ROBOTIS Manipulator SDK

User Manual

ROBOTIS

Contents

1. Introduction

1.1 About this document

ⅰ) This manual applies to the Dynamixel PRO-based Robotis Manipulator.

ⅱ) All parameters in this manual are based on default values.

ⅲ) The manipulator's configuration is provided. The ArmSDK is based on Windows7 and Visual Studio 2010.

ⅳ) It is strongly recommended with proficiency with Dynamixel PRO and C++.

ⅴ) The units utilized in the ArmSDK are in radians (rad) and millimeters (mm).

ⅵ) Modifying the wiring and components or performances not stated on this guide may result on adverse operations.

This manual utilizes the term "arm," "manipulator," and "robot" interchangeably to describe product. The guide also may refer to Dynamixel Pro actuators to simply "Dynamixel," "servo," "motor," or "actuator."

(last updated 5 November, 2014)

1.2 Safety

DANGER!

Information appearing in a DANGER concerns the protection of personnel from the immediate and imminent hazards that, if not avoided, will result in immediate, serious personal injury or loss of life in addition to equipment damage.

- Keep away from the robot while its moving..
- Do not touch with the robot with wet hands.
- Turn off power of the robot whenever robot is problematic.

WARNING!

Information appearing in a WARNING concerns the protection of personnel and equipment from potential hazards that can result in personal injury or loss of life in addition to equipment damage.

- Setup robot in an environment low on dust and humidity.
- The robot must always be attached to the based when powered on.
- The robot wiring must be checked prior to powering on.
- The robot connection to power supplly must be check prior to powering on.
- Do not change wiring on Robotis Manipulator while powered on.

CAUTION!

Information appearing in a CAUTION concerns the protection of personnel and equipment, software, and data from hazards that can result in minor personal injury or equipment damage.

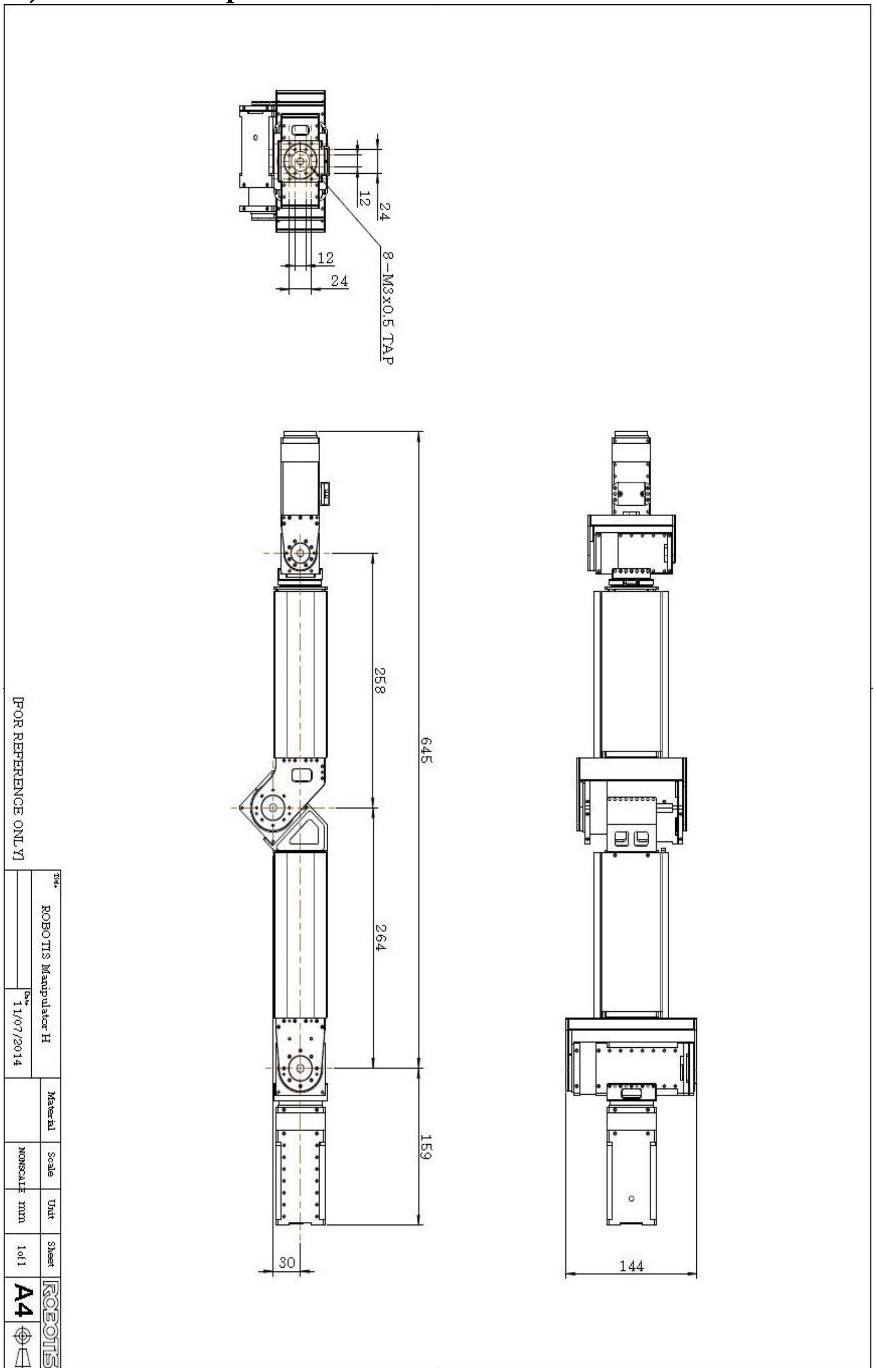
- Keep robot’s workspace clear of object.
- Ensure wiring is not tangled up on every joint.
- Make sure USB2Dynamixel and PC does not interfere with the robot’s moving

1.3 Package Contents

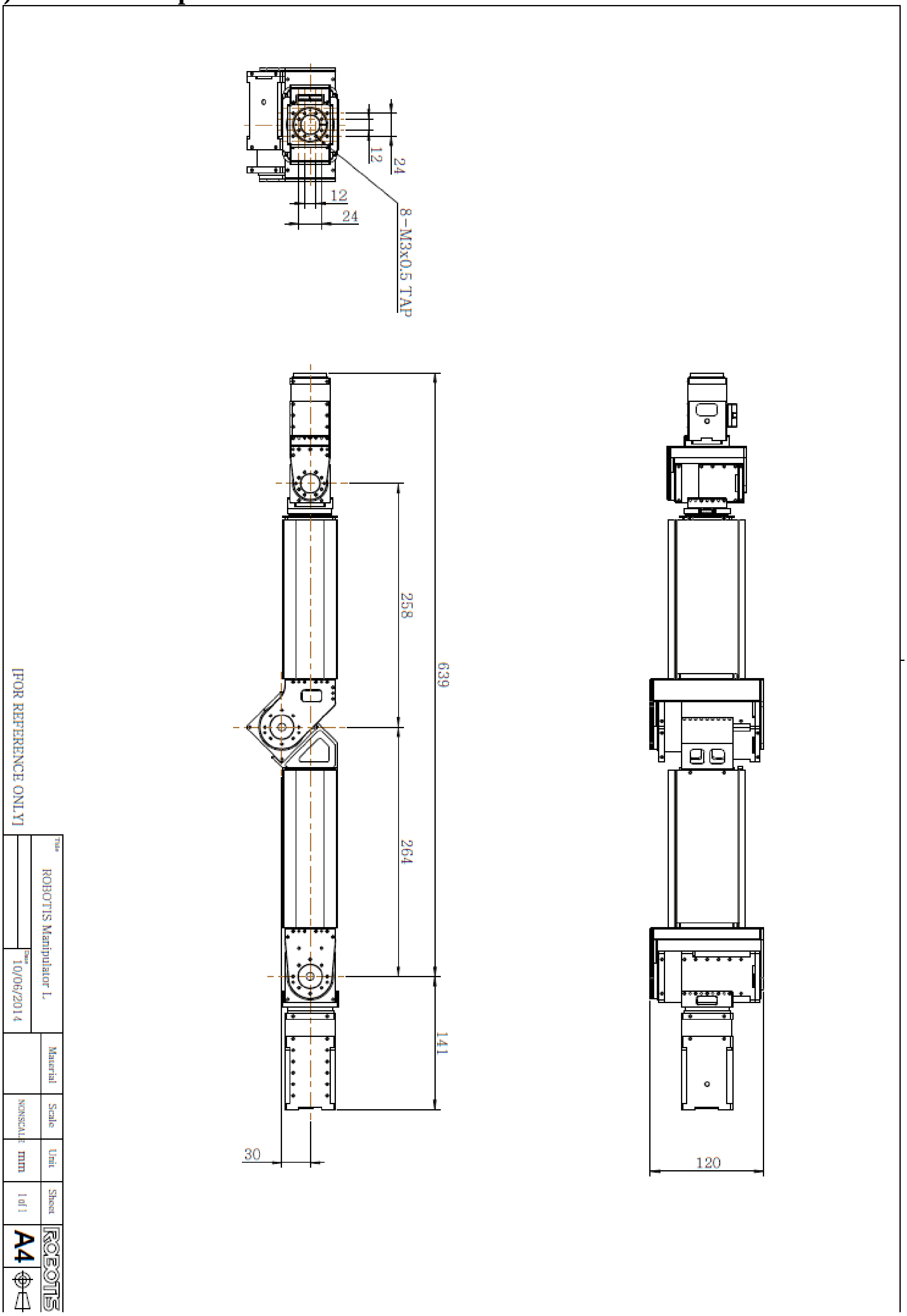
Name	Quantity
Manipulator	1
USB2Dyanmixel	1
4P Cable(500mm)	2
Power Cable(1,200mm)	2
4P expansion hub	1
Power expansion hub	1
Gripper(optional)	1
Support(optional)	2
Base Plate(optional)	1
3x8 wrench bolt	20
3x12 wrench bolt	20

1.4 Layout

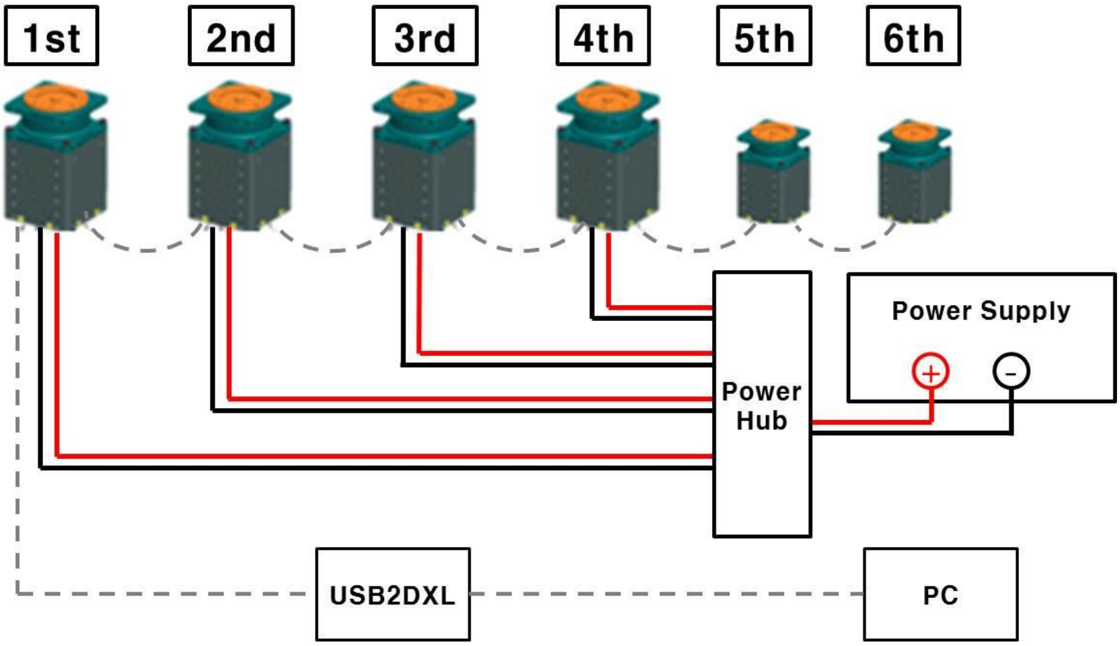
i) H Series Manipulator Dimension



ii) L Series Manipulator Dimension



ii) Wiring



- The diagram above illustrates joints 1~6 connected in daisy-chain (serial) configuration with 4P Cable.

- Joint 1 (labeled as "1st") connects to USB2Dynamixel via 4P Cable.

- USB2Dynamixel connects to PC via USB hub.

- Dynamixel Pro is powered from a a power supply via power expansion hub.

- Joints 5 and 6 (model: L42 - 10 - S300 – R) are not separately powered; instead power comes from the same 4P Cable.

- You may obtain more 4P or Power Cables via ROBOTIS or see section 2.2 Preparation - ii) Cable.

1.5 Specification

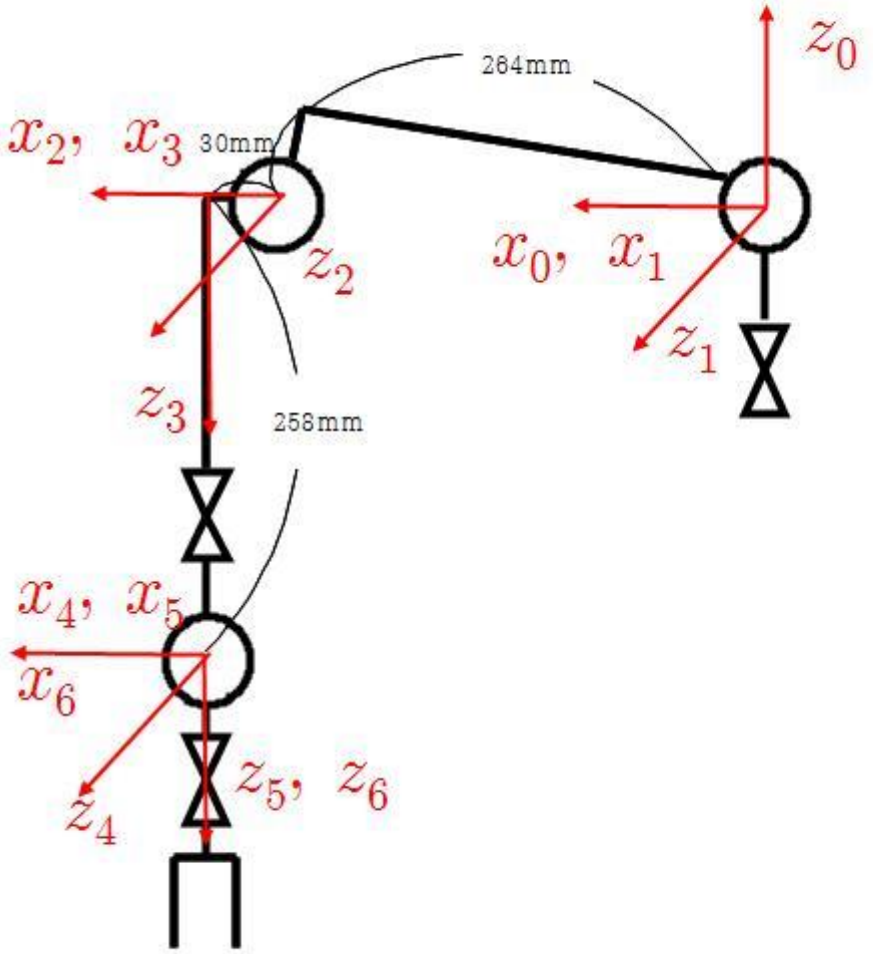
i) H Series Robotis Manipulator Specification

Item		description
DOF		6 DOF
Payload		3kg
Operating voltage		24V
Resolution	Joint 1	$-\pi$(rad) ~ π(rad) → -251000 ~ 251000 (pulse)
	Joint 2	$-\pi$(rad) ~ π(rad) → -251000 ~ 251000 (pulse)
	Joint 3	$-\pi$(rad) ~ π(rad) → -251000 ~ 251000 (pulse)
	Joint 4	$-\pi$(rad) ~ π(rad) → -251000 ~ 251000 (pulse)
	Joint 5	$-\pi$(rad) ~ π(rad) → -151875 ~ 151875 (pulse)
	Joint 6	$-\pi$(rad) ~ π(rad) → -151875 ~ 151875 (pulse)
Dynamixel Pro Model Name	Joint 1	H54 - 200 - S500 - R
	Joint 2	
	Joint 3	H54 - 100 - S500 - R
	Joint 4	
	Joint 5	H42 - 20 - S300 - R
	Joint 6	
Operating Range	Joint 1	$-\pi$(rad) ~ π(rad)
	Joint 2	$-\pi/2$(rad) ~ $\pi/2$(rad)
	Joint 3	$-\pi/2$(rad) ~ $3\pi/4$(rad)
	Joint 4	$-\pi$(rad) ~ π(rad)
	Joint 5	$-\pi/2$(rad) ~ $\pi/2$(rad)
	Joint 6	$-\pi$(rad) ~ π(rad)
Default ID		Joint 1 (ID:1), Joint 2 (ID:2), Joint 3 (ID:3) Joint 4 (ID:4), Joint 5 (ID:5), Joint 6 (ID:6)
Motor type		Brushless DC Servo(H54 Series) / Coreless DC Motor(H42 Series)
Position sensor type		Absolute Encoder(for Homing) & Incremental Encoder(for Control)
communications		RS485

ii) L Series Robotis Manipulator Specification

Item		Description
DOF		6 DOF
Payload		1kg
Operating voltage		24V
Resolution	Joint 1	-π(rad) ~ π(rad) → -125700 ~ 125700 (pulse)
	Joint 2	-π(rad) ~ π(rad) → -125700 ~ 125700 (pulse)
	Joint 3	-π(rad) ~ π(rad) → -125700 ~ 125700 (pulse)
	Joint 4	-π(rad) ~ π(rad) → -144180 ~ 144180 (pulse)
	Joint 5	-π(rad) ~ π(rad) → -2048 ~ 2048 (pulse)
	Joint 6	-π(rad) ~ π(rad) → -2048 ~ 2048 (pulse)
Dynamixel Pro Model Name	Joint 1	L54 – 50 – S500 - R
	Joint 2	
	Joint 3	
	Joint 4	L54 – 30 – S500 - R
	Joint 5	L42 - 10 - S300 - R
	Joint 6	
Operating Range	Joint 1	-π(rad) ~ π(rad)
	Joint 2	-π/2(rad) ~ π/2(rad)
	Joint 3	-π/2(rad) ~ 3π/4(rad)
	Joint 4	-π(rad) ~ π(rad)
	Joint 5	-π/2(rad) ~ π/2(rad)
	Joint 6	-π(rad) ~ π(rad)
Default ID		Joint 1 (ID:1), Joint 2 (ID:2), Joint 3 (ID:3) Joint 4 (ID:4), Joint 5 (ID:5), Joint 6 (ID:6)
Motor type		Brushless DC Servo(L54 Series) / Coreless DC Motor(L42 Series)
Position sensor type		Absolute Encoder(for Homing) & Incremental Encoder(for Control) ※ L42 models only contain absolute encoders
communications		RS485

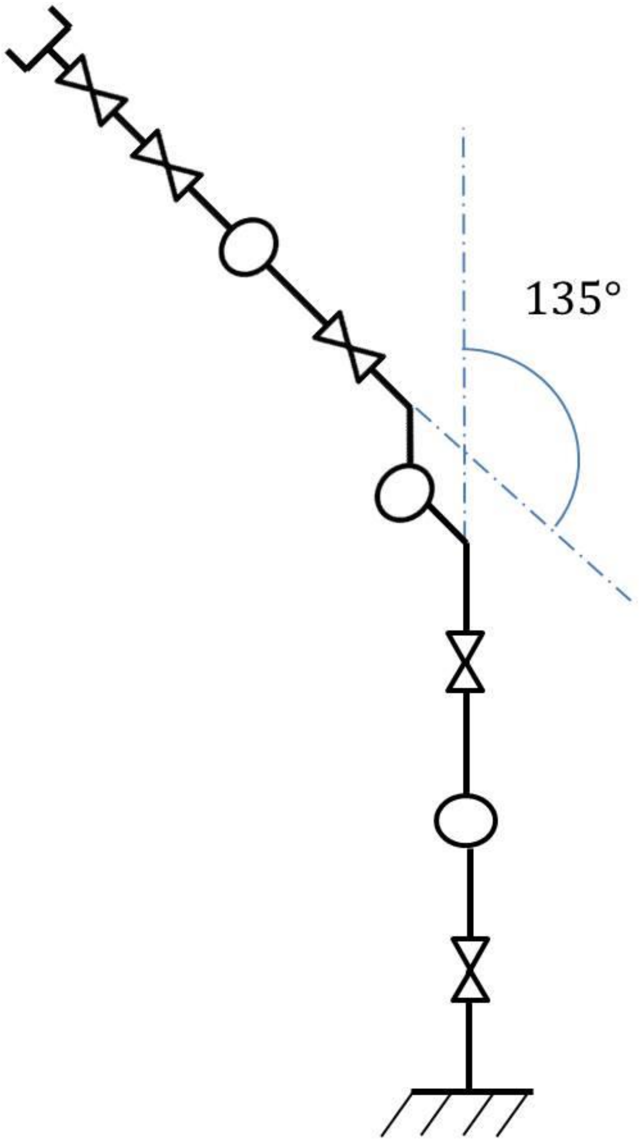
1.6 D-H Configuration



DH Parameter / Link No.	Link Length (mm)	Link Twist (rad)	Joint Offset (mm)	Joint Angle (rad)	DXL Angle (rad)
1	0	$-\frac{\pi}{2}$	0	0	0
2	265.69 ($\sqrt{264^2+30^2}$)	0	0	0	$\frac{\pi}{2}-\tan^{-1}(\frac{30}{264})$
3	30	$-\frac{\pi}{2}$	0	0	$\frac{\pi}{4}+\tan^{-1}(\frac{30}{264})$
4	0	$\frac{\pi}{2}$	258	0	0
5	0	$-\frac{\pi}{2}$	0	0	0
6	0	0	0	0	0

1.7 Home Position of Robotis Manipulator

- The diagram below shows the "home position" of the Dynamixel PROs from Robotis Manipulator.



