ME 5250 - Tutorial 3

This tutorial will guide you through the basics of using MATLAB to control a PincherX 100 robotic arm. In particular, you will learn how to communicate with the robot hardware to read joint positions/velocities and send commands to the robot. Follow the steps below carefully to complete the introductory lab session.

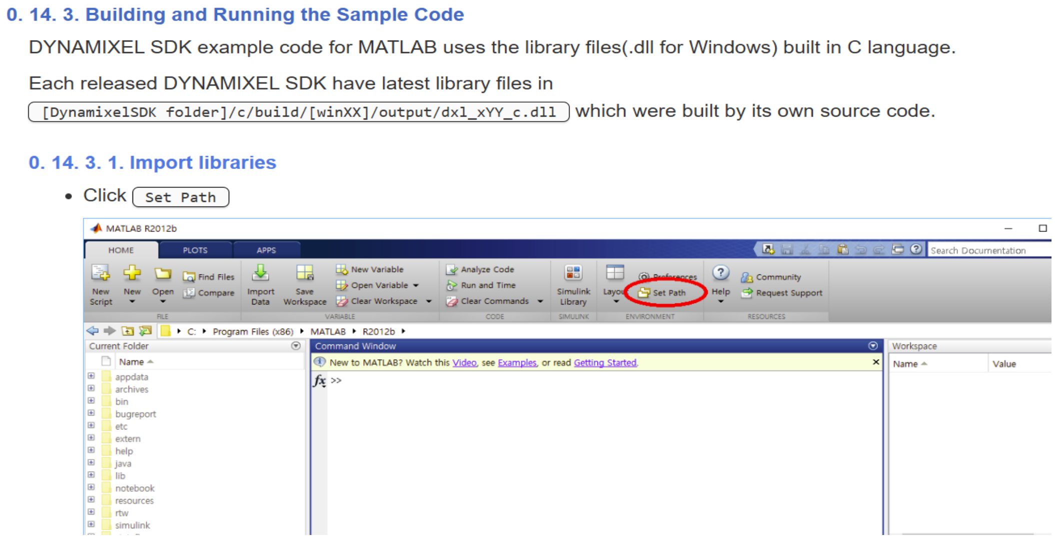
Requirements

Make sure the following software components and libraries are installed to ensure smooth operation:

1. Install MATLAB R2023A (or the latest version).
   MATLAB provides an interactive environment for algorithm development and robot control.
2. Install Robotics Systems Toolbox in MATLAB.
   - Go to Add-Ons > Get Add-Ons and search for 'Robotics Systems Toolbox.'

1. Download Dynamixel sdk library using the link below.
   https://github.com/ROBOTIS-GIT/DynamixelSDK/archive/3.7.31.zip
2. Build the Dynamixel_sdk library in MATLAB (Windows) using the documentation provided in this website.
   https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/library_setup/matlab_windows/#matlab-windows

Note: DYNAMIXEL SDK code for MATLAB uses the library files (.dll for Windows) built in C language. Please Build the library for C (Windows) and then do it for MATLAB (Windows)



0. 14. 3. Building and Running the Sample Code

DYNAMIXEL SDK example code for MATLAB uses the library files(.dll for Windows) built in C language.

Each released DYNAMIXEL SDK have latest library files in `[DynamixelSDK folder]/c/build/[winXX]/output/dxl_xYY_c.dll` which were built by its own source code.

0. 14. 3. 1. Import libraries

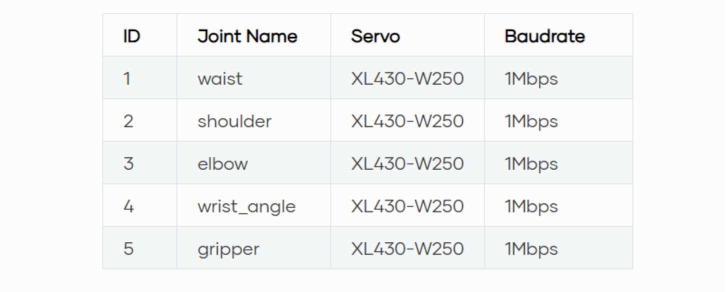
- Click `Set Path`

PincherX100 Robot Overview and Configuration

The PincherX100 robotic arm consists of:

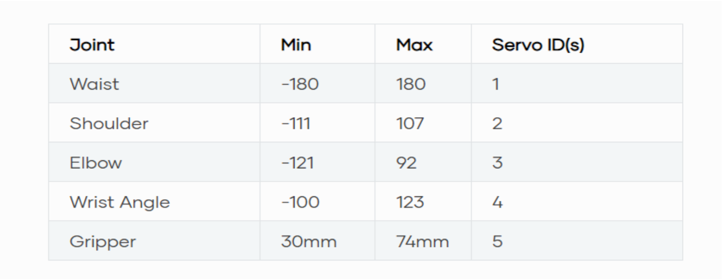
- Five Dynamixel XL430-W250 motors
- 4 Degrees of Freedom (DOF)
- A gripper at the end

This configuration allows for complex movements and manipulations. It's crucial to be aware of the joint limits to avoid self-collisions during operation.



