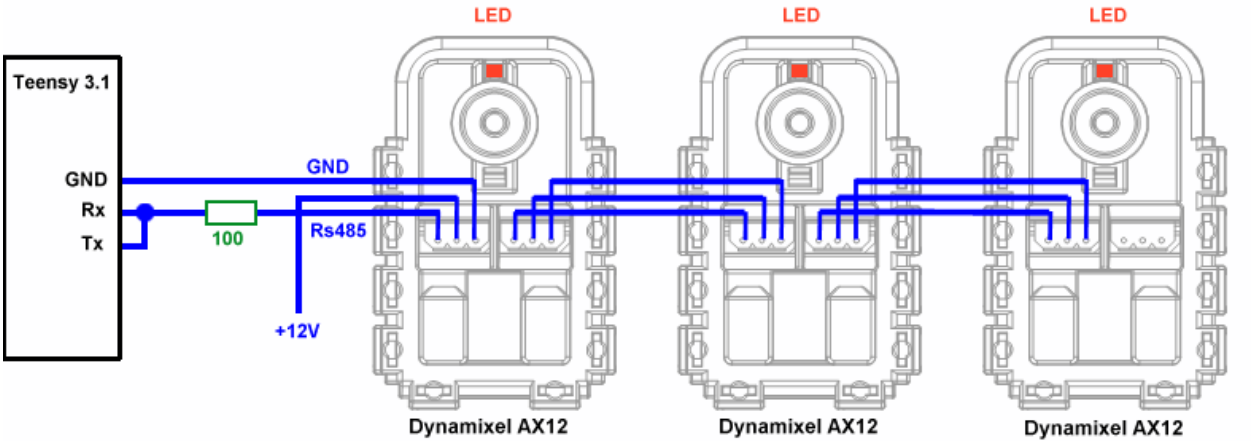
- Use serial communication:
  - TTL serial (3 wire) or RS485 (4 wire), and
  - 1Mbps to 3Mbps communication speeds.
- Dynamixels are driven by digital packet communication. The main communication protocol is half-duplex UART (8-bit, no parity, 1 stop).
- It supports TTL, RS485, or CAN networks. Dynamixel communicates with the main controller by sending and receiving packets



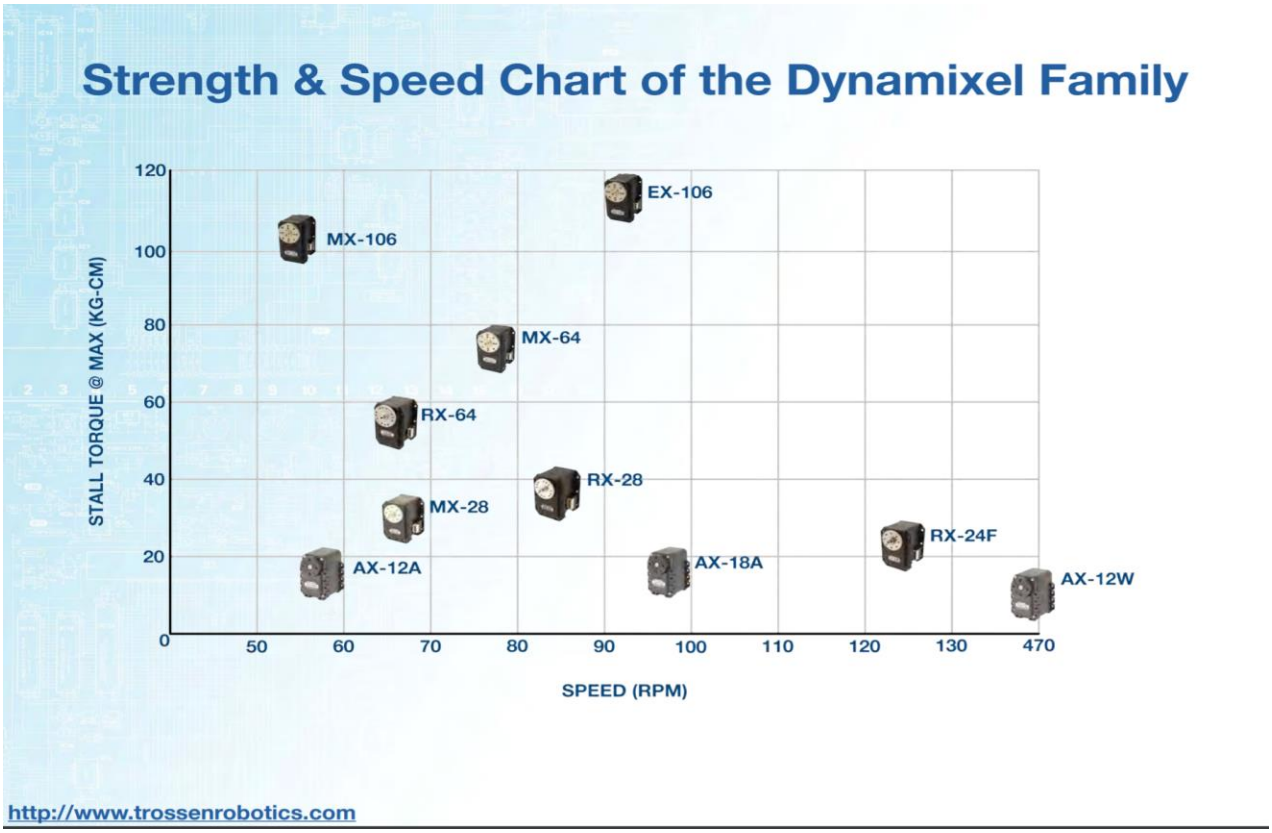
http://tribotix.com/Products/Robotis/Dynamixel/DynamixelIntro.htm