2. Getting Started

2.1 Prerequisites

ⅰ) The ArmSDK is based on Window 7 OS and Visual Studio 2010.

ⅱ) ⚠ The ArmSDK trajectory is generated from the MotionPlay class' instance and utilizes QueryPerformanceCounter. This requires the use of a thread, in which sharing said thread may reach to 100%. It is highly recommended your PC is at least dual-core-based.

ⅲ) ⚠ The Numerical IK implements Damped Least Square Method to reach target by acquiring each joint's angle. This allows joints to go from initial position to a point and then return to its initial pose. This will allow you to perform tests to the manipulator.

ⅳ) ⚠ Allow sufficient workspace prior to setup by clearing objects in the arm's vicinity.

ⅴ) ⚠ Always ensure the manipulator is properly fixed to the base plate prior to operations; otherwise arm movements can cause damage and physical injury.

ⅵ) ⚠ supply power to the manipulator after making sure all cables are properly connected. While powered on do not touch the cables as it may cause erroneous operations or/and damage.

ⅶ) ⚠ When handling the manipulator do so carefully as not to have your fingers stuck in the frames.

ⅷ) ⚠ If the manipulator operates erroneously quickly cut off power by turning the power supply off.

ⅸ) ⚠ while the manipulator is in operation keep out of its workspace; ensure no objects enter the workspace during operations.

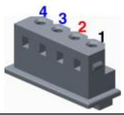
2.2 Preparation

ⅰ) Power Supply

The manipulator requires 24V for operations. Ensure the power supply is capable of supplying 24V and 15A or higher.

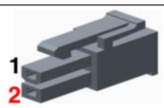
ⅱ) 4P Cable

The 4P Cable connects the manipulator and USB2Dynamixel.

4P Cable		
connector		
Pin array	1	GND
	2	VDD
	3	DATA +
	4	DATA -
cable		
Connector specifications		
PCB Header	MOLEX 22-03-5045	
Cable (Housing)	MOLEX 50-37-5043	
Cable (Terminal)	MOLEX 08-70-1039	

ⅲ) Power Cable

The power cable supplies power to the manipulator.

Power Cable		
Connector		
Pin array	1	GND
	2	VCC
Cable		
Connector specifications		
PCB Header	MOLEX 39-28-1023	
Cable (Housing)	MOLEX 39-01-2020	
Cable (Terminal)	MOLEX 39-00-0038	

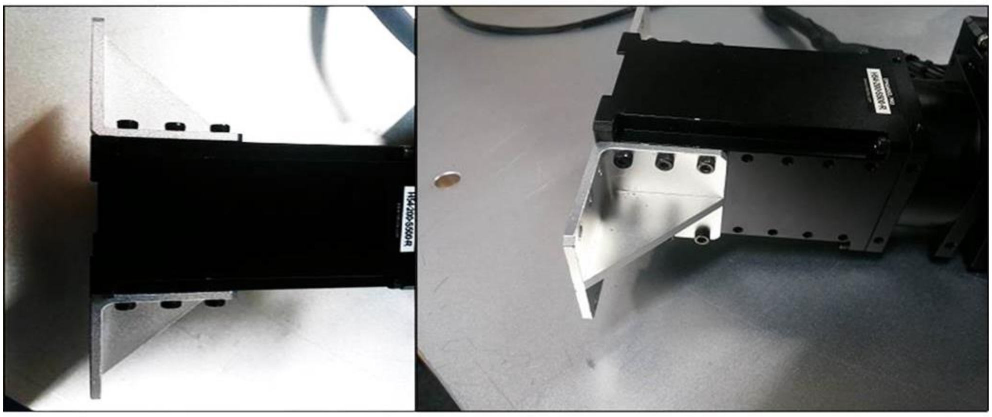
For additional power or 4P cables contact ROBOTIS or obtain them with the specifications listed above.

ⅳ) **USB2Dynamixel**

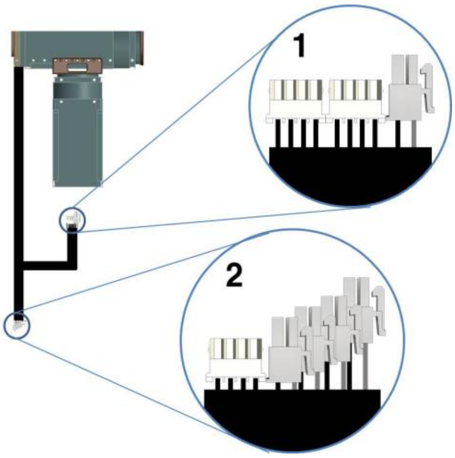
The USB2Dynamixel sends ArmSDK commands to the manipulator. Connect the USB2Dynamixel to the PC via USB hub.

2.3 Installation of Manipulator

⚠ The content below is based on an optional base plate and differs from the actual base plate.

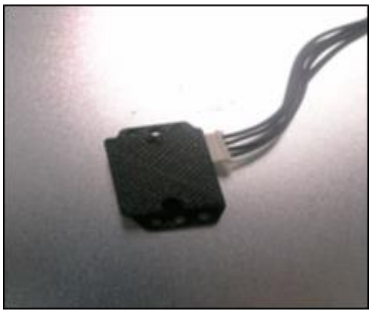


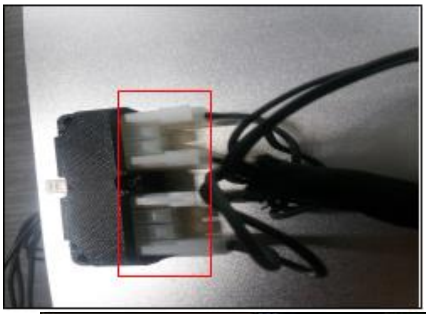
- Rest and fix joint 1 of the manipulator.



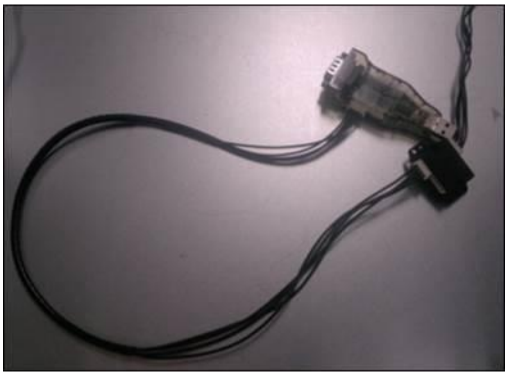
- The photo on the left is the external wiring for the arm. Label "1" shows a pair of 4P cables and power connector; these connect to joint 1 as shown on the right picture.

- Label "2" shows a 4P connector and 4 power connectors and these connect to the power expansion hub and the 4P cable connects to the extension.





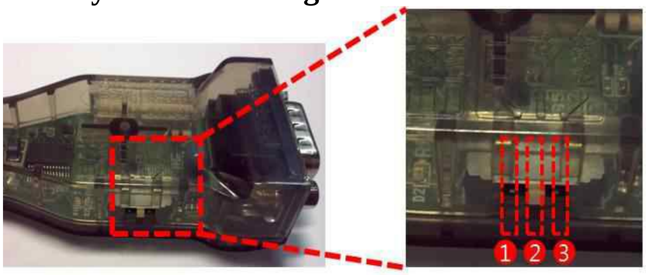
- Once connections are complete fix the arm to the base plate as shown on the photo above. The joint fixed to the plate is joint 1.



- Connect USB2Dynamixel to the hub with 4P cable; connect another port of the 4P hub to the extension.

- ⚠ Connect the USB2Dynamixel to the PC via USB hub. *The USB hub acts as an isolator to protect the PC from any possible unexpected surges caused by arm action*.

2.4 USB2Dynamixel Setting



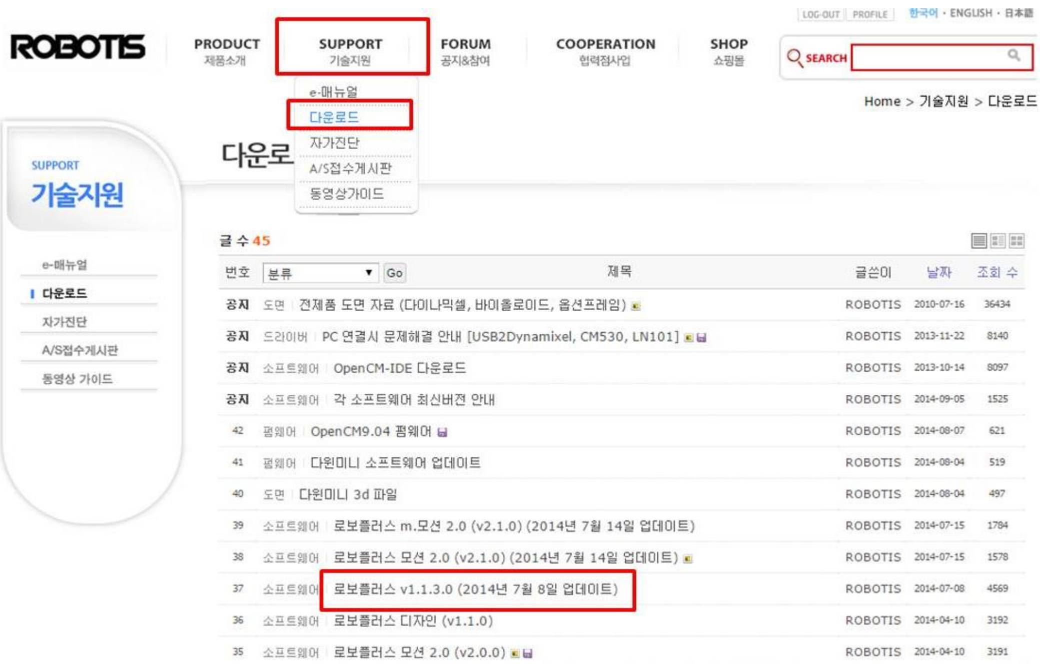
1. TTL communications	AX, 3-pin MX; communicate with 3-pin Dynamixel
2. RS485 communications	RX, 4-pin MX and **Pro**; communicate with 4-pin Dynamixel.
3. RS232 communications	CM-5and CM-510; communicate with these controllers. Communicate with other RS-232 devices.

The manipulator is based on RS-485 communications so make sure to set the dongle to 485.

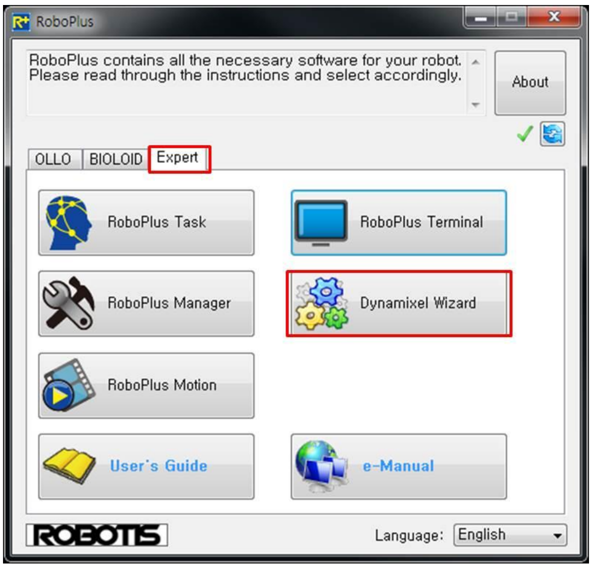
2.5 Manipulator Test on Dynamixel Wizard

- Test the arm with Dynamixel Wizard to check for any anomalies. DynamixelWizard is included in RoboPlus suite. RoboPlus can be downloaded from ROBOTIS home page's 'Support -> Downloads.

(do not download RoboPlus v2.0. instead, get RoboPlus v1.0).



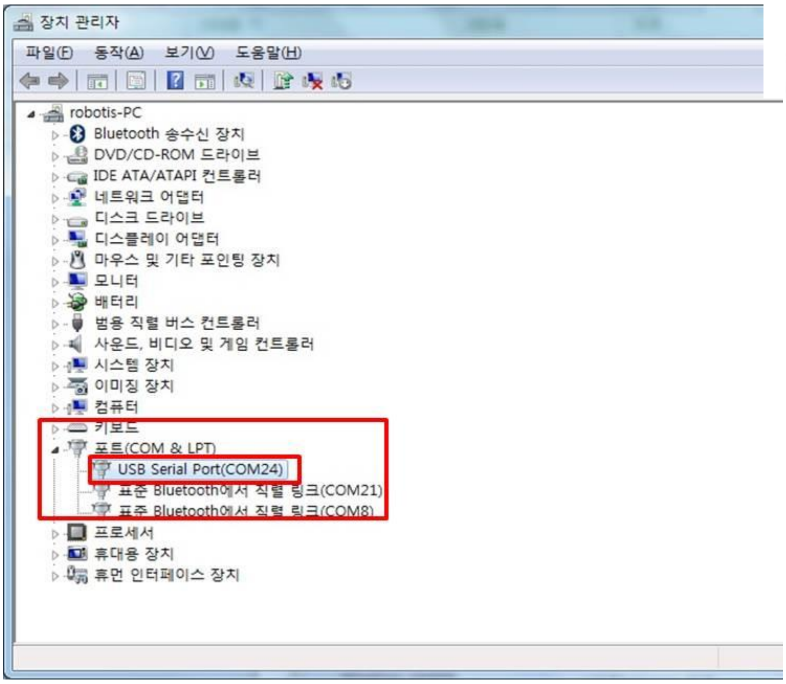
- Install and run RoboPlus; click on Dynamixel Wizard button to start Dynamixel Wizard.

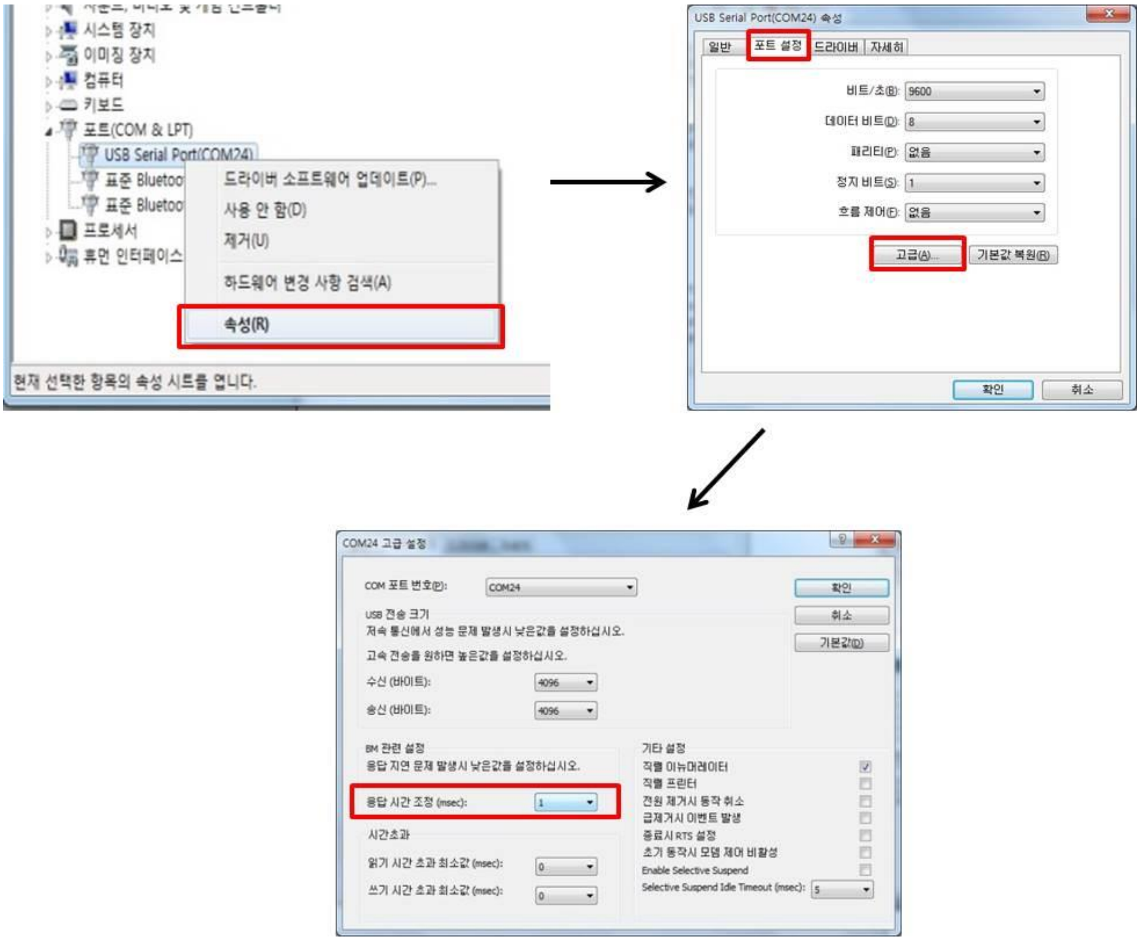


ⅰ) Moving the Manipulator

- **<u>Before starting Dynamixel Wizard ensure the arm is fixed to the base plate; then extend the arm. Otherwise; it may cause physical harm.</u>**

- USB2Dynamixel to the PC after wiring is complete. From the PC check the COM port number of USB2Dynamixel.