ID	Joint Name	Servo	Baudrate
1	waist	XL430-W250	1Mbps
2	shoulder	XL430-W250	1Mbps
3	elbow	XL430-W250	1Mbps
4	wrist_angle	XL430-W250	1Mbps
5	gripper	XL430-W250	1Mbps

Joint Limits to Avoid Self-Collisions



Joint	Min	Max	Servo ID(s)
Waist	-180	180	1
Shoulder	-111	107	2
Elbow	-121	92	3
Wrist Angle	-100	123	4
Gripper	30mm	74mm	5

Communication with the Real Robot

This section details how to set up the communication between MATLAB and the Dynamixel motors, enabling control of each joint in the PincherX100 robotic arm. For more information regarding the dynamixel servos and control table click here Dynamixel servos

Load Dynamixel SDK

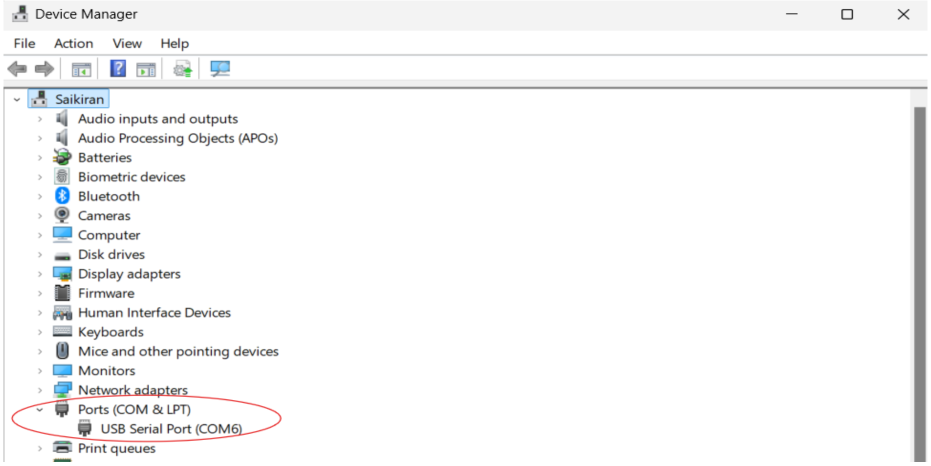
Download and build the Dynamixel SDK as described in the **Lab Requirements** section. Once installed, MATLAB can load the SDK based on your operating system. If you face any error in loading the SDK try rebuilding the library.

```
% Load the Dynamixel library
% once your build is completed successfully, on using this snippet it will load
the Dynamixel SDK depending whether it is Windows, Linux or Mac.
lib_name = '';
if strcmp(computer, 'PCWIN64')
    lib_name = 'dxl_x64_c';
elseif strcmp(computer, 'GLNXA64')
    lib_name = 'libdxl_x64_c';
elseif strcmp(computer, 'MACI64')
    lib_name = 'libdxl_mac_c';
end

if ~libisloaded(lib_name)
    [notfound, warnings] = loadlibrary(lib_name, 'dynamixel_sdk.h', ...
        'addheader', 'port_handler.h', 'addheader', 'packet_handler.h', ...
        'addheader', 'group_sync_write.h', 'addheader', 'group_sync_read.h',
...
        'addheader', 'group_bulk_read.h', 'addheader', 'group_bulk_write.h');
end
```

Setting Up Control Parameters

In this code, key control parameters are defined, including:

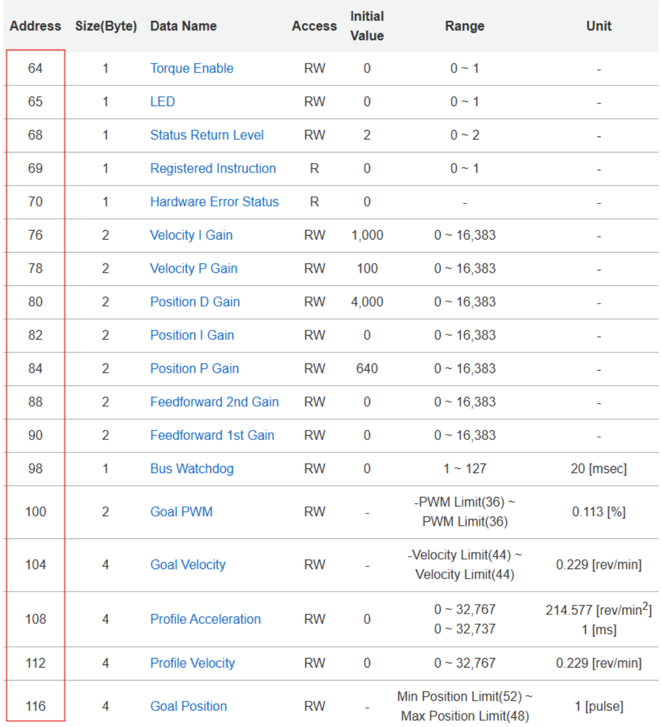
- **DXL_ID**: Array of motor IDs for each joint. Modify this to match your setup.
- **BAUDRATE**: Communication speed, set to 1 Mbps in this case.
- **DEVICENAME**: The port through which MATLAB communicates with the robot (e.g., COM6 in the figure below).



The addresses for a few specific control registers are also specified in the following figure. These include:

- ADDR_TORQUE_ENABLE: For enabling torque.
- ADDR_GOAL_POSITION: To set target positions for the servos.
- ADDR_PRESENT_POSITION: To read real-time position feedback from the servos.

For more information regarding the dynamixel servos and control table click here Dynamixel servos



Address	Size(Byte)	Data Name	Access	Initial Value	Range	Unit
64	1	Torque Enable	RW	0	0 ~ 1	-
65	1	LED	RW	0	0 ~ 1	-
68	1	Status Return Level	RW	2	0 ~ 2	-
69	1	Registered Instruction	R	0	0 ~ 1	-
70	1	Hardware Error Status	R	0	-	-
76	2	Velocity I Gain	RW	1,000	0 ~ 16,383	-
78	2	Velocity P Gain	RW	100	0 ~ 16,383	-
80	2	Position D Gain	RW	4,000	0 ~ 16,383	-
82	2	Position I Gain	RW	0	0 ~ 16,383	-
84	2	Position P Gain	RW	640	0 ~ 16,383	-
88	2	Feedforward 2nd Gain	RW	0	0 ~ 16,383	-
90	2	Feedforward 1st Gain	RW	0	0 ~ 16,383	-
98	1	Bus Watchdog	RW	0	1 ~ 127	20 [msec]
100	2	Goal PWM	RW	-	-PWM Limit(36) ~ PWM Limit(36)	0.113 [%]
104	4	Goal Velocity	RW	-	-Velocity Limit(44) ~ Velocity Limit(44)	0.229 [rev/min]
108	4	Profile Acceleration	RW	0	0 ~ 32,767 0 ~ 32,737	214.577 [rev/min^2] 1 [ms]
112	4	Profile Velocity	RW	0	0 ~ 32,767	0.229 [rev/min]
116	4	Goal Position	RW	-	Min Position Limit(52) ~ Max Position Limit(48)	1 [pulse]

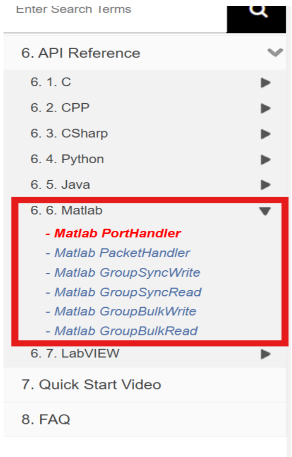
```
% Dynamixel control constants
DXL_ID = [1, 2, 3, 4, 5]; % ID’s of the servos you connected
BAUDRATE = 1000000; % by default use this
DEVICENAME = 'COM11'; % change according to your port number

% These are the addresses in the control table, please refer to the
documentation above
ADDR_OPERATING_MODE = 11;
ADDR_TORQUE_ENABLE = 64;
ADDR_GOAL_VELOCITY = 104;
ADDR_GOAL_POSITION = 116;
ADDR_PRESENT_VELOCITY = 128;
ADDR_PRESENT_POSITION = 132;

% Values of the bytes
LEN_GOAL_VELOCITY = 4;
LEN_PRESENT_VELOCITY = 4;
LEN_PRESENT_POSITION = 4;
VELOCITY_MODE = 1;
POSITION_MODE = 3;
PROTOCOL_VERSION = 2.0;
TORQUE_ENABLE = 1;
TORQUE_DISABLE = 0;

COMM_SUCCESS = 0;            % Communication Success result value
COMM_TX_FAIL = -1001;        % Communication Tx Failed
```

Initialize Handlers

Set up the communication handlers:

- **Port Handler**: Establishes the communication port.
- **Packet Handler**: Configures the protocol version (version 2.0 in this case).
- **Group Handlers**: Enables synchronized writing and reading across multiple motors.