- Dynamixels are 'networked':
  - each Dynamixel in the chain must have a unique ID
  - they are connected via a Daisy Chain, and
  - 255 Dynamixels can be connected to each network.

Basic Setup for Controlling Dynamixel Servos:



Dynamixel Servo Family:



- Figure shows the Strength and Speed characteristic chart of the Dynamixel family.

- **Servo Test/Control Setup**

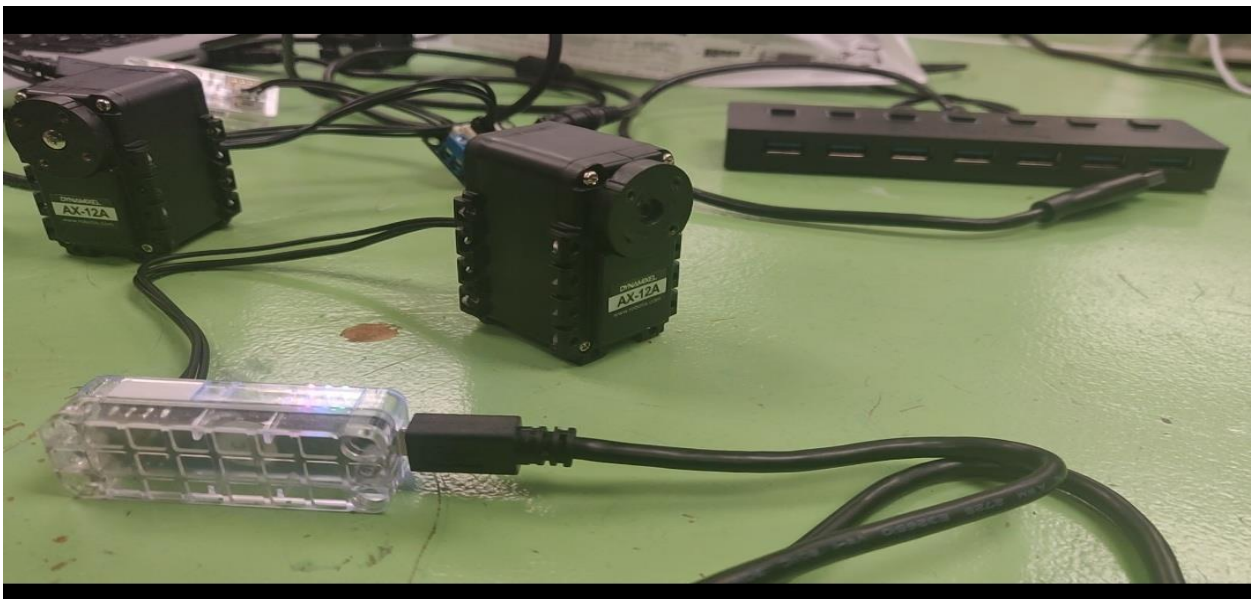
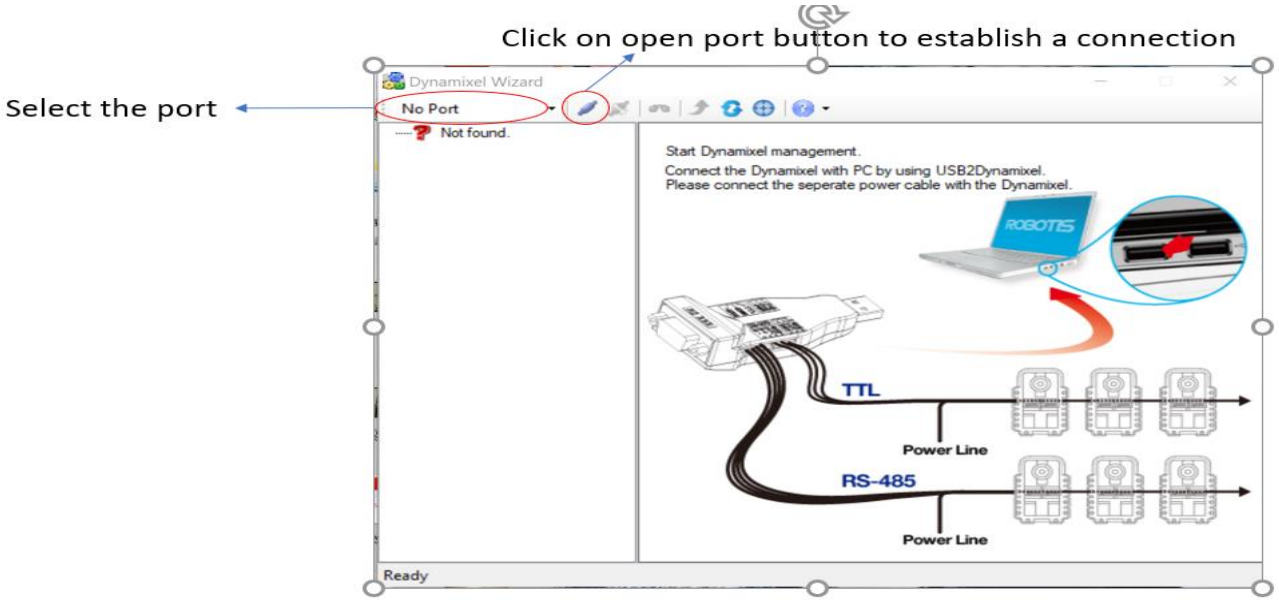