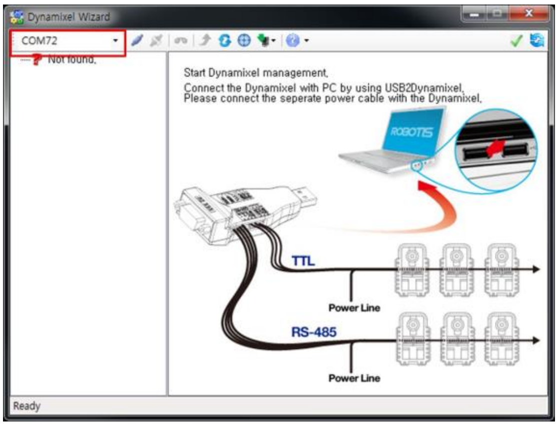


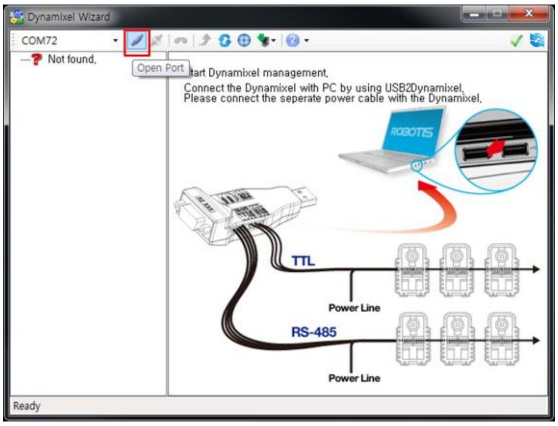


- select the Port Settings tab and click on the Advanced button-> change the latency time from 16 (default) to 1.

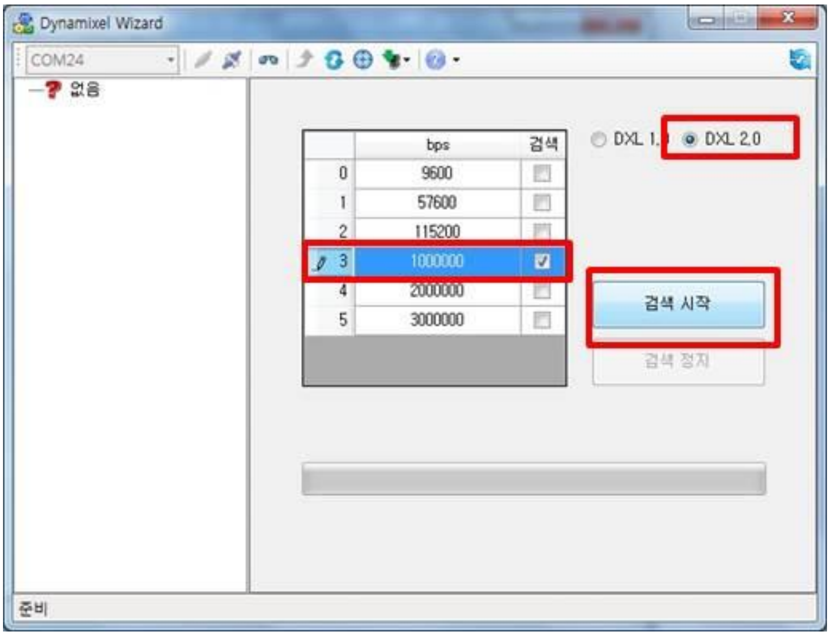
- After changing the COM port settings supply the 24V to the arm (of course, this means wiring is complete).

⚠ **Always ensure before powering on. While power is on du not change wires; otherwise it may cause undesired operations.**

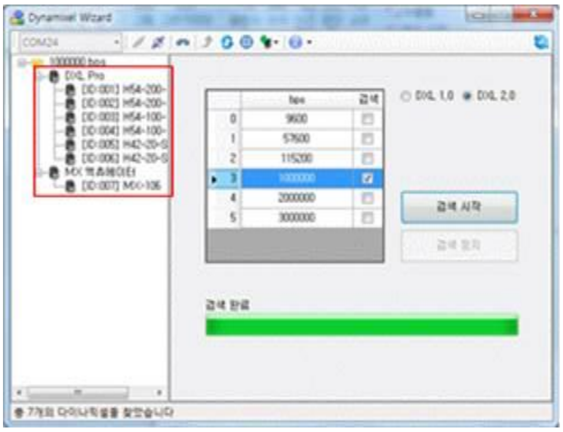


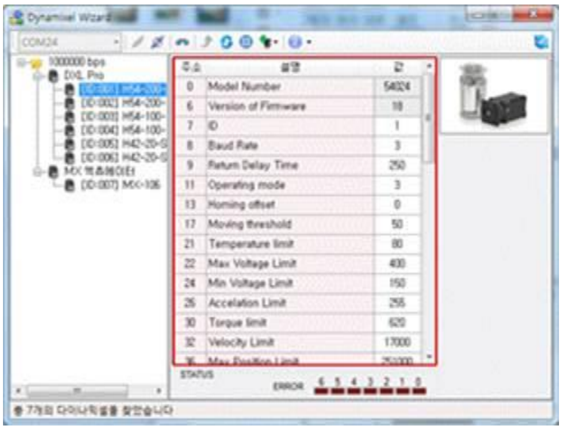


- The picture on the left is the COM port number of USB2Dynamixel (which should be connected to the arm). Click on the  to continue.

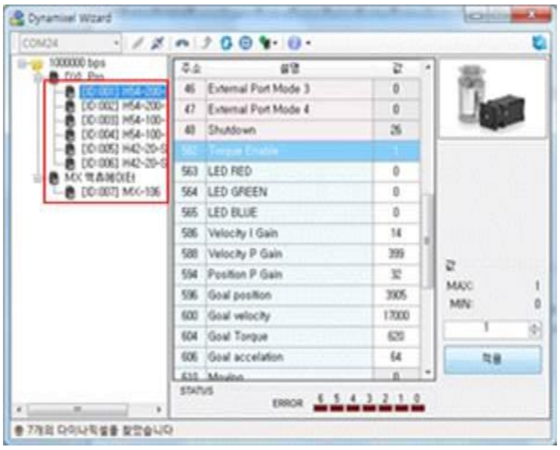


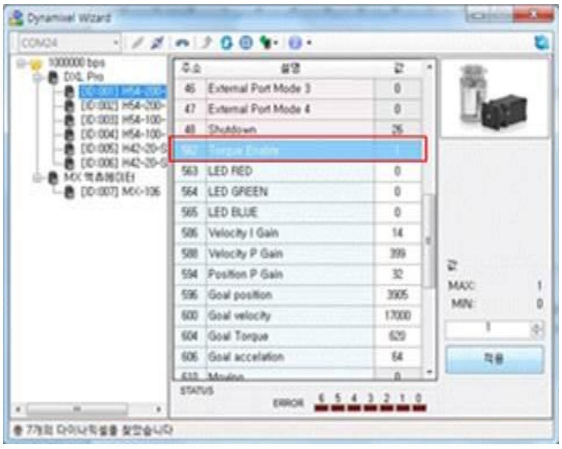
- Once connected make sure that 1000000bps box is checked and “DXL 2.0” is selected. Then click on Search. The arm’s default baud rate is 1 Mbps.



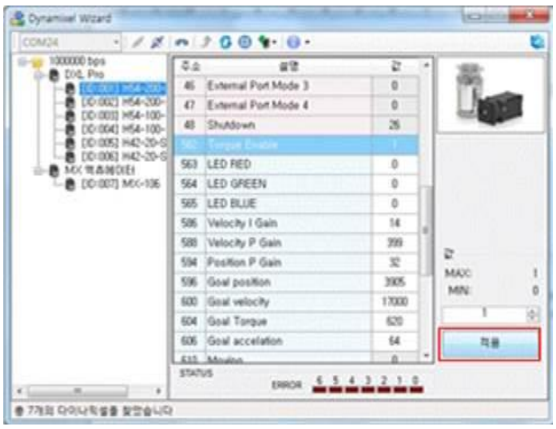


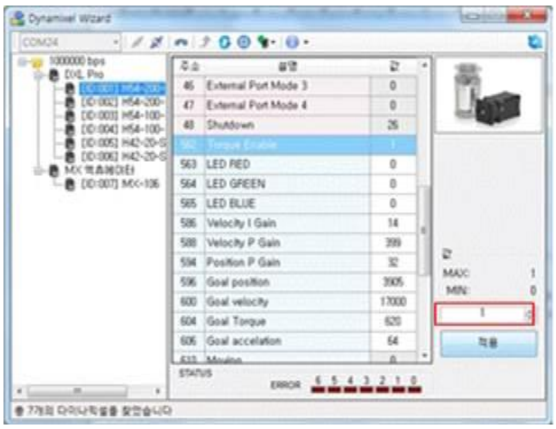
- Once search is complete the arm’s components (Dynamixel PROs) are listed on the left. Click on an individual Dynamixel PRO to display the contents of its Control Table.

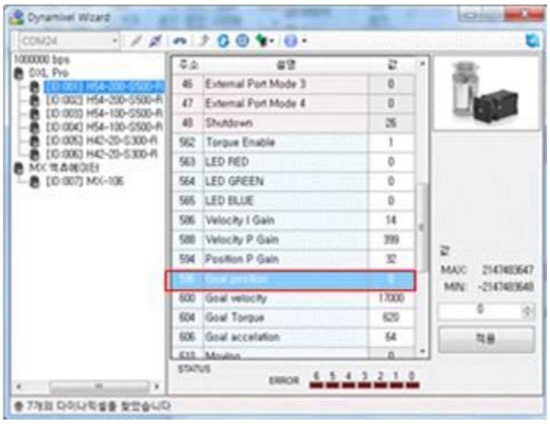


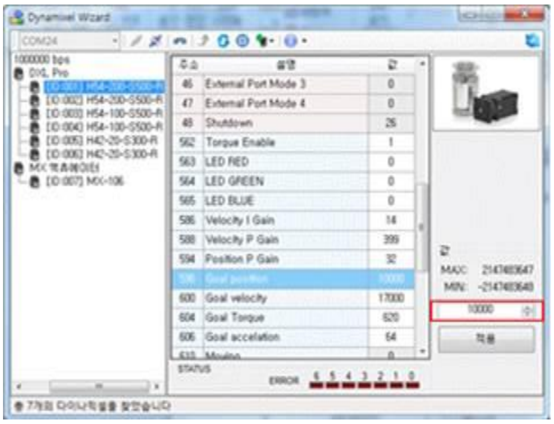


- Dynamixel Pro will only move (operate) when Torque Mode is on. So always make sure the Torque Mode is on prior to sending moving commands. Torque Enable is located on address number 562. A value of 1 means on and 0 means off.

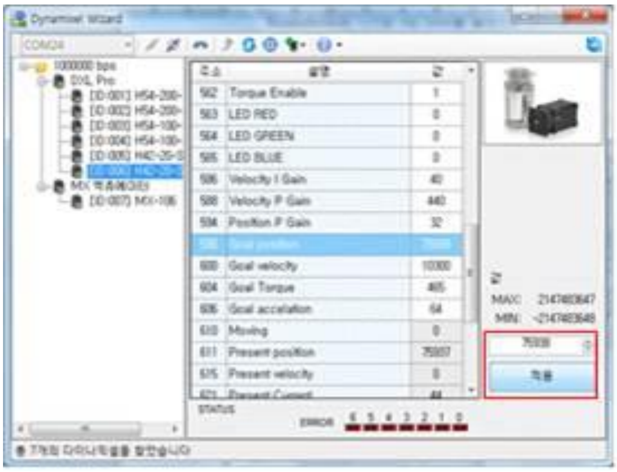




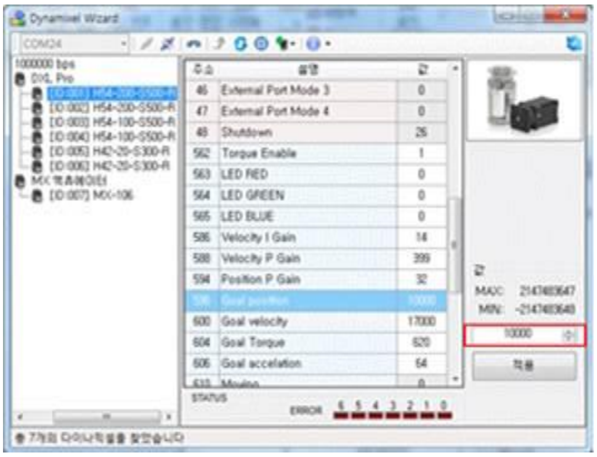




- Turn ‘Torque Enable’ on to all joints. The pose of the arm will become rigid (check by applying a small force). Afterwards click on joint 6.



- now verify the arm moves properly by changing Goal Position. Move the end effector (joint 6) +90 degrees. To move joint 6 to +90 degrees set Goal Position of the Dynamixel PRO model H42-20-S300-R to 75938 or L42-10-S300-R to 1024.
- once Goal Position has been set visually verify that joint 6 has rotated 90 degrees.
- to actually get Dynamixel PRO to move to its respective Goal Position click on the Apply button after setting the value. If no movement happen make sure Torque Enable is turned on (set to 1).
- Set Goal Position back to 0 to set position to its original position.



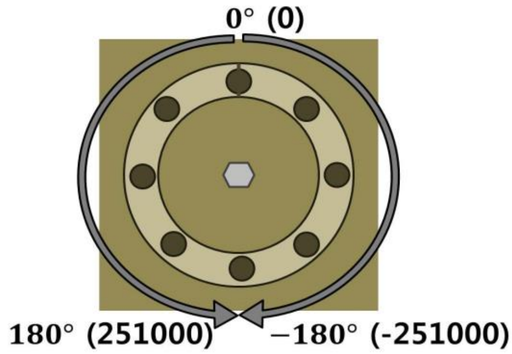
- click on ID. Set the Goal Position to 1000 (500 for L42 model).
- to actually get Dynamixel PRO to move to its respective Goal Position click on the Apply button after setting the value. If no movement happen make sure Torque Enable is turned on (set to 1).
- Set Goal Position back to 0 to set position to its original position..
- do the same procedure for joint 2 through 6.

ⅱ) Goal Position values with respect to rotation

- Goal Position value determines the rotational position of Dynamixel PRO.

<table>
<tr><th>Model Name</th><th>Relationship between angle(deg) and goal position value</th></tr>
<tr><td>H54-200-S500-R
H54-100-S500-R</td><td>

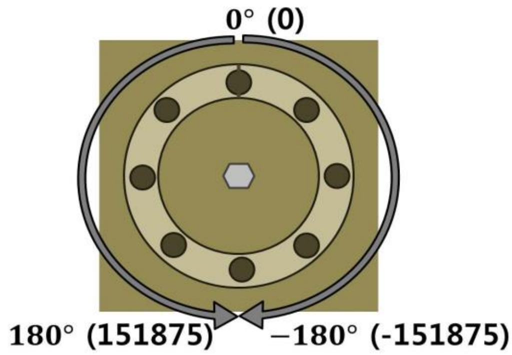
-180 ~ 180 (deg) → -251000 ~ 251000



$$Goal\ Angle(\mathrm{deg}) = Goal\ Position\ Value \times \frac{180°}{251000}$$

</td></tr>
<tr><td>H42-20-S300-R</td><td>

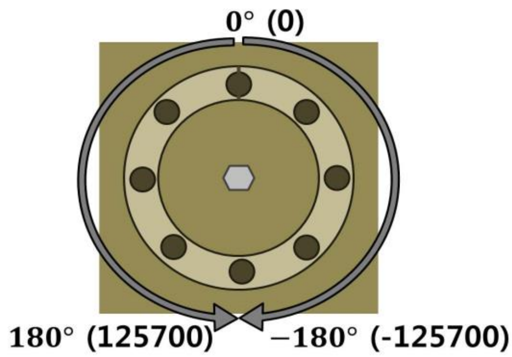
-180 ~ 180 (deg) → -151875 ~ 151875



$$Goal\ Angle(\mathrm{deg}) = Goal\ Position\ Value \times \frac{180°}{151875}$$

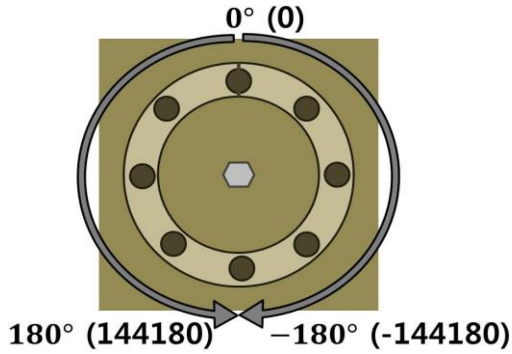
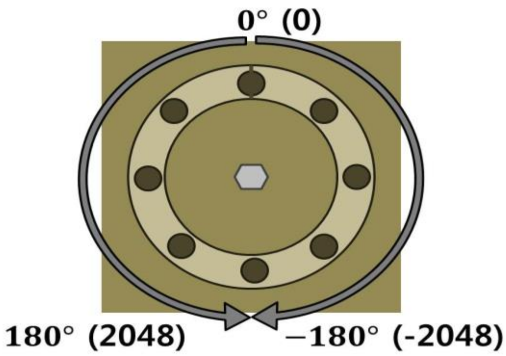
</td></tr>
<tr><td>L54-50-S500-R</td><td>

-180 ~ 180 (deg) → -125700 ~ 125700



$$Goal\ Angle(\mathrm{deg}) = Goal\ Position\ Value \times \frac{180°}{125700}$$

</td></tr>
</table>

L54-30-S500-R	*-180 ~ 180 (deg) → -144180 ~ 144180*  $Goal\ Angle(\mathrm{deg}) = Goal\ Position\ Value \times \frac{180^{\circ}}{144180}$
L42-20-S300-R	*-180 ~ 180 (deg) → -2048 ~ 2048*  $Goal\ Angle(\mathrm{deg}) = Goal\ Position\ Value \times \frac{180^{\circ}}{2048}$

2.6 How to use Robotis Manipulator SDK

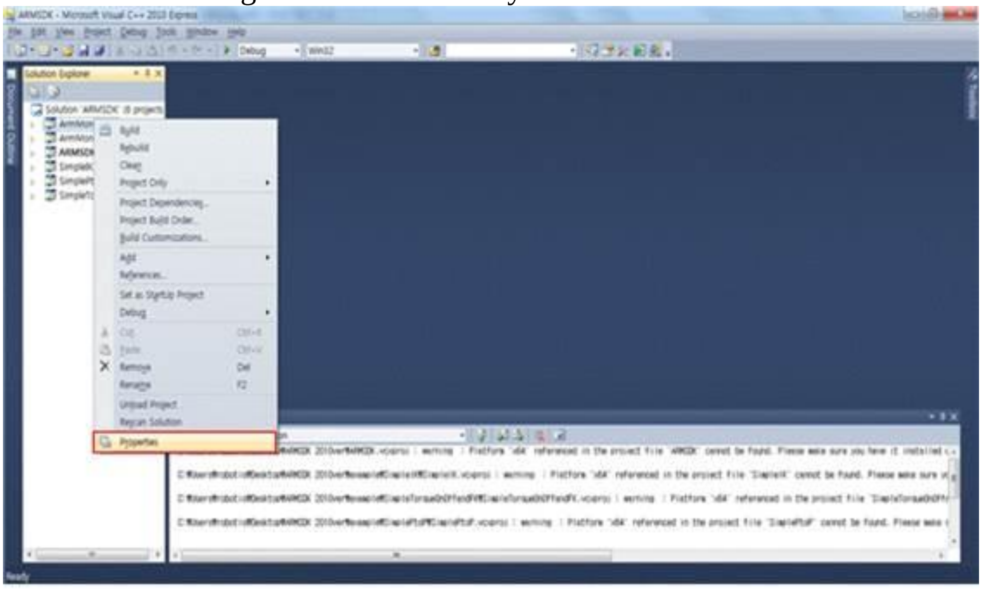
ⅰ) Preparation Before using Robotis Manipulator SDK

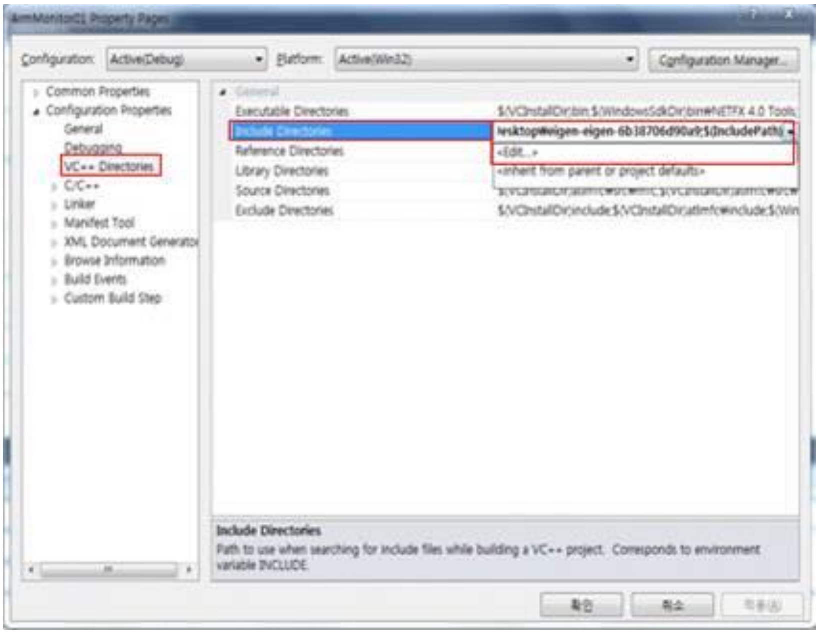
The following are pre-requisites for the ArmSDK.

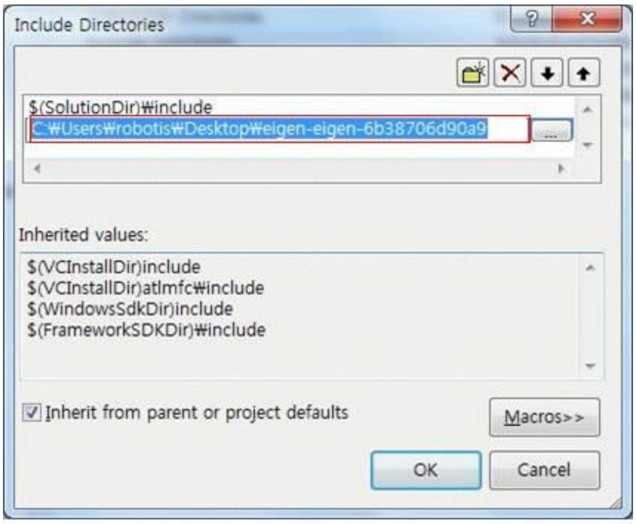
Eigen Package(http://eigen.tuxfamily.org, version 3.0.6 or Later)

ⅱ) Installation Package

- Download and unzip Eigen Package.
- Start Visual Studio go to “Project Properties -> VC++ Directories -> Include Directories” set Eigen’s source directory.







- repeat procedure (i)~(ii) to include the examples and include directories.
- once preparations are complete press the F7 key to compile and build.

3. Examples

The following examples are included with the ArmSDK; ArmMonitor01, ArmMonitor02, SimpleP2P, SimpleIK, and SimpleTorqueOnOffandFK.

3.1 ArmMonitor

In ArmMonitor allows viewing of a joint current position, target position, end effector's pose, and joint paramenters (Velocity, Acceleration, Position P, I, D Gain, Velocity P, I Gain). Change the values from the table below to see changes.

- Joint's values table

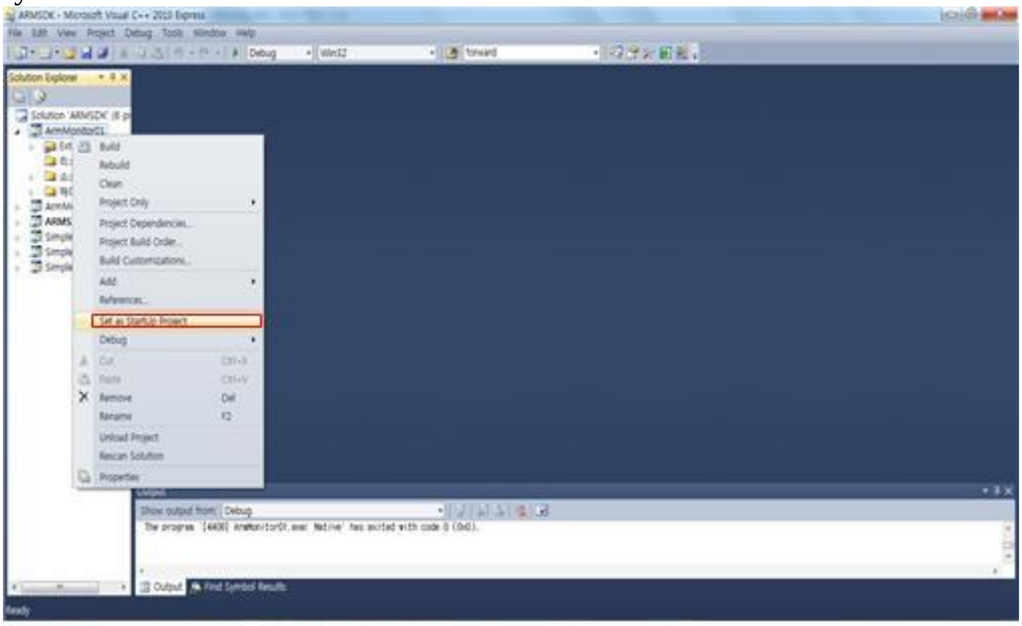
Note that a press of [and { key denote decrease by shown units and }] denote increase by shown units.

Key / Value	[	]	{	}
Position	$-\frac{\pi}{180}$	$+\frac{\pi}{180}$	$-2\times\frac{\pi}{180}$	$+2\times\frac{\pi}{180}$
Velocity	-100	+100	-200	+200
Acceleration	-1	+1	-2	+2
Position P Gain	-1	+1	-2	+2
Position I Gain	-1	+1	-2	+2
Position D Gain	-1	+1	-2	+2
Velocity P Gain	-1	+1	-2	+2
Velocity I Gain	-1	+1	-2	+2

ⅰ) How to Use ArmMonitor

(1) ArmMonitor01

- To start ArmMonitor01 create a new project; once created press the Ctrl + F5 keys to run.



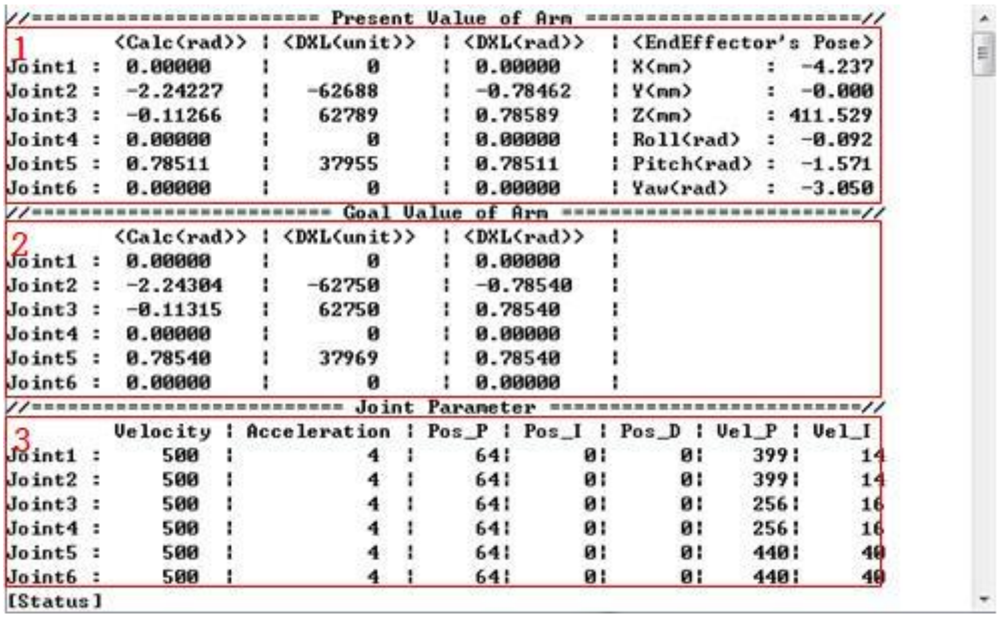
You will need to enter the COM port number and baud rate. Simply enter the values and “Succeed to open USB2Dynamixel” should appear onscreen followed by “Press any key to move first pose.” Use the keyboard to move the arm.

```
Input COM port number : 24
Input baud number : 3
Succeed to open USB2Dynamixel
Press any key to move first pose
```

The following table is the list of baud rate values and its corresponding speed; Robotis Maniupulator default value is 3 (1Mbps).

Baudrate Number	baudrate
0	2,400 bps
1	57,600 bps
2	115,200 bps
3	1,000,000 bps
4	2,000,000 bps
5	3,000,000 bps

The photo below is the arm in its “arrival” pose.



```
//========================= Present Value of Arm =========================//
1         <Calc(rad)> | <DXL(unit)>  | <DXL(rad)>  | <EndEffector's Pose>
Joint1 :  0.00000     |        0     | 0.00000     | X(mm)      :  -4.237
Joint2 :  -2.24227    |   -62688     | -0.78462    | Y(mm)      :  -0.000
Joint3 :  -0.11266    |    62789     | 0.78589     | Z(mm)      : 411.529
Joint4 :  0.00000     |        0     | 0.00000     | Roll(rad)  :  -0.092
Joint5 :  0.78511     |    37955     | 0.78511     | Pitch(rad) :  -1.571
Joint6 :  0.00000     |        0     | 0.00000     | Yaw(rad)   :  -3.050
//========================== Goal Value of Arm ==========================//
2         <Calc(rad)> | <DXL(unit)>  | <DXL(rad)>  |
Joint1 :  0.00000     |        0     | 0.00000     |
Joint2 :  -2.24304    |   -62750     | -0.78540    |
Joint3 :  -0.11315    |    62750     | 0.78540     |
Joint4 :  0.00000     |        0     | 0.00000     |
Joint5 :  0.78540     |    37969     | 0.78540     |
Joint6 :  0.00000     |        0     | 0.00000     |
//========================== Joint Parameter ==========================//
3         Velocity | Acceleration | Pos_P | Pos_I | Pos_D | Vel_P | Vel_I
Joint1 :      500  |            4 |    64|      0|      0|    399|     14
Joint2 :      500  |            4 |    64|      0|      0|    399|     14
Joint3 :      500  |            4 |    64|      0|      0|    256|     16
Joint4 :      500  |            4 |    64|      0|      0|    256|     16
Joint5 :      500  |            4 |    64|      0|      0|    440|     40
Joint6 :      500  |            4 |    64|      0|      0|    440|     40
[Status]
```

Press the Ctrl + F5 keys simultaneously and the screen should appears like the picture above

From ArmMonitor01 change the joint's target position and joint parameter to move the arm.

Use the directional keys to move cursor. Use the '[' '{' keys to lower values and ']' '}' to increase.

From the picture (from the screen output) with the red area with "1" it shows the joints current pose (Present Value) and end effector's pose.

The red area with "2" shows the target pose (Goal Value) for all joints.

The red area with "3" shows the parameters of all joints (Velocity, Acceleration, Position P Gain, I Gain, D Gain, Velocity P Gain, I Gain).

Values from joints 2 and 3, Calc<rad> and DYNAMIXEL<rad>, show on the red ares with "1" and "2" due to difference between point of origin and DH Configuration.

Calc<rad> is the calculated angle from DH and DYNAMIXEL<rad> from the servo's. The cursor and only control Goal Value Joint Parameter.

```
          <Calc(rad)> | <DXL(unit)>  | <DXL(rad)>  | <EndEffector's Pose>
Joint1 :  0.017548    |    1401      | 0.017548    | X(mm)       :   6.424
Joint2 :  -2.22568    |   -61363     | -0.76804    | Y(mm)       :   0.112
Joint3 :  -0.09565    |    64148     | 0.80290     | Z(mm)       : 408.635
Joint4 :  0.017463    |    1395      | 0.017463    | Roll(rad)   :  -2.618
Joint5 :  0.80261     |    38801     | 0.80261     | Pitch(rad)  :  -1.510
Joint6 :  0.002174    |    105       | 0.002174    | Yaw(rad)    :  -0.495
//======================== Goal Value of Arm ========================//
          <Calc(rad)> | <DXL(unit)>  | <DXL(rad)>  |
Joint1 :  0.017450    |    1394      | 0.017450    |
Joint2 :  -2.22559    |   -61356     | -0.76794    |
Joint3 :  -0.09570    |    64144     | 0.80285     |
Joint4 :  0.017450    |    1394      | 0.017450    |
Joint5 :  0.80285     |    38813     | 0.80285     |
Joint6 :  0.01745     |    844       | 0.01745     |
//========================= Joint Parameter =========================//
          Velocity | Acceleration | Pos_P | Pos_I | Pos_D | Vel_P | Vel_I
Joint1 :     500   |            4 |    64|      0|      0|    399|     14
Joint2 :     500   |            4 |    64|      0|      0|    399|     14
Joint3 :     500   |            4 |    64|      0|      0|    256|     16
Joint4 :     500   |            4 |    64|      0|      0|    256|     16
Joint5 :     500   |            4 |    64|      0|      0|    440|     40
Joint6 :     500   |            4 |    64|      0|      0|    440|     40
[Status]
```

The Goal Value of Arm의 Calc<rad> value (enclosed by the red frame) can be increased with the']' key. The unit is $\frac{\pi}{180} rad$

<u>Visually verify arm movement every time when changing position.</u>

(2) ArmMonitor02

⚠ Use of this example may pose safety risks. When testing the example keep a safe distance while able to cut power off in case of undesired operation.

ArmMonitor02 allows direct control of the end effector. Control the end effector is done by ComputeIK function where it moves each joint to its solution position (rad).

- EndEffector pose table

Pose \ Key	[	]	{	}
Position X	-2mm	+2mm	-4mm	+4mm
Position Y	-2mm	+2mm	-4mm	+4mm
Position Z	-2mm	+2mm	-4mm	+4mm
Orientation Roll	$-\frac{\pi}{180}$ rad	$+\frac{\pi}{180}$ rad	$-2\times\frac{\pi}{180}$ rad	$+2\times\frac{\pi}{180}$ rad
Orientation Pitch	$-\frac{\pi}{180}$ rad	$+\frac{\pi}{180}$ rad	$-2\times\frac{\pi}{180}$ rad	$+2\times\frac{\pi}{180}$ rad
Orientation Yaw	$-\frac{\pi}{180}$ rad	$+\frac{\pi}{180}$ rad	$-2\times\frac{\pi}{180}$ rad	$+2\times\frac{\pi}{180}$ rad

To setup and run ArmMonitor02 follow the same procedure as in ArmMonitor01.

As in ArmMonitor01 you will be asked to enter COM port number and baud rate. You should also see “Succeed to open USB2Dynamixel” followed by “Press any key to move first pose.” The arm moves to its initial pose.

```
Input COM port number : 24
Input baud number : 3
Succeed to open USB2Dynamixel
Press any key to move first pose
```

The photo below is the arm in its “arrival” pose.

```
//========================= Present Value of Arm =======================//
          <Calc(rad)> | <DXL(unit)>  | <DXL(rad)>  | <EndEffector's Pose>
Joint1 :  0.00000     |          0   |  0.00000    | X(mm)     :  -4.237
Joint2 :  -2.24306    |     -62751   |  -0.78541   | Y(mm)     :  -0.000
Joint3 :  -0.11314    |      62751   |  0.78541    | Z(mm)   2 : 411.529
Joint4 :  0.00000     |          0   |  0.00000    | Roll(rad)  :  -1.571
Joint5 :  0.78538     |      37968   |  0.78538    | Pitch(rad) :  -1.571
Joint6 :  0.00002     |          1   |  0.00002    | Yaw(rad)   :  -1.571
//========================= Goal Value of Arm ==========================//
          <Calc(rad)> | <DXL(unit)>  | <DXL(rad)>  | <EndEffector's Pose>
Joint1 :  0.00000     |          0   |  0.00000    | X(mm)     :  -4.239
Joint2 :  -2.24304    |     -62750   |  -0.78540   | Y(mm)     :  -0.000
Joint3 :  -0.11315    |      62750   |  0.78540    | Z(mm)   1 : 411.527
Joint4 :  0.00000     |          0   |  0.00000    | Roll(rad)  :  -0.927
Joint5 :  0.78540     |      37969   |  0.78540    | Pitch(rad) :  -1.571
Joint6 :  0.00000     |          0   |  0.00000    | Yaw(rad)   :  -2.214
//========================== Joint Parameter ===========================//
          Velocity | Acceleration | Pos_P | Pos_I | Pos_D | Vel_P | Vel_I
Joint1 :      500  |            4 |    64|      0|      0|   399|     14
Joint2 :      500  |            4 |    64|      0|      0|   399|     14
Joint3 :      500  |            4 |    64|      0|      0|   256|     16
Joint4 :      500  |            4 |    64|      0|      0|   256|     16
Joint5 :      500  |            4 |    64|      0|      0|   440|     40
Joint6 :      500  |            4 |    64|      0|      0|   440|     40
[Status]
```

The different values of the end effector depicted from the red areas with “1” and “2” (from the screen output image above) is due to the difference of Dynamixel Pro’s Goal Position and Present Position values (gear backlash) and DH with the point of origin. <u>1”1” shows the end effector’s pose via calculations from kinematics and “2” the actual pose.</u>

```
          <Calc(rad)> | <DXL(unit)>  | <DXL(rad)>  | <EndEffector's Pose>
Joint1 :  0.00000     |          0   |  0.00000    | X(mm)     :  -2.237
Joint2 :  -2.23822    |     -62365   |  -0.78058   | Y(mm)     :  -0.000
Joint3 :  -0.11305    |      62758   |  0.78550    | Z(mm)     : 411.529
Joint4 :  0.000841    |         67   |  0.000841   | Roll(rad)  :  -1.596
Joint5 :  0.78048     |      37731   |  0.78048    | Pitch(rad) :  -1.571
Joint6 :  -0.00265    |       -128   |  -0.00265   | Yaw(rad)   :  -1.528
//========================= Goal Value of Arm ==========================//
          <Calc(rad)> | <DXL(unit)>  | <DXL(rad)>  | <EndEffector's Pose>
Joint1 :  0.00005     |          4   |  0.00005    | X(mm)     :  -2.233
Joint2 :  -2.23823    |     -62366   |  -0.78059   | Y(mm)     :  -0.000
Joint3 :  -0.11306    |      62758   |  0.78549    | Z(mm)     : 411.529
Joint4 :  0.02437     |       1947   |  0.02437    | Roll(rad)  :   1.571
Joint5 :  0.78064     |      37739   |  0.78064    | Pitch(rad) :  -1.569
Joint6 :  -0.01722    |       -833   |  -0.01722   | Yaw(rad)   :   1.571
//========================== Joint Parameter ===========================//
          Velocity | Acceleration | Pos_P | Pos_I | Pos_D | Vel_P | Vel_I
Joint1 :      500  |            4 |    64|      0|      0|   400|     15
Joint2 :      500  |            4 |    64|      0|      0|   400|     15
Joint3 :      500  |            4 |    64|      0|      0|   257|     17
Joint4 :      500  |            4 |    64|      0|      0|   257|     17
Joint5 :      500  |            4 |    64|      0|      0|   441|     41
Joint6 :      500  |            4 |    64|      0|      0|   441|     41
[Status]
```

Press the] key to increase the end effector’s pose value by $\frac{\pi}{180} rad$; X increases by 2mm.