Figure. Dynamixel Servo Test Setup using U2D2

Using Dynamixel Wizard for Setting up Servos:

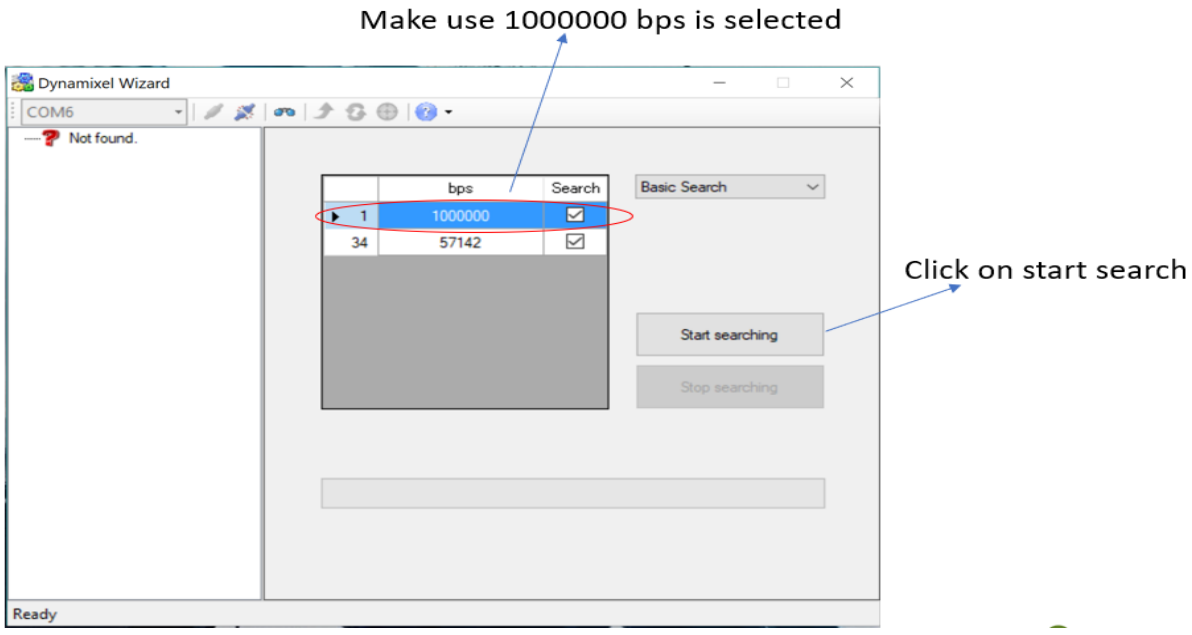
For this, we need to install Dynamixel Wizard software from Robotis and follow these steps:

Step 1:



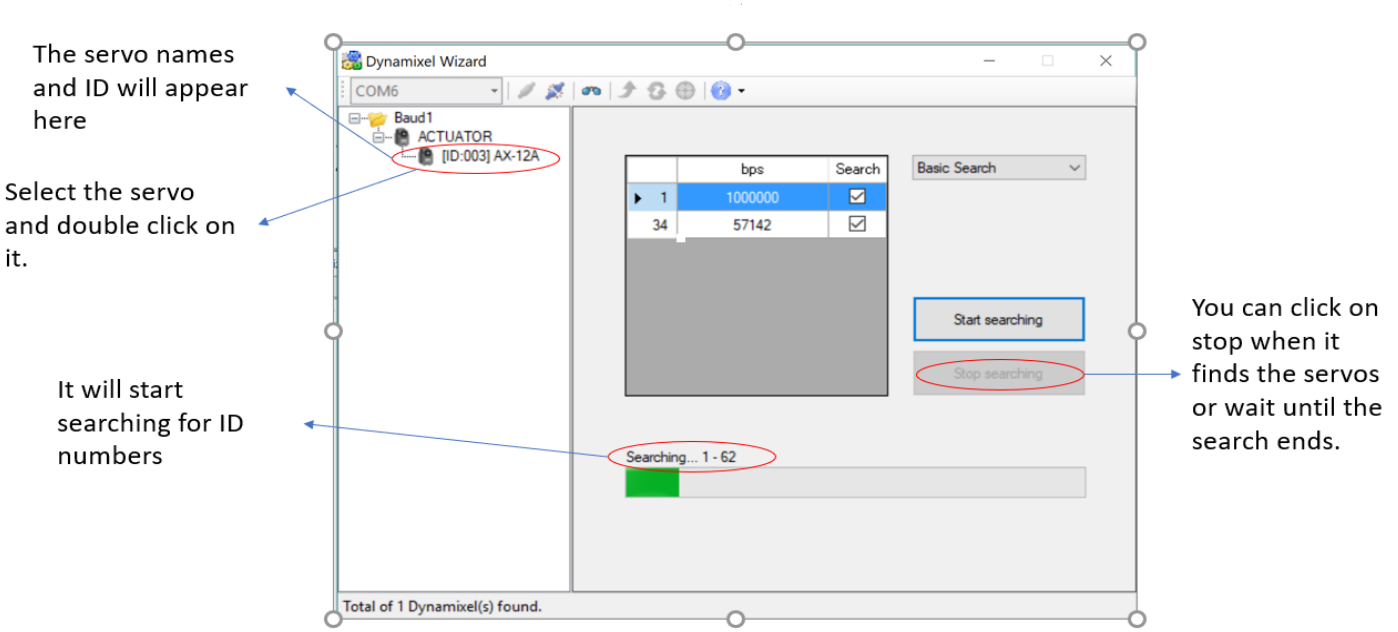
- As shown in above figure, first we need to Open a port.

Step 2:



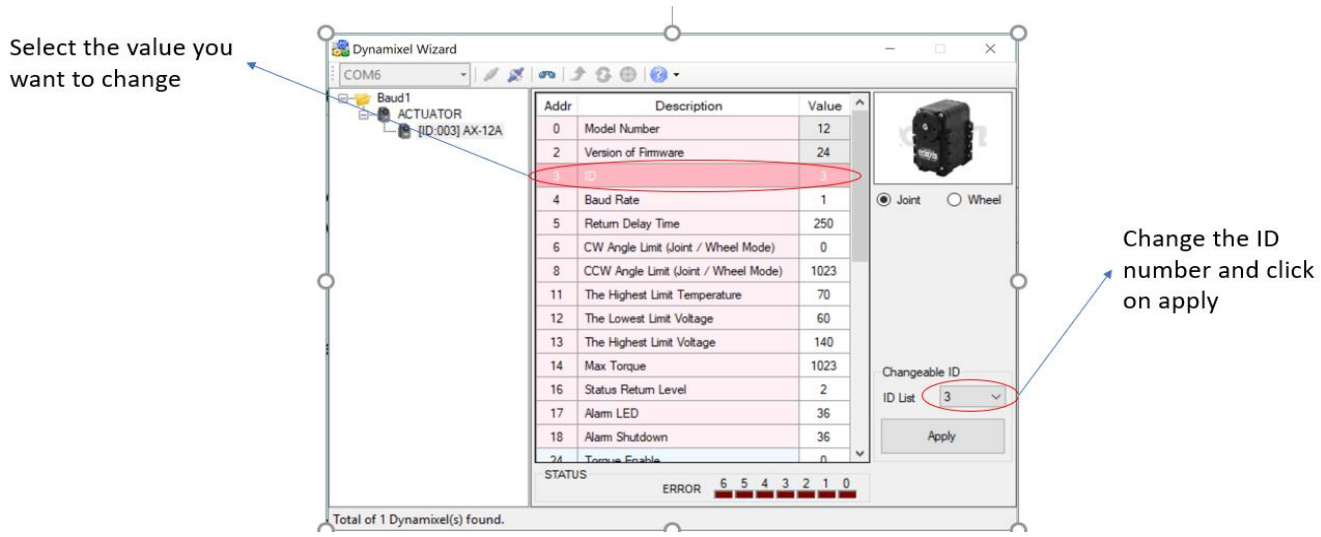
- Then we need to search for the servos
- Important thing is the baud rate should be 1000000 bps.

Step 3:



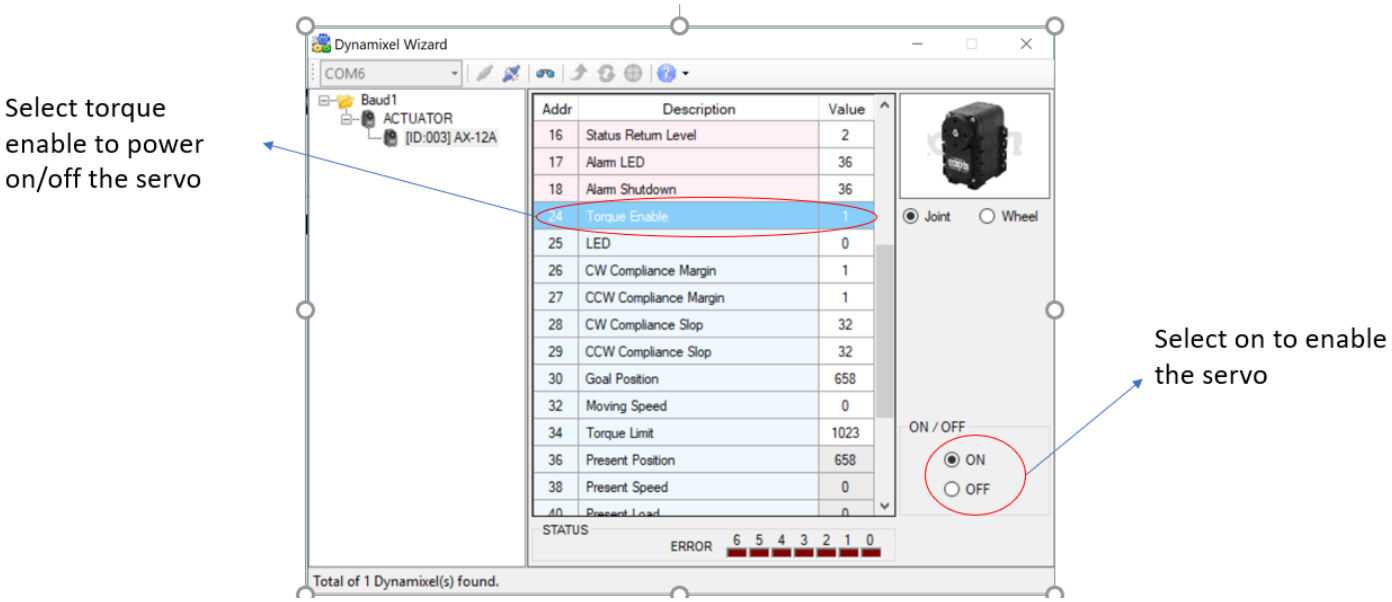
- Once we will start searching the servos, there name and IDs will start appearing which means we connected everything properly.
- Once we search the servos we want, we can select ‘Stop Searching’ option or also we can wait till the search ends.

Step 4:



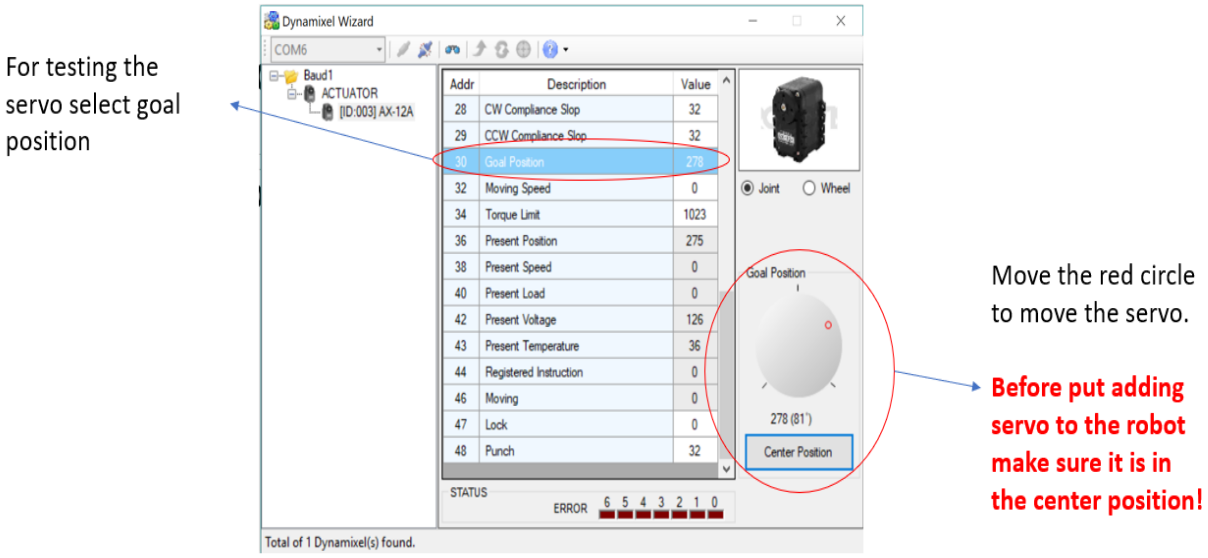
- As shown in figure, we can change the ID number to the value we want.