<u>Visually verify arm movement every time when changing position.</u>

ⅱ) Arm Monitor Source Description

(1) **cmd_process.cpp**

- void DrawPage(void)

```
printf("//======================== Present Value of Arm =======================//\n");
printf("            <Calc(rad)> | <DXL(unit)>   | <DXL(rad)>   | <EndEffector's Pose> \n");
printf("Joint1 :                |               |              | X(mm)       :          \n");
printf("Joint2 :                |               |              | Y(mm)       :          \n");
printf("Joint3 :                |               |              | Z(mm)       :          \n");
printf("Joint4 :                |               |              | Roll(rad)   :          \n");
printf("Joint5 :                |               |              | Pitch(rad)  :          \n");
printf("Joint6 :                |               |              | Yaw(rad)    :          \n");
printf("//========================= Goal Value of Arm ========================//\n");
printf("            <Calc(rad)> | <DXL(unit)>   | <DXL(rad)>   |                     \n");
printf("Joint1 :                |               |              |                     \n");
printf("Joint2 :                |               |              |                     \n");
printf("Joint3 :                |               |              |                     \n");
printf("Joint4 :                |               |              |                     \n");
printf("Joint5 :                |               |              |                     \n");
printf("Joint6 :                |               |              |                     \n");
printf("//========================== Joint Parameter =========================//\n");
printf("            Velocity | Acceleration | Pos_P | Pos_I | Pos_D | Vel_P | Vel_I   \n");
printf("Joint1 :             |              |       |       |       |       |        \n");
printf("Joint2 :             |              |       |       |       |       |        \n");
printf("Joint3 :             |              |       |       |       |       |        \n")
printf("Joint4 :             |              |       |       |       |       |        \n");
printf("Joint5 :             |              |       |       |       |       |        \n");
printf("Joint6 :             |              |       |       |       |       |        \n");
printf("[Status]                                                                     \n");

GotoCursur(GOAL_JOINT1_ROW, CALC_ANGLE_RAD_COL);
```

- the above is DrawPage code for ArmMonitor.

void GotoCursur(int row, int col)

```
COORD pos={col, row};
SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
```

- This allows the cursor to jump between rows and columns.

- `void MoveUpCursor()`
- `void MoveDownCursor()`
- `void MoveLeftCursor()`
- `void MoveRightCursor()`

these 4 functions allows the directional keys to control cursor location..

- `void UpDownValue(int dir)`

```
giOldRow = giRow;
giOldCol = giCol;

GotoCursur(STATUS_ROW, STATUS_COL);
printf("                                                  ");

if(giRow <= GOAL_JOINT6_ROW)
   UpDownGoalValue(dir);

else if((giRow > GOAL_JOINT6_ROW) && (giRow <= PARAMETER_JOINT6_ROW))
   UpDownJointParameter(dir);

WriteValue();

GotoCursur(giOldRow, giOldCol);
giRow = giOldRow;
giCol = giOldCol;
```

UpDownValue allows cursor to increase/decrease values.

- `void initialize(void)`

initialize() function described below.

```
gpArmComm = new Pro_Arm_Comm_Win()
```

gpArmComm is a class of Pro_Arm_Comm_Win. Pro_Arm_Comm_Win includes ID and baud num-related aspects.

```
gpArmComm->DXL_Set_Init_Param(Port, Baud)
```

Sets Port Number　Baudrate from pointer

gpRobotisArm->AddJoint	double LinkLength
	double LinkTwist
	double JointOffset
	double JointAngle
	double MaxAngleInRad
	double MinAngleInRad
	int MaxAngleValue
	int MinAngleValue
	double MaxAngleLimitInRad
	double MinAngleLimitInRad
	unsigned int Dynamixel_ID

input each joint's DH joint parameters.

```
gpArmComm->Arm_ID_Setup(gpRobotisArm->GetArmIDList())
```

from AddJoint rearrange ID List.

```
gpArmKinematics = new Kinematics(gpRobotisArm)
```

gpRobotisArm->AddJoint (RobotInfo Class) generates Kinematics Class.

```
gpArmKinematics->SetMaximumNumberOfIterationsForIK(60)
```

this fuction calculates the IK's max number of iterations. In this case 60.

gpArmKinematics->SetConvergenceCondition(0.001, 5.0)

sets convergence for IK. 1st value to determine solution; second value maximum allowed.

gvdGoalCalculationAngleRad.resize(gpRobotisArm->GetRobotInfo()->size())

setup target pose value (rad).

gvdRealCalculationAngleRad.resize(gpRobotisArm->GetRobotInfo()->size())

current pose value (rad).

gvdGoalDynamixelAngleRad.resize(gpRobotisArm->GetRobotInfo()->size())

target joint's position value (rad)

gvdRealDynamixelAngleRad.resize(gpRobotisArm->GetRobotInfo()->size())

actual joint's position value (rad).

gviGoalDynamixelAngleUnit.resize(gpRobotisArm->GetRobotInfo()->size())

actual joint's target position value (value).

gviRealDynamixelAngleUnit.resize(gpRobotisArm->GetRobotInfo()->size())

actual joint's position value.(value).

gviPositionPGain.resize(gpRobotisArm->GetRobotInfo()->size())

Position P Gain value.

gviPositionIGain.resize(gpRobotisArm->GetRobotInfo()->size())

Position I Gain value.

gviPositionDGain.resize(gpRobotisArm->GetRobotInfo()->size())

Position D Gain value.

```
gviVelocityPGain.resize(gpRobotisArm->GetRobotInfo()->size())
```

Velocity P Gain value.

```
gviVelocityIGain.resize(gpRobotisArm->GetRobotInfo()->size())
```

Velocity I Gain value.

```
gviDynamixelVelocity.resize(gpRobotisArm->GetRobotInfo()->size())
```

Velocity value.

```
gviDynamixelAcceleration.resize(gpRobotisArm->GetRobotInfo()->size())
```

Acceleration value

```
gvdGoalCalculationAngleRad = gpArmKinematics->GetCurrentAngle();
gvdRealCalculationAngleRad = gpArmKinematics->GetCurrentAngle();
```

from gpArmKinematics (mCurrentAngle) current pose value initialize gvdGoal CalculationAngleInRad and gvdRealCalculationAngleInRad.

```
gvdAngleGapCalcandDynamixelRad.resize(gpRobotisArm->GetRobotInfo()->size())
gvdAngleGapCalcandDynamixelRad<< 0.0, ML_PI_2 - 6.4831 * ML_PI/ 180.0, ML_PI_4 +
6.4831 * ML_PI/ 180.0,    0.0,    0.0,    0.0
```

This function has been introduced due to the differences between point of origin and actual joints' point of origin from the DH Configuration. Once the size of angle adjustment has been assigned per joint enter the difference between point of origin and the joint actual point of origin. The values above are default values (ML_PI is in $\pi\ rad$, ML_PI_2 in $\frac{\pi}{2} rad$, and ML_PI_4 in $\frac{\pi}{4} rad$.)

```
gvdGoalDynamixelAngleRad<< 0.0, ML_PI/4.0, -ML_PI/4.0,0.0, -ML_PI/4.0, 0.0;
```

GoalDynamixelAngleRad is the initial pose default values.

```
gviPositionPGain.fill(DEFAULT_POSITION_P_GAIN)
gviPositionIGain.fill(DEFAULT_POSITION_I_GAIN)
gviPositionDGain.fill(DEFAULT_POSITION_D_GAIN)
```

Position P, I, D Gain functions. Default P gain value is 64 이고; I and D Gain are 0. the **.fill** contains every joint's PID values individually.

```
gpArmComm->Arm_Torque_On();
```

This function gets initialized before moving the arm to its initial pose.

```
gpArmComm->Arm_Set_Position_PID_Gain(DEFAULT_POSITION_P_GAIN,
                                     DEFAULT_POSITION_I_GAIN,
                                     DEFAULT_POSITION_D_GAIN);
```

Sets the manipulator joints' PID gain values..

```
gpArmTrajectory = new TrajectoryGenerator(gpArmKinematics)
gpArmTrajectory->Set_P2P(
    gvdRealDynamixelAngleRad-gvdAngleGapCalcandDynamixelRad,
    gvdGoalDynamixelAngleRad-gvdAngleGapCalcandDynamixelRad,
    5.0, 1.0)
```

The generated Kinematics, StartPose, EndPose, TotalTime, AccelTime get inputted into the trajectory. Trajectory is generated via P2P. The StartPose is the current pose and EndPose is ArmMonitor's initial pose. TotalTime is 5.0sec where AccelTime is 1.0sec. **For more information on trajectory generation please go to 4.2 How to Program and 6.3 MotionEngine's Trajectory Generator.**

```
gpMotionPlayer = new MotionPlay(gpArmKinematics, gpArmTrajectory);
```

MotionPlay's CurrentTime, ElapsedTime get initialized and setp up MotionProfile. These are required variables for kinematics and trajectory's motion.

```
gpMotionPlayer->Set_Time_Period(5);
```

Motion's time period in msec.

```
_tempMotionTimer.Start();
gvdGoalCalculationAngleRad = gpMotionPlayer->NextStep(&ErrorStatus);
gvdGoalDynamixelAngleRad = gvdGoalCalculationAngleRad +
                              gvdAngleGapCalcandDynamixelRad;
gviGoalDynamixelAngleUnit = gpRobotisArm
                                  ->Rad2Value(gvdGoalDynamixelAngleRad);
CommResult = gpArmComm
          ->Arm_Set_JointPosition(gviGoalDynamixelAngleUnit);
gvdGoalCalculationAngleRad = gpMotionPlayer->NextStep(&ErrorStatus)
gvdGoalDynamixelAngleRad = gvdGoalCalculationAngleRad +
                             gvdAngleGapCalcandDynamixelRad
_tempMotionTimer.Stop();
_tempMotionTimer.Wait(Period - _tempMotionTimer.GetElapsedTime());
```

The functions above have set motion time periods where functions are performed via while loop during their duration.

First, the set Control Time Period gets matched.

```
_tempMotionTimer.Start();
      ...
      ...
_tempMotionTimer.Stop();
_tempMotionTimer.Wait(Period - _tempMotionTimer.GetElapsedTime());
```

Measure elapsed start and stop time then subtract its difference with elapsed calculated time in set Control Time Period(5msec in this case).

The target pose from the current step obtained from the algorithm below.

```
gvdGoalCalculationAngleRad = gpMotionPlayer->NextStep(&ErrorStatus);
gvdGoalDynamixelAngleRad = gvdGoalCalculationAngleRad +
                              gvdAngleGapCalcandDynamixelRad;
gviGoalDynamixelAngleUnit = gpRobotisArm
                                ->Rad2Value(gvdGoalDynamixelAngleRad);
CommResult = gpArmComm
                        ->Arm_Set_JointPosition(gviGoalDynamixelAngleUnit);
gvdGoalCalculationAngleRad = gpMotionPlayer->NextStep(&ErrorStatus)
gvdGoalDynamixelAngleRad = gvdGoalCalculationAngleRad +
                             gvdAngleGapCalcandDynamixelRad
```

First, NextStep gets the current step's target angles, which are from the D-H Configuration. However, the actual Dynamixel PRO start point and the D-H Configuration's differ. This difference is taken into account and each joint Goal

Position Value is set again with Rad2Value, the resulting Arm_Set_JointPosition moves the manipulator.

If successful , the manipulator's communications Arm_Set_JointPosition returns a value of 1.

```
gvdRealDynamixelAngleRad =
            gpRobotisArm->Value2Rad(gviRealDynamixelAngleUnit);
gvdRealCalculationAngleRad= gvdRealDynamixelAngleRad
                                    -   gvdAngleGapCalcandDynamixelRad;
```

After motion is complete it print's the joint's actual pose(rad) and calculated pose(rad).

```
gpArmKinematics->Forward(gvdRealCalculationAngleRad, &gRealPose);
gpArmKinematics->Forward(gvdGoalCalculationAngleRad, &gGoalPose);
```

Factor's current angle and goal angle to Forward Kinematics. The end effectot's actual and goal pose (gRealPose, gGoalPose) can be assigned.

```
gpArmComm->Arm_Set_JointAcceleration(DEFAULT_JOINT_ACCELERATION);
gviDynamixelAcceleration.fill(DEFAULT_JOINT_ACCELERATION);
```

Sets every joint's acceleration value individually. DEFAULT_JOINT_ACCELERATION has 4 values

```
gpArmComm->Arm_Set_JointVelocity(DEFAULT_JOINT_VELOCITY);
gviDynamixelVelocity.fill(DEFAULT_JOINT_VELOCITY);
```

Sets every joint's velocity value individually. DEFAULT_JOINT_VELOCITY 9000 values.

```
void UpDownGoalValue(int dir);
void UpDownJointParameter(int dir);
void UpDownValue(int dir);
```

UpdownGoalValue allows change in goal pose, UpDownJointParameter allows changes to joint's parameters (Velocity, Acceleration, Position P, I, D Gain, Velocity P, I Gain). UpDownValue combines both functions.

3.2 SimplePtoP

> ⚠ **Product may move fast with this example. When testing this example keep a safe distance while able to cut power off in case of undesired operation**

SimplePtoP is the end effector’s move point (from P1 to P2).

ⅰ) How to Use SimplePtoP

To start SimplePtoP follow the same procedure for ArmMonitor. Then press the Ctrl + F5 keys to run.

```
Input COM port number : 7
Input baud number : 1
Succeed to open USB2Dynamixel
Press any key to move first pose
move to first pose
Press any key to start P2P Motion
_
```

You will be asked for COM port number and baud rate.

If succeded then you will see a ‘Succeed to open USB2Dynamixel’ followed by ‘Press any key to move first pose.' Press a key to move the arm to its initial pose. Then press a key to begin P2P Motion. The photo below is the arm in its initial pose.

```
Input COM port number : 7
Input baud number : 1
Succeed to open USB2Dynamixel
Press any key to move first pose
move to first pose
Press any key to start P2P Motion
start
Current Calculated Angle is
0.00063 -1.45702 -0.89792 0.00063 0.00063 0.00063
```

SimplePtoP displays the joints’ pose(rad). In SimplePtoP prssing the 'p' or ‘P' will cause motion to pause. Press the ESC key to end.

ⅱ) SimplePtoP Source Description

```
vecd P1, P2;
P1.resize(RobotisArm.GetRobotInfo()->size());
P2.resize(RobotisArm.GetRobotInfo()->size());
```

P1, P2 sets every joint's position.

```
P1.fill(0.0);
P1 -= gvdAngleGapCalcandDynamixelRad;
P2.fill(0.5);
P2 -= gvdAngleGapCalcandDynamixelRad;
```

P1.fill, P2.fill input every joint's position(rad) individually. Differences between DH Configuration's point of origin and actual point of origin are taken into consideration so P1 and P2 are to be adjusted accordingly.

```
ArmComm.Arm_Set_Position_PID_Gain(64, 0, 0);
```

Joint's Position P, I, and D gain values.respectively.

```
ArmComm.Arm_Set_JointVelocity(0);
```

Joint's velocity value .0 denotes max velocity.

```
ArmComm.Arm_Set_JointAcceleration(0);
```

Joint's acceleration value .0 denotes max velocity.

```
ArmTrajectory.ClearMF();
```

MotionProfile clears the set space.

```
ArmTrajectory.Set_P2P(P1, P2, 10.0, 0.5);
```

sets P1, P2(Start, EndPose). In this case P1 is 0.0 rad and P2 is 0.5 rad. Trajectory is from P1 to P2

```
ArmTrajectory.Set_P2P(P2, P1, 10.0, 0.5);
```

sets P1, P2(Start, EndPose). In this case P1 is 0.0 rad and P2 is 0.5 rad. Trajectory is from P2 to P1

```
MotionPlayer.All_Info_Reload();
```

MotionProfile calls Info(Robot, Kinematics, Trajectory).

```
MoionPlayer.Initialize();
```

MotionProfile, Step, are initialized.

```
MotionPlayer.Set_Time_Period(DEFAULT_Ctrl_TIME_PERIOD);
```

sets time period. For value lesser than 0 then a default value (=8) gets inputted.

3.3 SimpleIK

⚠ **Use of this example may pose safety risks. When testing the example keep a safe distance while able to cut power off in case of undesired operation.**

Allows operation of end effector's pose via position(X, Y, Z) and orientation(Roll, Pitch, Yaw). The keys for SimpleIK are q, w, e, r, t, y and a, s ,d ,f, g, h.

- EndEffector Pose table

Control / EndEffector	Position -> +5mm orientation -> + $(3\times\frac{\pi}{180})$rad	Position -> -5mm orientation -> - $(3\times\frac{\pi}{180})$rad
Position X	q	a
Position Y	w	s
Position Z	e	d
Orientation Roll	r	f
Orientation Pitch	t	g
Orientation Yaw	y	h

ⅰ) How to Use SimpleIK

To start SimpleIK start a new project just like SimplePtoP. Then press the Ctrl + F5 ekys to begin.

```
Input COM port number : 7
Input baud number : 1
Succeed to open USB2Dynamixel
Press any key to move first pose
```

In SimpleIK you will be asked for COM port and baud rate numbers. If succeeded you will see a 'Succeed to open USB2Dynamixel' followed by ‘Press any key to move first pose.' Press a key to begin. The arm moves to its initial pose as shown below.

```
Input COM port number : 7
Input baud number : 1
Succeed to open USB2Dynamixel
Press any key to move first pose
move to first pose
CurrentPose
-4.23677
6.496e-009
411.529
-3.1409
-1.5708
-0.000695224

CurrentPose
-1.53325e-009
     -2.24304
    -0.113151
 5.50717e-010
     0.785398
            0

Demonstration Start
```

This windows pops up after the arms moves to its initial pose. The values printed are the joints' angles(rad). Press the keys(ex : q, w....) to move the end effector.

```
Answer
 1.60557e-008
     -2.23096
    -0.113022
-2.49533e-008
     0.773196
  1.5934e-008

Answer
 1.84708e-009
     -2.21869
    -0.113265
 4.84405e-009
     0.761185
-7.41367e-009

Answer
 9.88268e-010
     -2.20624
     -0.11388
 6.25259e-009
     0.749368
-8.50075e-009
```

SimpleIK q key control the 3rd value.

Q controls the end effector position (X) by increasing delta(5mm)amounts.