Step 5:



- We can select the option Torque enable to Power ON/OFF the servo.

Step 6:



- We can move the servo by controlling the position of the red circle as shown in the above figure.
- Before controlling the servo, we need to make sure that it is in the center position.

Framework

- HRO-S5 Framework, based on Darwin-OP framework, is a collaborative software project based on the open source Darwin-OP Framework, with the intent to continue development, implement an API to expose higher level functions of the framework, and develop additional features.
- The Data Folder of the framework contains the motion binary files, mp3 files the configuration file where walk tuner settings are saved and loaded from, as well as vision settings.
- The Framework folder contains the core Darwin-OP/HR-OS5 Framework source code, written in C++ and OS independent.
- The Linux folder contains Darwin-OP/HR-OS5 Linux-specific source code, the darwin.a library object Build folder, and projects/demos.
- The dxl monitor program is a utility for managing your Dynamixel sensors and CM-730/Arbotix sub controller register values.
- This program can be used to assign network IDs, change limits, monitor heat and load, as well modify register table values on the sub controller.
- The robot motion editor(rme) is a customized version of the original Darwin-OP 'action_editor'.
- This node server file connects to the HR-OSX-Framework libraries via Linux/build/api_wrapper/api_wrapper.so The api_wrapper.so has basic high-level

functions of the robot exposed to the node server API, allowing users to send high-level commands.

- This Framework was based on the HR-OS1 framework; therefore it needed more adjustments and get it to work on HR-OS5 and implement it in ROS.
- According to the ROS tutorials, any movement commands should be hard coded into .yaml file and launch it through the .launch file which would send these commands to the dynamixel servos either for single servo motion or multiple servos motions.
- Therefore, implementing the HR-OS5 framework into ROS needed a python script to execute the launch files according the requested command or movement.
- The finding of HR-OS5 framework was team effort as it was not provided to us by the course teaching assistance, so getting it to be implemented into ROS was basically a testing method that resulted into unsuccessful attempts which needed more time to make any progress.

Challenges Faced and Limitations

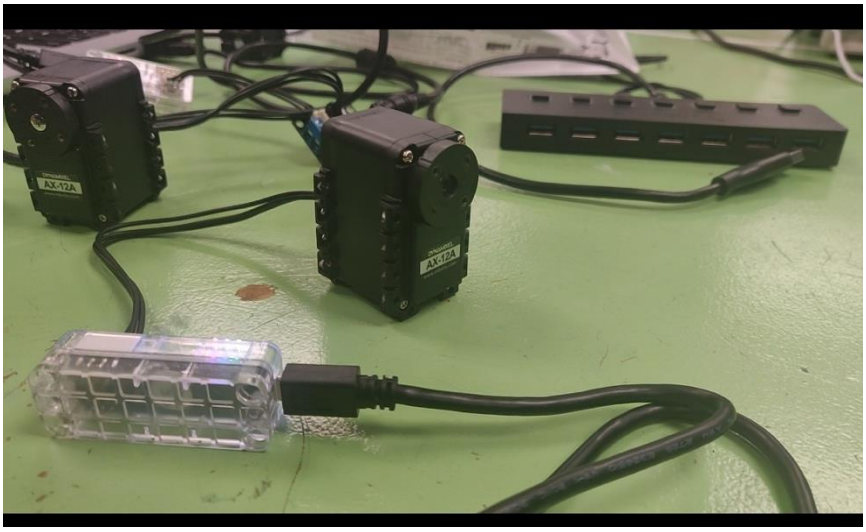
- At the beginning of this project we weren't able to have access to the HR-OS5 robot and we only had access to a single dynamixel servo and a robotis dynamixel adapter to connect the dynamixel servo to out computers for the tutorial and get familiar with ROS and dynamixel wizard for testing and identifying the dynamixel servo.
- However the first issue face us was that we weren't able to see the dynamixel servo on the dynamixel wizard software and the ttyUSB0 file was not showing on the dev/tty folder on the linux system, therefore after a week we got another dynamixel adapter from our teaching assistant and tried it but after several attempts, we found out that this type of adapter is outdated and we must use the U2D2 adapter by robotis which it is the new version of the needed adapter that is supported by the dynamixel wizard software.
- Therefore, we ordered two new adapters from an online supplier that took more than a week to get them delivered. By then we were able to test the single servo and learn how to do so and how to use the dynamixel wizard software.
- The dynamixel servo testing was done almost on week seven and about that time we had our first access to the HR-OS5 robot as we start to learn about its components and found out that the robot arms' servos were assembled in different order so we had to re-assemble one of the arms to match the other one which took few days as some screws and idle separators were not available and had to find some replacements.
- The other issue faced us was the lack of a working USB hub to use it to connect the Arbotix pro board, so we had to purchase a new one. Therefore, by almost week nine the

existing robot’s dynamixel servos were under testing on how to control them through ROS according to the tutorial which we finished by week six.