<u>**Visually verify arm movement every time when changing position.**</u>

```
Answer
-2.53639e-006
     -2.20624
     -0.11388
  0.000437539
     0.749365
    0.0518977

Answer
-5.07326e-006
     -2.20624
     -0.11388
  0.000874298
     0.749346
     0.103796

Answer
-7.61084e-006
     -2.20624
     -0.11388
   0.00130914
     0.749311
     0.155695
```

Press the q and r keys 3 times each. The r key controls the end effector's roll. The orientation (Roll, Pitch, Yaw) change by $(3 \times \frac{\pi}{180})rad$ per keystroke.

<u>Visually verify arm movement every time when changing position.</u>

ⅱ) SimpleIK Source Description

```
if(temp == 'q') {
   DesiredPose = CurrentPose;
   DesiredPose.x += delta;
   ArmKinematics.ComputeIK(DesiredPose, &angle_rad, angle_rad, &ErrorStatus);
   if(ErrorStatus == ARMSDK_NO_ERROR)
   {
      cout<<"Answer"<<endl;
      cout<<angle_rad<<endl<<endl;
      ArmComm.Arm_Set_JointPosition(RobotisArm.Rad2Value(angle_rad +
                                                  gvdAngleGapCalcandDynamixelRad));
   }
   else if(ErrorStatus & ARMSDK_ACCEPTABLE_ERROR)
   {
      cout<< "No IK solution"<<endl;
      cout<< "But the calcuation result is acceptable"<<endl;
      char answer;
      while(true)
      {
         cout<< "Do you want make the Robot move? (Y/N)"

         cin >> answer;
         if((answer == 'y')    || (answer == 'n') || (answer == 'Y') || (answer == 'N'))
            break
         else
            cout<< "Invaild Answer"<<endl;
      }

      if((answer == 'y')    || (answer == 'Y') )
         ArmComm.Arm_Set_JointPosition(RobotisArm.Rad2Value(angle_rad +
                                                  gvdAngleGapCalcandDynamixelRad));
      else
         continue
   }
   else   {
      cout<< "No IK Solution"<<endl;
      continue
   }
   ArmKinematics.Forward(angle_rad, &CurrentPose);
}
```

The code shows that by pressing the q key the program runs. A press of q moves the end effector pose in the (X) coordinate by delta (5mm).

If there are no errors the end effector will move according to keystroke. All joints are in radians.

Press the 'q‘ key to to goal pose by X position in delta incrememts.

Despite having errors and not being able to get the IK moving can be allowed. If 'Do you want make the Robot move? (Y/N)' appears onscreen press the y key to move the end effector in the X coordinate by +5mm. Then the joints pose(rad) are displayed.

⚠ **Product may go to pose fast after pressing the Y key posing a safety risk. When testing the example keep a safe distance while able to cut power off in case of undesired operation**

When error is too large and IK is unrealizable 'No IK Solution‘ will be displayed the end effector will remain as is.

The sample code from above is broken down below.

```
ArmKinematics.ComputeIK(DesiredPose, &angle_rad, angle_rad, &ErrorStatus);
```

All joints set to a desired pose by taking input from DesiredPose and angle_rad. Once DesiredPose values go to CurrentPose then the arm moves in X coordinate and DesirePose gets set again. angle_rad is CurrentPose’s consistent joints angles. IK’s solution for desired pose joint angles and &angle_rad get set. &ErrorStatus is the error sent to Dynamixel.

```
ArmComm.Arm_Set_JointPosition(RobotisArm.Rad2Value(angle_rad +
                                   gvdAngleGapCalcandDynamixelRad));
```

The ComputeIK function sets an array for joint position in &angle_rad.

```
ArmKinematics.Forward(angle_rad, &CurrentPose);
```

Once moved to desired pose angle_rad(array) gets the end effector's pose and runs forward kinematics; then CurrentPose sets the pose. This function returns the end effectors transform matric (4x4).

```
else if(temp == 'r')
{
    DesiredPose = CurrentPose;
    matd DesiredRotation = Algebra::GetOrientationMatrix(delta_angle_rad, 0.0, 0.0) *
    Algebra::GetOrientationMatrix(CurrentPose.Roll, CurrentPose.Pitch, CurrentPose.Yaw);

    vecd DesiredRPY = Algebra::GetEulerRollPitchYaw(DesiredRotation);
    DesiredPose.Roll = DesiredRPY(0);
    DesiredPose.Pitch = DesiredRPY(1);
    DesiredPose.Yaw = DesiredRPY(2);
    ArmKinematics.ComputeIK(DesiredPose, &angle_rad, angle_rad, &ErrorStatus);

    if(ErrorStatus == ARMSDK_NO_ERROR)
    {
        cout<<"Answer"<<endl;
        cout<<angle_rad<<endl<<endl;
        ArmComm.Arm_Set_JointPosition(RobotisArm.Rad2Value(angle_rad +
                                                gvdAngleGapCalcandDynamixelRad));
    }
    else if(ErrorStatus & ARMSDK_ACCEPTABLE_ERROR)
    {
        cout<< "No IK solution"<<endl;
        cout<< "But the caluation result is acceptable"<<endl;
        char answer;
        while(true)
        {
            cout<< "Do you want make the Robot move? (Y/N)"

            cin >> answer;
            if((answer == 'y')   || (answer == 'n') || (answer == 'Y') || (answer == 'N'))
                break;
            else
                cout<< "Invaild Answer"<<endl;
    }
        If((answer == 'y')   || (answer == 'Y') )
        ArmComm.Arm_Set_JointPosition(RobotisArm.Rad2Value(angle_rad +
                                                gvdAngleGapCalcandDynamixelRad));
        else
            continue;
    }
    else   {
        cout<< "No IK Solution"<<endl;
        continue;
    }

    ArmKinematics.Forward(angle_rad, &CurrentPose);
}
```

The goal pose runs IK my moving the roll gets increased by delta(rad). The end effector moves to whatever the IK has solved and displays the joint poses(rad).

Despite having errors and not being able to get the IK moving can be allowed. If 'Do you want make the Robot move? (Y/N)' appears onscreen press the y key to turn the end effector in the roll axis by delta_angle_rad. Then the joints pose(rad) are displayed.

A roll (roll-only) delta is (delta_angle_rad = $3 \times \frac{\pi}{180} rad$)

When error is too large and IK is unrealizable 'No IK Solution‘ will be displayed the end effector will remain as is.

The sample code from above is broken down below. Press the r key to move the roll by delta_angle_rad.

The desired rotation matrix can then be obtain with the following

$$R_{desired} = R(\Delta\phi,\ 0.0,\ 0.0)^{*} R_{current}$$

Where the code is shown below.

```
matd DesiredRotation = Algebra::GetOrientationMatrix
                                                  (delta_angle_rad, 0.0, 0.0)
                        *Algebra::GetOrientationMatrix(CurrentPose.Roll,
                                                      CurrentPose.Pitch,
                                                      CurrentPose.Yaw);
```

The CurrentPose’s Orientation roll increase by delta_angle_rad GoalPose(DesiredRotation).

.

```
vecd DesiredRPY = Algebra::GetEulerRollPitchYaw(DesiredRotation);
```

DesiredRotation’s roll, pitch, and yaw..

3.4 SimpleTorqueOnOffandFK

Turns the manipulator joints' torque on/off. When torque goes off→on Forward Kinematics runs and putputs all joints pose(rad) and end effector's position and orientation.

ⅰ) How to Use SimpleTorqueOnOffandFK

To start SimpleTorqueOnOffandFK start a new project just like SimplePtoP. Then press the Ctrl+F5 keys to begin.SimpleTorqueOnOffandFK.

```
Input COM port number : 7
Input baud number : 1
Succeed to open USB2Dynamixel
```

Input the COM port and baud rate numbers. If succeeded you will see a 'Succeed to open USB2Dynamixel;' then torque gets turned off. Press the Enter key turn torque on and the arm's joints pose(rad) and end effector's pose(rad) will be displayed (joints 1 through 6).

```
JointAngle is
  -0.514395
  -0.412971
  -0.730969
 0.00553221
  -0.785279
-0.00219265

Angle of Dynamixel is
  -0.514395
    1.04467
   0.167581
 0.00553221
  -0.785279
-0.00219265

EndEffector's Pose is
x     = 427.133
y     = -241.391
z     = 27.1234
roll  = -0.00850204
pitch = -1.21237
yaw   = 2.6321
```

```
  -0.514395
  -0.412971
  -0.730969
 0.00553221
  -0.785279
-0.00219265

Angle of Dynamixel is
  -0.514395
    1.04467
   0.167581
 0.00553221
  -0.785279
-0.00219265

EndEffector's Pose is
x      = 427.133
y      = -241.391
z      = 27.1234
roll   = -0.00850204
pitch  = -1.21237
yaw    = 2.6321

Torque Off
```

Press Enter again to turn torque off and it will display 'Torque Off.'

Press the Enter key once again to turn torque on and the values be displayed again.

ⅱ) SimpleTorqueOnOffandFK Source Description

```
while(true) {
   char temp = _getch();
   if(temp == 27)
      break
   else if(temp == 13)   {
      if(gbArmTorque)     {
         ArmComm.Arm_Torque_Off();
         std::cout<<"Torque Off"<<std::endl;
         gbArmTorque = false
      }
      else     {
         ArmComm.Arm_Torque_On();
         cout<<"Torque On"<<endl;

         if(ArmComm.Arm_Get_JointPosition(&angle_unit) != COMM_RXSUCCESS)   {
            printf("Communication Error Occurred\n");
        }

         cout<<"JointAngle is"<<endl;
         angle_rad = RobotisArm.Value2Rad(angle_unit);
         cout<< angle_rad - gvdAngleGapCalcandDynamixelRad <<endl<<endl;
         cout<<"Angle of Dynamixel is"<<endl;
         angle_rad = RobotisArm.Value2Rad(angle_unit);
         cout<< angle_rad <<endl<<endl;
         cout<<"EndEffector's Pose is"<<endl;
         Pose3D CurrentPose;
         ArmKinematics.Forward(angle_rad
                                    - gvdAngleGapCalcandDynamixelRad, &CurrentPose);

         cout<<"x = "<< CurrentPose.x <<endl;
         cout<<"y = "<< CurrentPose.y <<endl;
         cout<<"z = "<< CurrentPose.z <<endl;
         cout<<"roll = "<< CurrentPose.Roll <<endl;
         cout<<"pitch = "<< CurrentPose.Pitch <<endl;
         cout<<"yaw = "<< CurrentPose.Yaw    <<endl<<endl;
         gbArmTorque = true
      }
   }
   else
      continue
}
```

The program aborts without starting by pressing the Esc key.

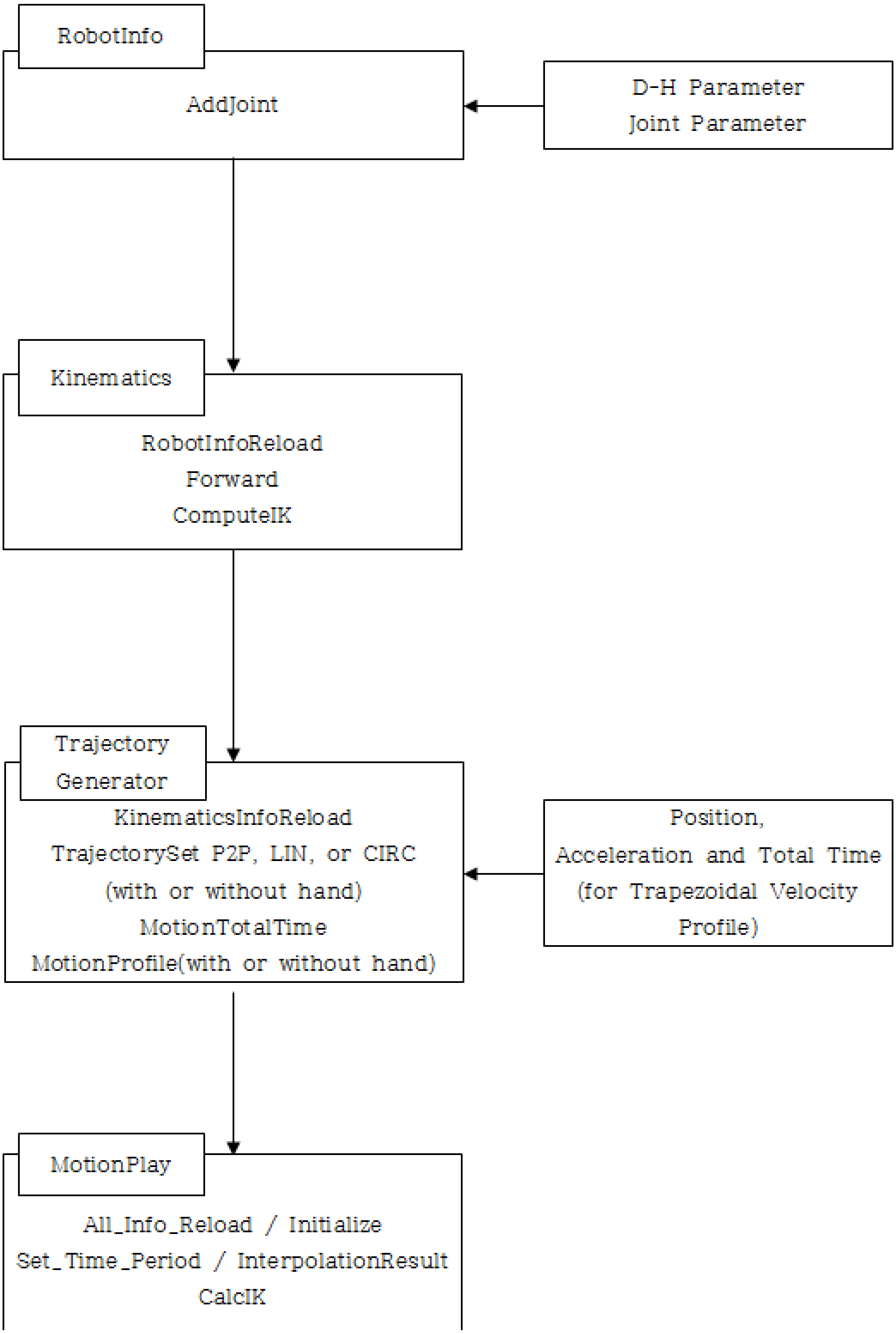
While the program is running press the Enter key to toggle torque between on and off, **When torque gets turned on the joints and end effector pose get outputted onscreen**. This happens with every “on” state.

Press the Esc key then Enter key and the arm remains as is.

The sample code from above is broken down below. The joint angles and Dynamixel angles may not be the same so it must be taken into consirderation. Angle of Dynamixel is the output of the actual angle of Dynamixel.

4. Robotis Manipulator SDK Programming

4.1 Robotis Manipulator SDK FlowChart



4.2 Description for Robotis Maniupulator SDK

ⅰ) RobotInfo

When making them anipulator's build RobotInfo class's Addjoint generated instance. AddJoint gets values from D-H Parameter and actuator's max and min turn angle in rad and value as well as actuator ID number (min and max turn angles may not be the same as joint angle limits).

ⅱ) Kinematics

Forward Kinematics(FK), Inverse Kinematics(IK) can be calculated once the instance for kinematics class is generated. Kinematics class get the instance from RobotInfo class.

ComputeIK's factor's the pose from end effector and joint values, as well as, initial joint angle and error for IK. The result is joint angles when the returned error status is not 0 then the IK is not properly solved.

```
armsdk::RobotInfo robot;
robot.AddJoint(  0.0, -ML_PI_2,    0.0,      0.0, ML_PI, -ML_PI, 251000, -251000, ML_PI, -ML_PI, 1);
robot.AddJoint(  0.0,  ML_PI_2,    0.0,      0.0, ML_PI, -ML_PI, 251000, -251000, ML_PI, -ML_PI, 2);
robot.AddJoint( 30.0,  ML_PI_2,  246.0,      0.0, ML_PI, -ML_PI, 251000, -251000, ML_PI, -ML_PI, 3);
robot.AddJoint(-30.0, -ML_PI_2,    0.0, -ML_PI_2, ML_PI, -ML_PI, 251000, -251000, ML_PI, -ML_PI, 4);
robot.AddJoint(  0.0,  ML_PI_2,  216.0,      0.0, ML_PI, -ML_PI, 151875, -151875, ML_PI, -ML_PI, 5);
robot.AddJoint(  0.0,      0.0,    0.0,      0.0, ML_PI, -ML_PI, 151875, -151875, ML_PI, -ML_PI, 6);

Kinematics kin(&robot);
vecd JointAngle = *kin.GetCurrentAngle();

armsdk::Pose3D DesiredPose;
DesiredPose.x     = 200.0;
DesiredPose.y     = 0.0;
DesiredPose.z     = 400.0;
DesiredPose.Roll  = ML_PI/2;
DesiredPose.Pitch = -ML_PI/2;
DesiredPose.Yaw   = 0.0;

int Error;
kin.ComputeIK(DesiredPose, &JointAngle, JointAngle, &Error);
```

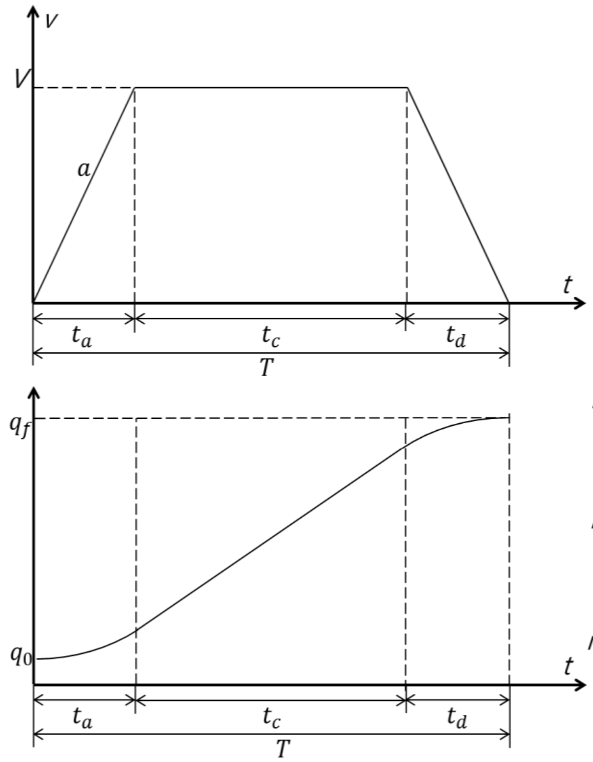
The $Roll(\phi)$, $Pitch(\theta)$, and $Yaw(\psi)$ are calculated as $R_z(\psi)\ R_y(\theta)\ R_x(\phi)$ in the rotation transformation matrix. This is to be taken into consideration when entering the pose for ComputeIK.

ⅲ) Trajectory Generating

The TrajectoryGenerator class generates an instance for the arm’s trajectory. The SDK’s Point to Point, Linear, and Circular can generate a trajectory. For arm-only trajectory then only Set_PTP, Set_LIN, Set_Circular; for the gripper then Set_PTPwithHand, Set_LINwithHand, Set_CIRCwithHand.

ⅳ) Velocity Profile

The SDK’s Velocity Profile does not take max velocity and max acceleration into consideration in the Trapezoidal Velocity Profile. The initial and final velocity are always set to 0. The following methods generate velocity profile in Joint Space and Cartesian Space, where both are independent of each other.



$t_a = t_d$

$t_a + t_c + t_d = T$

$\therefore\ V = \frac{q_f}{T - t_a}\ ,\quad a = \frac{V}{t_a}$

i) $0 \le t < t_a$

$$q(t) = q_0 + \frac{1}{2}at^2$$

ii) $t_a \le t < t_a + t_c$

$$q(t) = -\frac{1}{2}a{t_a}^2 + Vt$$

ii) $T - t_a \le t \le T$

$$q(t) = q_f - \frac{1}{2}aT^2 + aTt - \frac{1}{2}at^2$$

ⅴ) Set_PTP

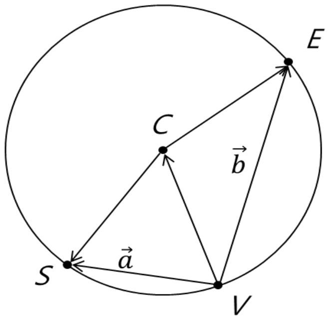
The Set_PTP function determines 2 poses for the manipulator (initial and final) by factoring in Trapezoidal Velocity Profile and receives velocity time and total time. Initial and final pose are in rad and joint angle in mm or rad (x, y, z, roll, pitch, yaw). When generating the trajectory it is recommended to factor in joint angles.

ⅵ) Set_LIN

The Set_LIN function generates a 3-point coordinates for the robot's straight trajectory. This factors in initial and final pose for Linear Euler Interpolation for orientation.

vii) Set_CIRC

The Set_CIRC function generates a 3-point coordinates for the robot's circular trajectory. This factors in initial and final pose. It sets a point of origin in the area and proceeds to trajectory via MotionPlay and vector generation.



$$\vec{VC} = p\vec{a} + q\vec{b}$$

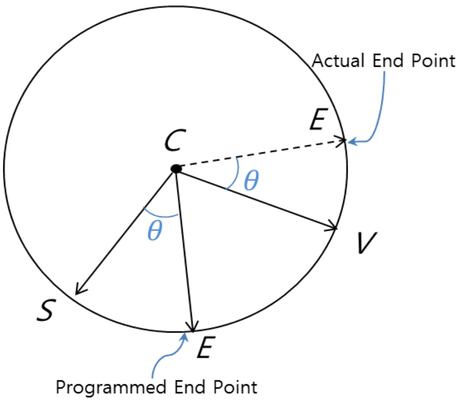
$$|\vec{VC}|^2 = |\vec{VC} - \vec{a}|^2$$

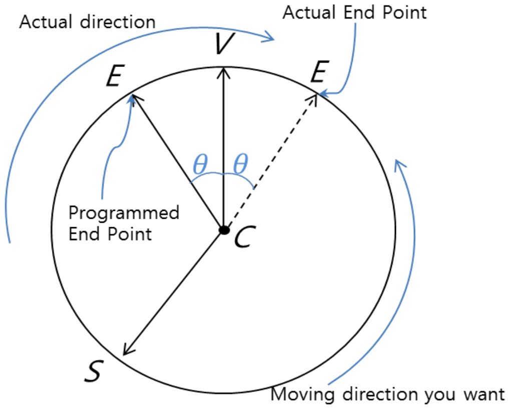
$$|\vec{a}|^2 = 2(p|\vec{a}|^2 + q(\vec{a}\cdot\vec{b}))$$

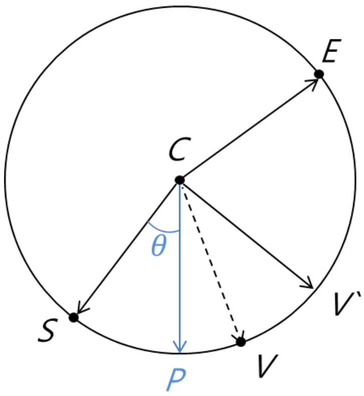
$$|\vec{VC}|^2 = |\vec{VC} - \vec{b}|^2$$

$$|\vec{b}|^2 = 2\left(p(\vec{a}\cdot\vec{b}) + q|\vec{b}|^2\right)$$

$$\therefore\ p = \frac{|\vec{b}|^2(|\vec{a}|^2 - (\vec{a}\cdot\vec{b}))}{2\left((|\vec{a}||\vec{b}|)^2 - (\vec{a}\cdot\vec{b})^2\right)} \qquad \therefore\ q = \frac{|\vec{a}|^2\left((\vec{a}\cdot\vec{b}) - |\vec{b}|^2\right)}{2\left((\vec{a}\cdot\vec{b})^2 - (|\vec{a}||\vec{b}|)^2\right)}$$







$$\vec{CV`} = (\vec{CV} - (\vec{a}\cdot\vec{CV})\vec{a})$$

$$\vec{CV`} = \frac{|\vec{CS}|}{|\vec{CV`}|} \times \vec{CV`}$$

$$\therefore\ \vec{CP} = \cos\theta\,\vec{CS} + \sin\theta\,\vec{CV`}$$

ⅷ) Trajectory Following

When moving by the generated trajectory from TrajectorGenerator class's instance just use NextStep function from MotionPlay. MotionPlay class accounts trajectoryGenerator class.

The control period from MotionPlay default value is 8ms but can be changed with SetTimePeriod. If TimePeriod is 0 then 8ms default value is applied.

ⅸ) Pro_Arm_Comm_Win

Pro_Arm_Comm_Win utilizes DYNAMIXEL 2.0 Protocol from the Windows version of DYNAMIXEL SDK. Pro_Arm_Comm_Win's functions utilizes DYNAMIXEL Pro's control (i.e. read/write Control Table values). This is useful when writing separate code.

5. Maintenance
5.1 Restore Firmware

<u>When Dynamixel detection fails ensure is properly wired.</u>

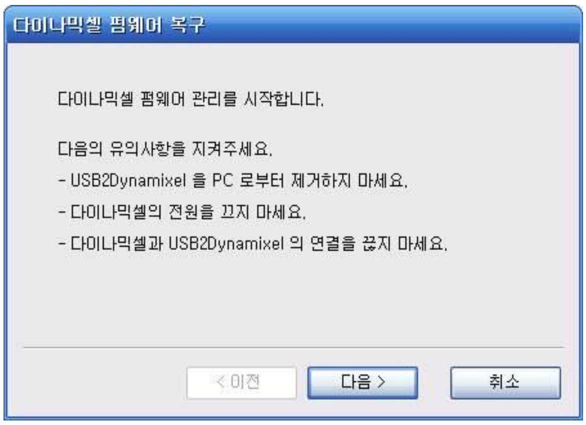
If problems perists **<u>restore Dynamixel firmware</u>** (shown below).

> ⚠ **After firmware restoration you will need to set ID and baud rate values again. Always make sure to set USB2Dynamixel switch to "485."**

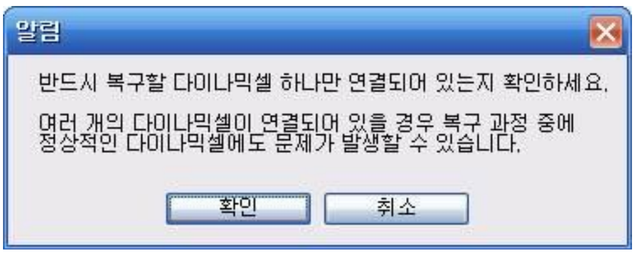
ⅰ) restoring firmware

- From Dynamixel Wizard click on the  icon to begin.
- select the corresponding COM port number for USB2Dynamixel.

ⅱ) Firmware restore process steps explained.

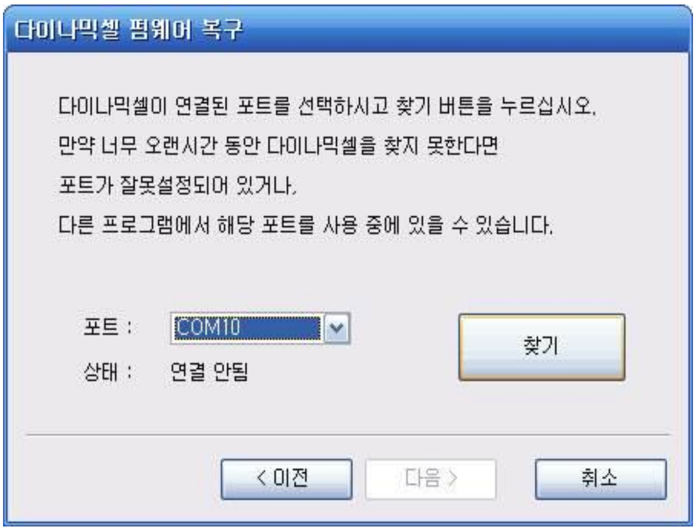


ⅲ) always connect one Dynamixel at a time.



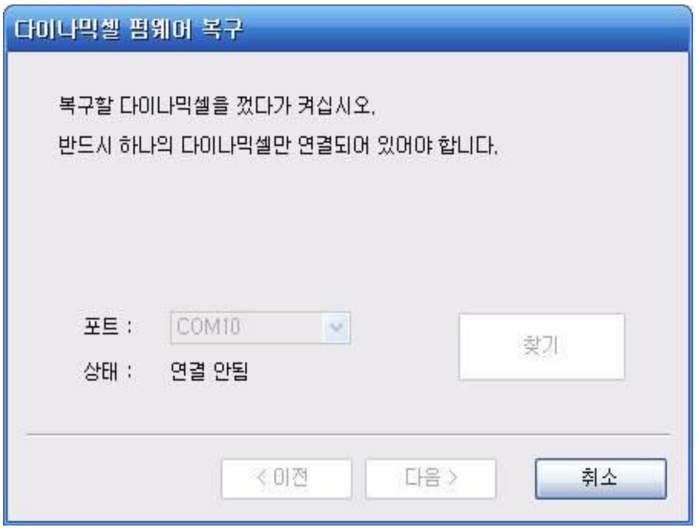
ⅳ) pick the COM port number

- with an incorrect number Dynamixel cannot be automatically detected. Always make sure to get the port number right.
- click on Search.

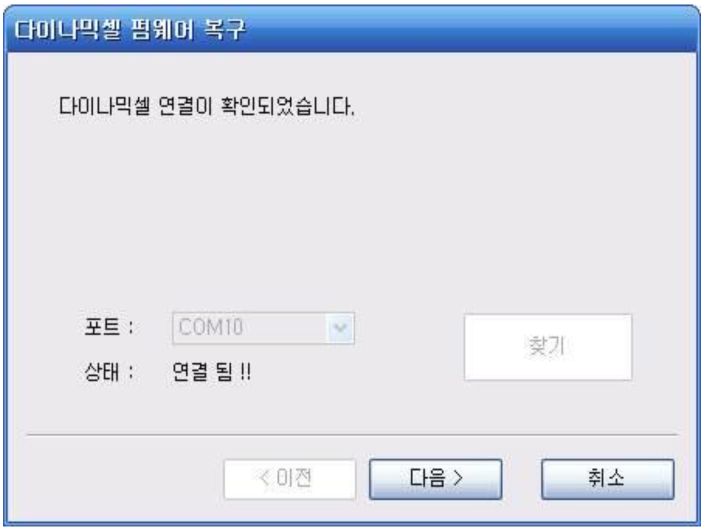


ⅴ) Disconnect and connect Dynamixel

- The Next button should become clickable

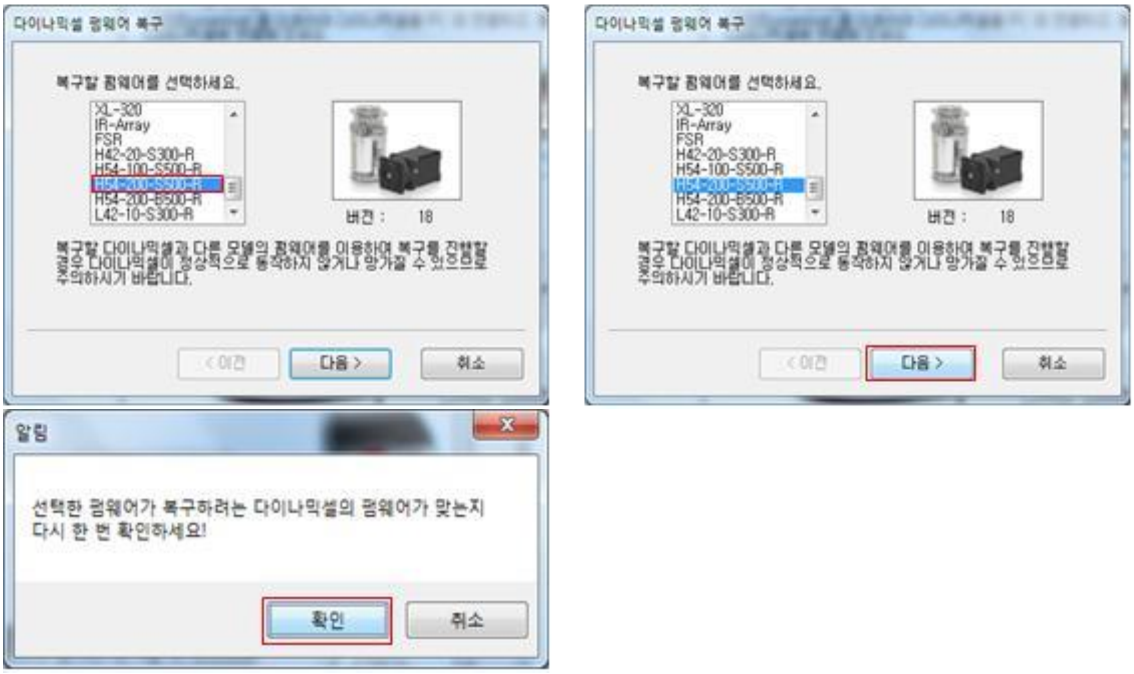


ⅵ) **Upon successful detection the Next button is clickable**



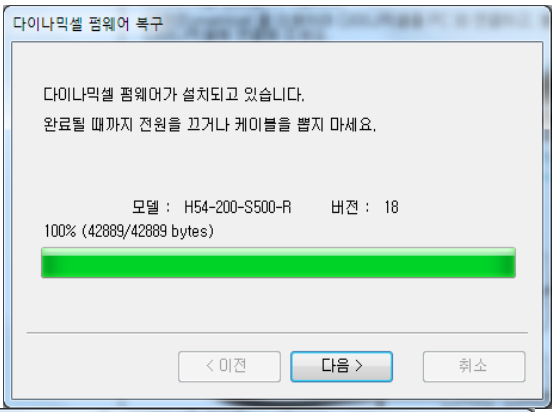
ⅶ) **Pick the right model**

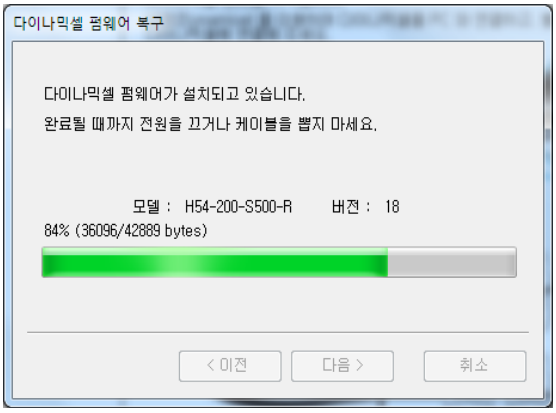
- pick the right type from the list. If not it may result in problems

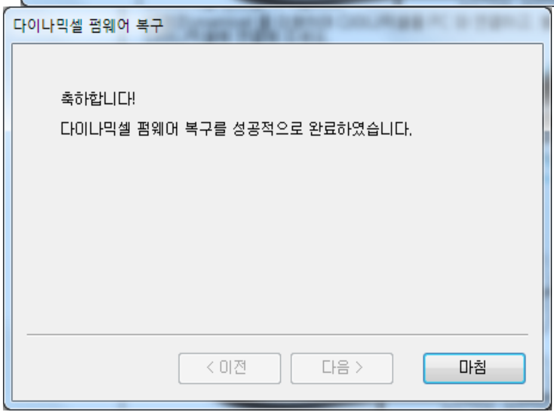


ⅷ) **during restoration**

- while restoring the LED will blink. Do not cut power off during this stage.







<u>All Control Table settings are set to default values.</u>

6. Reference

6.1 ARMSDK_Definitions

(1) Pose3D

Data Fields	double x, y, z double Roll, Pitch, Yaw
Description	Position(x,y,z) and Orientation(Roll, Pitch, Yaw) elements

(2) timeprofile

Data Fields	double ta, tc, td, totaltime double a0[3], a1[3], a2[3] double distance, distance1 int Method
Description	Trapezoidal Velocity Profile's elements distance1 only used in circular trajectory.

(3) MotionPose

Data Fields	vecd StartPose, EndPose Pose3D StartPose3D, ViaPose3D, EndPose3D Position3D CenterPosition int Method
Description	Declaration of manipulator step's StartPose, EndPose and trajectory method ViaPose and CenterPosition for circular trajectory

6.2 ARMSDK_Math.h

(1) static matd GetOrientationMatrix(double Roll, double Pitch, double Yaw)

Parameter	double Roll, double Pitch, double Yaw
Returns	3 x 3 Rotation Matrix
Description	orientation(Roll, Pitch, Yaw) input 3 x 3 orientation matrix output

(2) static matd GetTransformMatrix(double Roll, double Pitch, double Yaw,
double x, double y, double z)

Parameter	double Roll, double Pitch, double Yaw double x, double y, double z
Returns	4 x 4 Transformation Matrix
Description	orientation(Roll, Pitch, Yaw)와 Position(X, Y, Z) input 4 x 4 transform Matrix output

(3) static vecd rot2omega(mat3d Rerr)

Parameter	Rotation Matrix
Returns	angular velocity array
Description	rotation matrix gets input and outputs velocity array

(4) static vecd ConvertRad2Deg(vecd q)

Parameter	radian Array
Returns	Degree Array
Description	(rad) gets input, change to (value) and return

(5) static vecd GetEulerRollPitchYaw(matd T)

Parameter	3 x 3 rotation Matrix or 4 x 4 Transformation Matrix
Returns	3 x 1 array (Roll, Pitch, Yaw)^T

Description	vecd rpy(3);
	rpy(0) = atan2(T(2,1), T(2,2));
	rpy(1) = atan2(-T(2,0), sqrt(T(2,1)*T(2,1) + T(2,2)*T(2,2)));
	rpy(2) = atan2(T(1,0), T(0,0));

6.3 MotionEngine

ⅰ) Error.h

(1) void ErrorCheck(int Error)

Parameter	int Error
Return	void
Description	1. no error (ARMSDK_NO_ERROR 0x00) 2. IK solution does not exist (ARMSDK_NO_IK_SOLUTION 0x01) 3. no IK solution and allowable error (ARMSDK_ACCEPTABLE_ERROR 0x02) 4. Joints' next and previous step large difference in angle (ARMSDK_TOO_MUCH_ANGLE_CHANGE 0x04) 5. angle or not within JointData's limit (ARMSDK_OUT_OF_JOINT_RANGE 0x08) The 5 types of ERROR

ⅱ) JointData.h

(1) void SetJointID(unsigned int ID)

Parameter	unsigned int ID
Return	void
Description	Assign Joint ID

(2) void SetJointAngle(double JointAngle);

Parameter	double JointAngle
Return	void
Description	Set Joint Angle

(3) void SetMinAngleInRad(double MinAngleInRad);

Parameter	double MinAngleInRad
Return	void
Description	Set actuator min angle(rad) Value utilized in 6.3 MotionEngine - iii) RobotInfo's rad2value function

(4) void SetMaxAngleInRad(double MaxAngleInRad);

Parameter	double MaxAngleInRad
Return	Void
Description	Set actuator max angle(rad) Value utilized in 6.3 MotionEngine - iii) RobotInfo's rad2value function