- So, with the time limitation we were able to test and control a single arm of the HR-OS5 robot and we are planning to continue to do the same for the rest part of the robot and use the HR-OS5 framework through ROS next term.

Implementation

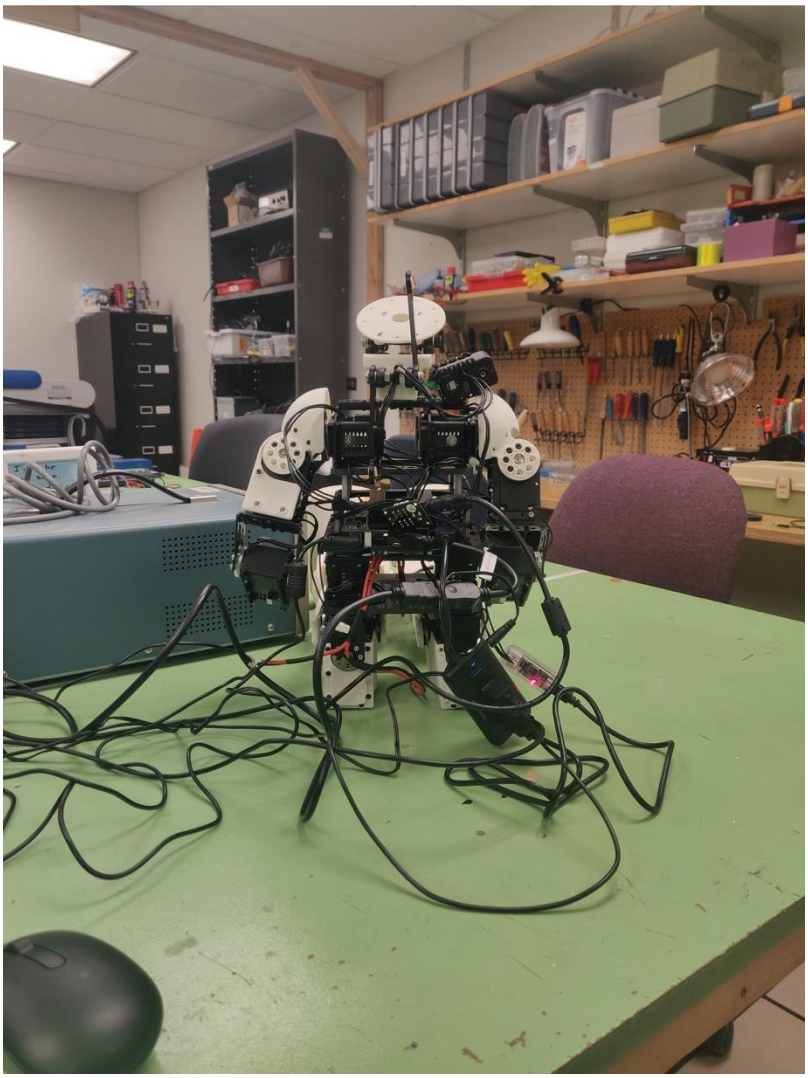
- Finish the ROS tutorial on how to setup the ROS OS on a linux system and install all the necessary dependencies.
- Finish the dynamixel tutorial on how to install the necessary dependencies and on how to control the dynamixel servos as a single servo or multiple servos as a joint.
- Adding root privilege to main user of the linux system to the dev/tty/ttyUSB0 file so it can be accessed by the necessary software.
- Install the dynamixel wizard software and use it to check the dynamixel servos IDs and configure each servo as a joint or a wheel.



- Reconfigure the robot's right arm dynamixel servos order to match the left arm.
- Check all the robot's dynamixel servos IDs and their operating conditions.