(5) void SetMinAngleInValue(int Min_AngleValue);

Parameter	int Min_AngleValue
Return	void
Description	Set actuator min value utilized in 6.3 MotionEngine - iii) RobotInfo's rad2value, value2rad functions

(6) void SetMaxAngleInValue(int Max_AngleValue);

Parameter	int Max_AngleValue
Return	void
Description	Set actuator max value Utilized in 6.3 MotionEngine - iii) RobotInfo's rad2value, value2rad functions

(7) void SetMinAngleLimitInRad(double MinAngleLimitInRad);

Parameter	double MinAngleLimitInRad
Return	void
Description	Set joint min angle(rad) Also sets the value

(8) void SetMaxAngleLimitInRad(double MaxAngleLimitInRad);

Parameter	double MaxAngleLimitInRad
Return	void
Description	Set joint max angle(rad) Also sets the value

(9) unsigned int GetID(void);

Parameter	void
Return	unsigned int (ID)
Description	Returns joint ID (number)

(10) void SetJointDataDH(double LinkLength, double LinkTwist,
double JointOffset, double JointAngle);

Parameter	double LinkLength, double LinkTwist double JointOffset, double JointAngle
Return	void
Description	Set manipulator's joint DH parameters in DH Configuration

(11) double GetJointAngle(void);

Parameter	void
Return	double current Angle
Description	Returns joint angle limit(rad)

(12) double GetMinAngleInRad(void);

Parameter	void
Return	MinAngle(rad) of Actuator
Description	SetMinAngleInRad returns actuator min angle(rad)

(13) double GetMaxAngleInRad(void);

Parameter	void
Return	MaxAngle(rad) of Actuator
Description	SetMaxAngleInRad returns actuator max angle(rad)

(14) int GetMinAngleInValue(void);

Parameter	void
Return	MinAngle(value) of Actuator
Description	SetMinAngleInValue returns actuator min angle(value)

(15) int GetMaxAngleInValue(void);

Parameter	void
Return	MaxAngle(value) of Actuator
Description	SetMaxAngleInValue returns actuator max angle(value)

(16) double GetMinAngleLimitInRad(void);

Parameter	Void
Return	MinAngle(rad) of Joint
Description	SetMinAngleLimitInRad returns joint min angle(rad)

(17) double GetMaxAngleLimitInRad(void);

Parameter	void
Return	MaxAngle(rad) of Joint
Description	SetMaxAngleLimitInRad returns joint max angle(rad)

(18) int GetMinAngleLimitInValue(void);

Parameter	void
Return	MinAngle(value) of Joint
Description	SetMinAngleLimitInRad returns joint min angle(value)

(19) int GetMaxAngleLimitInValue(void);

Parameter	void
Return	MaxAngle(value) of Joint
Description	SetMaxAngleLimitInRad returns joint max angle(value)

(20) matd GetTransformMatirx(void);

Parameter	void
Return	matd TransformMatrix of each Link
Description	returns transform matrix for each link

iii) RobotInfo.h

(1) int AddJoint (double LinkLength, double LinkTwist, double JointOffset,
double JointAngle, double MaxAngleInRad, double MinAngleInRad,
int MaxAngleValue , int MinAngleValue,
double MaxAngleLimitInRad, double MinAngleLimitInRad,
unsigned int Dynamixel_ID);

Parameter	LinkLength, LinkTwist, JointOffset, JointAngle – DH parameter MaxAngleInRad - Maximum Angle of Actuator(not Joint Limit) MinAngleInRad – Minimum Angle of Actuator(not Joint Limit) MaxAngleInValue – AngleValue corresponding to the Maxangle MinAngleInValue – AngleValie corresponding to the Minangle MaxAngleLimitInRad – Maximum Joint Angle Limit of Actuator MinAngleLimitInRad – Minimum Joint Angle Limit of Actuator
Return	Error Value
Description	Sets joint's DH-Parameter and Joint-Parameter values Error of 0 is no error and 1 when there is error. Error happens when min value is greater than max value

(2) JointData GetJointInfo(int joint_number);

Parameter	int Joint_number
Return	JointData
Description	Returns JointData from AddJoint

(3) std::vector<JointData>* GetRobotInfo(void);

Parameter	JointData
Return	address of robotInfo
Description	Returns address values from RobotInfomation

(4) void ClearRobotInfo(void);

Parameter	void
Return	void
Description	Clears out RobotInfo

(5) void GetArmIDList(void);

Parameter	void
Return	ID List of Robot Actuators
Description	Returns joint ID in aray form inAddJoint

(6) veci Rad2Value(vecd q);

Parameter	double array of Actuators Angle(Rad)
Return	int array of Actuators Angle(Value)
Description	Transforms joint's rad to value.

(7) vecd Value2Rad(veci q);

Parameter	int array of Actuators Angle(Value)
Return	double array of Actuators Angle(Rad)
Description	Transforms joint's value to rad.

ⅳ) Kinematics.h

(1) void RobotInfoReload(void);

Parameter	void
Return	void
Description	Calls RobotInfo

(2) matd Forward(vecd angle);

Parameter	Angle of All Joints(rad)
Return	4x4 TransformMatrix form
Description	RobotInfoReload calls joints angles runs FK and returns end effector's transformation matrix

(3) matd Forward(vecd angle, Pose3D *pose);

Parameter	Angle of All Joints(rad)
Return	4x4 EndEffector's TransformMatrix form
Description	RobotInfoReload calls joint angles runs FK and returns end effector's transformation matrix. It also sets pose pointer (*pose)

(4) void SetMaximumNumberOfIterationsForIK(unsigned int max_num);

Parameter	unsigned int max_num for IK
Return	void
Description	Sets IK's number of iterations for solution

(5) void SetConvergenceCondition(double max_error, double max_acceptable_error);

Parameter	double max_error, double max_acceptable_error
Return	void
Description	IK's amount of telorable error. The first input value is max convergence error. A lesser value than max can allow solution. The second value is max allowable error; acceptable as long as is lower than value entered. When value exceeds then there is no solution..

(6) matd Jacobian(void);

Parameter	void
Return	Matrix of Jacobian
Description	Returns jacobian for IK solution

(7) vecd CalcError(Pose3D _desired, matd _current);

Parameter	Pose3D goalPose, TransformMatrix of EndEffector
Return	Error between Goal and Currnet Pose
Description	Compares end effector's goal pose and current pose

(8) void ComputeIK(Pose3D _desired , vecd *q_rad,
vecd Initangle_rad, int *ErrorStatus);

Parameter	Pose3D goalPose, vecd initangle, int ErrorStatus
Return	void
Description	get jacobian's Damped Least Square Method for IK solution _desired is end effector's desired pose *q_rad sets joints pose after running IK Initangle_rad is joint angles prior to running IK *ErrorStatus is pointer for error type ErrorStatus. 1. no error (ARMSDK_NO_ERROR 0x00) 2. No solution from IK (ARMSDK_NO_IK_SOLUTION 0x01) 3. no solution from IK, allowable error (ARMSDK_ACCEPTABLE_ERROR 0x02) 4. joint angles exceed JointData's set angles (ARMSDK_OUT_OFF_JOINT_RANGE 0x08)

ⅴ) TrajectoryGenerator.h

(1) void KinematicsInfoReload(void);

Parameter	void
Return	void
Description	Calls Kinematics info

(2) void Set_P2P(vecd StartPose, vecd EndPose,

double TotalTime, double AccelTime)

Parameter	vecd StartPose / vecd EndPose double TotalTime / double AccelTime
Returns	void
Description	sets P2P trajectory fromStartPose, EndPose, TotalTime, AccelTime

(3) void Set_P2P(Pose3D StartPose, Pose3D EndPose,

double TotalTime, double AccelTime)

Parameter	Pose3D StartPose / Pose3D EndPose double TotalTime / double AccelTime
Returns	Void
Description	Sets P2P trajectory from StartPose, EndPose, TotalTime, AccelTime

(4) void Set_LIN(vecd StartPose, vecd EndPose,

double TotalTime, double AccelTime)

Parameter	vecd StartPose / vecd EndPose double TotalTime / double AccelTime
Returns	void
Description	Sets linear trajectory from StartPose, EndPose, TotalTime, AccelTime

(5) void Set_LIN(Pose3D StartPose, Pose3D EndPose,
double TotalTime, double AccelTime)

Parameter	Pose3D StartPose / Pose3D EndPose double TotalTime / double AccelTime
Returns	void
Description	Sets linear trajectory from StartPose, EndPose, TotalTime, AccelTime.

(6) void Set_CIRC(vecd StartPose, vecd ViaPose, vecd EndPose,
double TotalTime, double AccelTime)

Parameter	vecd StartPose / vecd ViaPose / vecd EndPose double TotalTime / double AccelTime
Returns	void
Description	Sets circular trajectory from StartPose, ViaPose, EndPose, TotalTime, AccelTime.

(7) void Set_CIRC(Pose3D StartPose, Pose3D ViaPose, Pose3D EndPose,
double TotalTime, double AccelTime)

Parameter	Pose3D StartPose / Pose3D EndPose double TotalTime / double AccelTime
Returns	void
Description	Sets circular trajectory fom StartPose, ViaPose, EndPose, TotalTime, AccelTime

(8) void Set_P2PwithHand(vecd StartPose, vecd EndPose,
veci Hand1, veci Hand2,
double TotalTime, double AccelTime)

Parameter	vecd StartPose / vecd EndPose veci Hand1 / veci Hand2 double TotalTime / double AccelTime
Returns	void
Description	Sets P2P trajectory from StartPose, EndPose, Start HandPose, End HandPose, TotalTime, AccelTime

(9) void Set_P2PwithHand(Pose3D StartPose, Pose3D EndPose,
veci Hand1, veci Hand2,
double TotalTime, double AccelTime)

Parameter	Pose3D StartPose / Pose3D EndPose veci Hand1 / veci Hand2 double TotalTime / double AccelTime
Returns	void
Description	Sets P2P trajectory from StartPose, EndPose, Start HandPose, End HandPose, TotalTime, AccelTime

(10) void Set_LINwithHand(vecd StartPose, vecd EndPose,
veci Hand1, veci Hand2,
double TotalTime, double AccelTime)

Parameter	vecd StartPose / vecd EndPose veci Hand1 / veci Hand2 double TotalTime / double AccelTime
Returns	void
Description	Sets linear trajectory from StartPose, EndPose, Start HandPose, End HandPose, TotalTime, AccelTime.

(11) void Set_LINwithHand(Pose3D StartPose, Pose3D EndPose,
veci Hand1, veci Hand2,
double TotalTime, double AccelTime)

Parameter	Pose3D StartPose / Pose3D EndPose veci Hand1 / veci Hand2 double TotalTime / double AccelTime
Returns	void
Description	Sets linear trajectory from StartPose, EndPose, Start HandPose, End HandPose, TotalTime, AccelTime

(12) void Set_CIRCwithHand(vecd StartPose, vecd ViaPose, vecd EndPose,
veci Hand1, veci Hand2,
double TotalTime, double AccelTime)

Parameter	vecd StartPose / vecd EndPose / vecd ViaPose veci Hand1 / veci Hand2 double TotalTime / double AccelTime
Returns	void
Description	Sets circular trajectory from StartPose, ViaPose, EndPose, Start HandPose, End HandPose, TotalTime, AccelTime

(13) void Set_CIRCwithHand(Pose3D StartPose, Pose3D ViaPose,
Pose3D EndPose,
veci Hand1, veci Hand2,
double TotalTime, double AccelTime)

Parameter	Pose3D StartPose / Pose3D ViaPose / Pose3D EndPose veci Hand1 / veci Hand2 double TotalTime / double AccelTime
Returns	void
Description	Sets circular trajectory from StartPose, ViaPose, EndPose, Start HandPose, End HandPose, TotalTime, AccelTime

(14) void ClearMF(void)

Parameter	void
Returns	void
Description	Clears out motion profile

(15) double GetMotionTotalTime(void)

Parameter	void
Returns	TotalTime in sec
Description	Returns motion run time

ⅵ) MotionPlay.h

(1) void All_Info_Reload(void);

Parameter	void
Returns	void
Description	Calls motion’s Info(RobotInfo, Kinematics, Trajectory)

(2) void initialize(void)

Parameter	void
Returns	void
Description	Initializes motion profile, done time, step, current time

(3) void Set_Time_Period(int MilliSecond)

Parameter	int MilliSecond
Returns	void
Description	Provides period time for motion

(4) vecd NextStepAtTime(double CurrentTime, int *ErrorStatus)

Parameter	double CurrentTime int *ErrorStatus
Returns	Joint Angle of next Step
Description	Returns next Goal Joint Angle array *ErrorStatus is pointer for error type

(5) veci NextStepAtTimeforHand(double CurrentTime)

Parameter	CurrentTime - current time in sec
Returns	Angle Value array of Fingers for next step
Description	Returns following Goal Joint Angle array for hand Assumes hand is attached to arm

(6) vecd CalcIK(Pose3D desiredPose, int *ErrorStatus)

Parameter	Pose3D desiredPose / int * ErrorStatus
Returns	Joint Angle of desiredPose
Description	Returns desired pose of end effector via IK *ErrorStatus is pointer for error type 1. no error (ARMSDK_NO_ERROR 0x00) 2. No solution from IK (ARMSDK_NO_IK_SOLUTION 0x01) 3. no solution from IK, allowable error (ARMSDK_ACCEPTABLE_ERROR 0x02) 4. joint angles exceed JointData's set angles (ARMSDK_OUT_OFF_JOINT_RANGE 0x08)

(7) vecd NextStep(int* ErrorStatus)

Parameter	ErrorStatus
Returns	Angle rad array for next step
Description	Returns next motion's joint angles *ErrorStatus is pointer for error type

(8) veci NextStepforHand(void)

Parameter	void
Returns	Angle Value array of Fingers for next step
Description	Returns hand's next motion position

(9) vecd GetCurrentAngle(void);

Parameter	void
Returns	All Joint Angle(rad)
Description	Returns robot's current pos(rad) array

(10) Pose3D GetCurrentEndPose(void);

Parameter	void
Returns	Pose3D of EndEffector
Description	Returns end effector's current pose

(11) double Get_CurrentTime(void);

Parameter	void
Returns	double CurrentTime
Description	Returns current time

6.4 RobotisLib

ⅰ) Pro_Arm_Comm_win.h

(1) void DXL_Set_Init_Param(int portnum, int baudnum);

Parameter	int portnum, int baudnum
Returns	void
Description	Sets Dynamixel comms from portnum and baudnum

(2) int DXL_Open();

Parameter	void
Returns	void
Description	Opens/access comms to DYNAMIXEL_Set_Init_Param

(3) SerialPort* DXL_Get_Port(void);

Parameter	void
Returns	PortNumber
Description	Returns SerialPort pointer address

(4) void DXL_Close(void);

Parameter	void
Returns	void
Description	End communications with Dynamixel

(5) void Arm_ID_Setup(veci Arm_ID_LIST);

Parameter	array of ID List
Returns	void
Description	Sets arm's ID list

(6) int Arm_Torque_On(void);

Parameter	void
Returns	Communication Result
Description	Turn torque on every joint COMM_RXSUCCESS return of 1

(7) int Arm_Torque_Off(void);

Parameter	void
Returns	Communication Result
Description	Turns torque off on every joint COMM_RXSUCCESS return of 1

(8) int Arm_Set_JointPosition(veci position);

Parameter	joint angle array
Returns	Communication Result
Description	Sets joint angles COMM_RXSUCCESS return of 1

(9) int Arm_Set_JointVelocity(veci velocity);

Parameter	int joint velocity array
Returns	Communication Result
Description	Sets joint valocities COMM_RXSUCCESS return of 1

(10) int Arm_Set_JointVelocity(int velocity);

Parameter	int joint velocity
Returns	Communication Result
Description	Sets joint velocities COMM_RXSUCCESS return of 1

(11) int Arm_Set_JointAcceleration(veci accel);

Parameter	int joint Acceleration array
Returns	Communication Result
Description	Sets joint accelerations COMM_RXSUCCESS return of 1

(12) int Arm_Set_JointAcceleration(int accel);

Parameter	int joint Acceleration
Returns	Communication Result
Description	Sets joint accelerations COMM_RXSUCCESS return of 1

(13) int Arm_Set_Position_PID_Gain(int P_Gain, int I_Gain, int D_Gain);

Parameter	int joint Position P, I, D gain
Returns	Communication Result
Description	Sets joints' PID gain values COMM_RXSUCCESS return of 1

(14) int Arm_Set_Position_PID_Gain(int id, int P_Gain, int I_Gain, int D_Gain, int* ErrorStatus);

Parameter	int id, int joint Position P, I, D gain
Returns	Communication Result
Description	Sets joints' PID gain values *ErrorStatus is error pointer COMM_RXSUCCESS return of 1

(15) int Arm_Get_JointPosition(veci *position);

Parameter	joint position array
Returns	Communication Result
Description	Access position array and gets joint positions COMM_RXSUCCESS return of 1

(16) int Arm_Get_JointCurrent(veci *torque);

Parameter	joint current array
Returns	Communication Result
Description	Reads joint's electrical current flow and saves in (*torque) return pointer COMM_RXSUCCESS return of 1

(17) int Arm_LED_On(void);

Parameter	void
Returns	Communication Result
Description	Turns joints' LED on COMM_RXSUCCESS return of 1

(18) int Arm_LED_Off(void);

Parameter	void
Returns	Communication Result
Description	Turns joints' LED off COMM_RXSUCCESS return of 1

(19) int Arm_LED_On(int r, int g, int b);

Parameter	int r, int g, int b
Returns	Communication Result
Description	Controls DYNAMIXEL Pro's RGB LED r, g, b, rage is 0~255 each COMM_RXSUCCESS return of 1

(20) int Arm_Red_LED_On(void);

(21) int Arm_Green_LED_On(void);

(22) int Arm_Blue_LED_On(void);

Parameter	void
Returns	Communication Result
Description	(20) turns joints' LED on to red (21) turns joints' LED on to green (22) turns joints' LED on to blue COMM_RXSUCCESS return of 1

(23) void Gripper_ID_Setup(veci Gripper_ID_List);

Parameter	ID array
Returns	void
Description	Sets ID for gripper.

(24) int Gripper_Ping(void);

Parameter	Void
Returns	Communication Result
Description	Pings comm to gripperGripper COMM_RXSUCCESS return of 1

(25) int Gripper_Torque_On(void);

Parameter	Void
Returns	Communication Result
Description	Turns gripper torque on COMM_RXSUCCESS return of 1

(26) int Gripper_Torque_Off(void);

Parameter	void
Returns	Communication Result
Description	Turns gripper torque off COMM_RXSUCCESS return of 1

(27) int Gripper_Get_Joint_Value(veci *value);

Parameter	Joint value array stored in address
Returns	Communication Result
Description	Access gripper's angles from stored address COMM_RXSUCCESS return of 1

(28) int Gripper_Set_Joint_Value(veci value);

Parameter	Joint value array
Returns	Communication Result
Description	Sets gripper joint value COMM_RXSUCCESS return of 1

6.5 Timer

ⅰ) **MotionTimer.h**

time measurement fromQueryPerformanceCounter

(1) void Start(void)

Parameter	void
Returns	void
Description	Sets start time

(2) void Stop(void)

Parameter	void
Returns	void
Description	Stops time measurement

(3) double GetElapsedTime(void)

Parameter	void
Returns	ElapsedTime in milliseconds
Description	Returns time from start to stop

(4) void Wait(double millisec)

Parameter	millisecond - waiting time in milliseconds you want
Returns	void
Description	Waits amount of time(msec) for standby