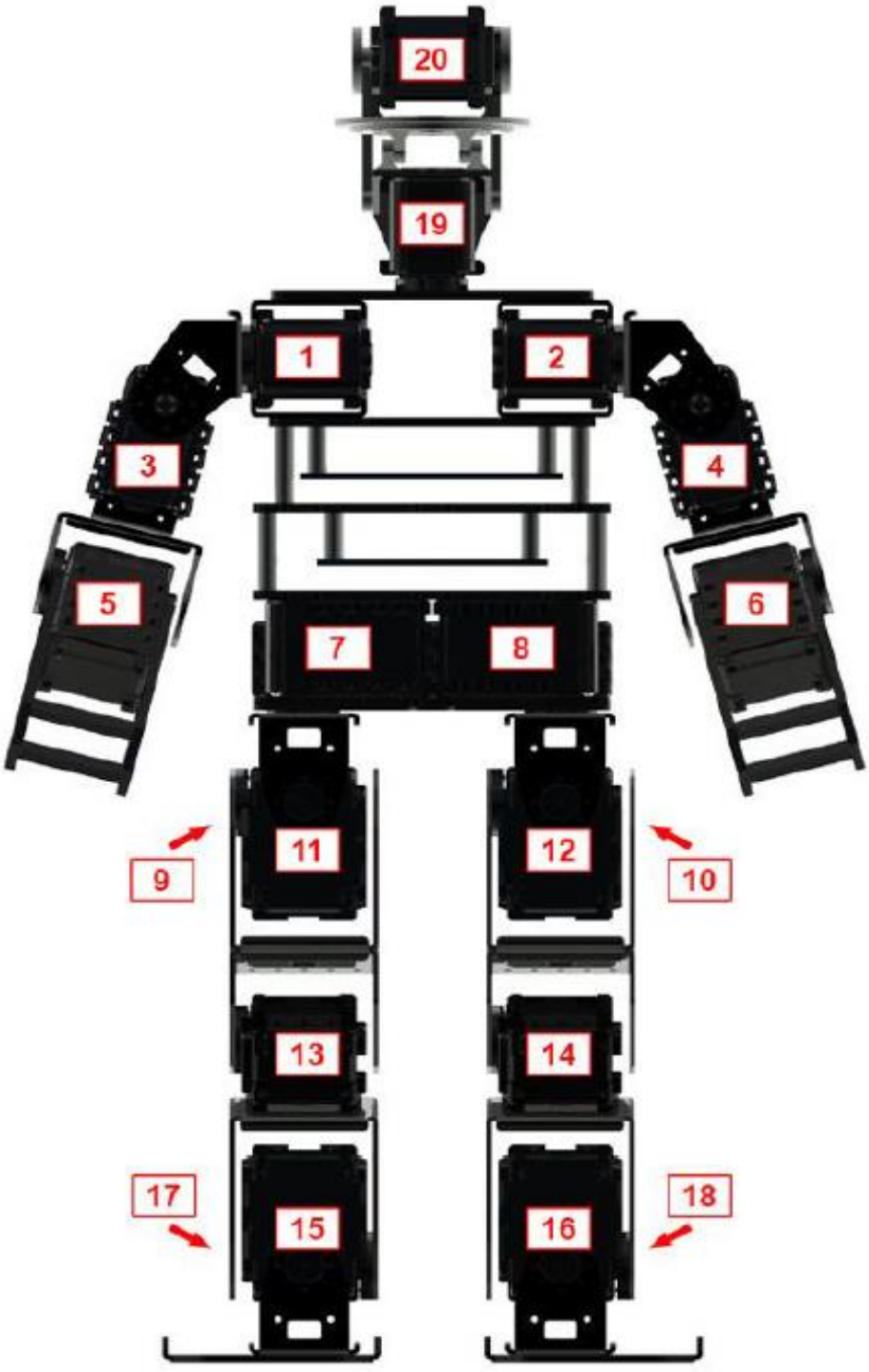
- Document the dynamixel servos' IDs according to the dynamixel servos' locations and movements.

- Implement different .yaml file settings and .launch file settings for testing the robot's dynamixel servos.
- Sourcing all the necessary setup files in command line to make sure all the settings are stored and implemented when ruuning the launch file.
- Testing the right arm movement by controlling a single dynamixel servo and multiple dynamixel servos as a joint.



- Trying to test the HR-OS5 framework by itself and trying to implement it into ROS.
- Installing all the necessary dependencies needed for the HR-OS5 framework.

Dynamixel Servos' IDs



Servo Positions:

No.	**Position**	**ID**	Type
1	Left Shoulder	2	MX-106
2	Left Bicep	4	MX-64
3	Left Elbow	6	MX-64
4	Right Shoulder	1	MX-106
5	Right Bicep	3	MX-64
6	Right Elbow	5	MX-64
7	Right Waist	7	MX-64
8	Right Ankle	17	MX-106
9	Right Foot	15	MX-106
10	Right Knee	13	MX-106
11	Right Fumor	11	MX-106
12	Right Hip	9	MX-106
13	Left Waist	8	MX-64
14	Left Ankle	18	MX-106
15	Left Foot	16	MX-106
16	Left Knee	14	MX-106
17	Left Fumor	12	MX-106
18	Left Hip	10	MX-106
19	Neck	19	MX-28
20	Mouth		

Learnings

- Got familiar with ROS environment.
- Learnt how to control Dynamixel servos using the software Dynamixel Wizard.
- Also learnt controlling the Dynamixel servos with help of scripting.

YouTube Video Links

1) Dynamixel Servo Control: https://www.youtube.com/watch?v=NHNJTO2-MgY
2) HR-OS5 Arm motion Control: https://www.youtube.com/watch?v=WE-tsQba5b0

References

1. HR-OS5: https://www.trossenrobotics.com/HR-OS5

2. ROS WIKI: http://wiki.ros.org/ROS/Tutorials

3. ROS Dynamixel Servos: http://wiki.ros.org/dynamixel

4. U2D2 Adapter: http://www.robotis.us/u2d2/

5. HR-OS5 Framework: https://github.com/Interbotix/HROS5-Framework