For more information regarding the function APIs on MATLAB click here api_reference_matlab

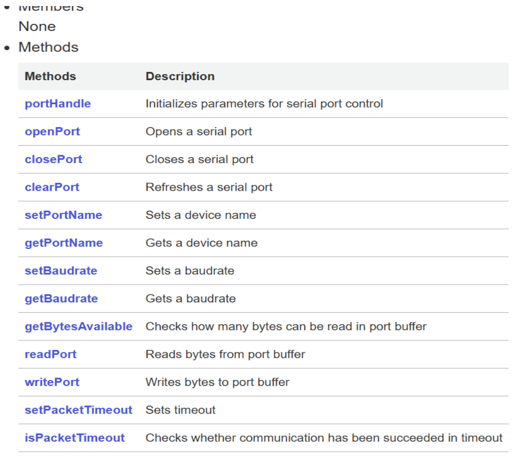


Enter Search Terms

6. API Reference
- 6. 1. C
- 6. 2. CPP
- 6. 3. CSharp
- 6. 4. Python
- 6. 5. Java
- 6. 6. Matlab
  - *Matlab PortHandler*
  - *Matlab PacketHandler*
  - *Matlab GroupSyncWrite*
  - *Matlab GroupSyncRead*
  - *Matlab GroupBulkWrite*
  - *Matlab GroupBulkRead*
- 6. 7. LabVIEW

7. Quick Start Video

8. FAQ



- Members

  None
- Methods

Methods	Description
portHandle	Initializes parameters for serial port control
openPort	Opens a serial port
closePort	Closes a serial port
clearPort	Refreshes a serial port
setPortName	Sets a device name
getPortName	Gets a device name
setBaudrate	Sets a baudrate
getBaudrate	Gets a baudrate
getBytesAvailable	Checks how many bytes can be read in port buffer
readPort	Reads bytes from port buffer
writePort	Writes bytes to port buffer
setPacketTimeout	Sets timeout
isPacketTimeout	Checks whether communication has been succeeded in timeout

```
% Initialize PortHandler and PacketHandler
port_num = portHandler(DEVICENAME);
packetHandler();
% Initialize groupBulkWrite Struct
groupwrite_num = groupBulkWrite(port_num, PROTOCOL_VERSION);
% Initialize Groupbulkread Structs
groupread_num = groupBulkRead(port_num, PROTOCOL_VERSION);
```

To begin communication, open the port and set the baud rate:

```
% Open port and set baudrate
if openPort(port_num)
    fprintf('Succeeded to open the port!\n');
else
    fprintf('Failed to open the port!\n');
    unloadlibrary(lib_name);
    return;
end
if setBaudRate(port_num, BAUDRATE)
    fprintf('Succeeded to change the baudrate!\n');
else
    fprintf('Failed to change the baudrate!\n');
    unloadlibrary(lib_name);
    return;
end
```

Reading Joint Positions and Velocities from the Robot

```
%% Reading Joint Positions and Velocities
% Add parameters for each motor
for i = 1:length(DXL_ID)
    % Add position parameter
    groupBulkReadAddParam(groupread_num, DXL_ID(i), ADDR_PRESENT_POSITION,
LEN_PRESENT_POSITION);
    % Add velocity parameter
    groupBulkReadAddParam(groupread_num, DXL_ID(i), ADDR_PRESENT_VELOCITY,
LEN_PRESENT_VELOCITY);
end

% Read data from the motors
groupBulkReadTxRxPacket(groupread_num);

fprintf('\nCurrent Joint States:\n');
for i = 1:length(DXL_ID)
    % Read position and velocity
    joint_position = groupBulkReadGetData(groupread_num, DXL_ID(i),
ADDR_PRESENT_POSITION, LEN_PRESENT_POSITION);
    joint_velocity = groupBulkReadGetData(groupread_num, DXL_ID(i),
ADDR_PRESENT_VELOCITY, LEN_PRESENT_VELOCITY);

    % Convert to integer values for display
    joint_position = typecast(uint32(joint_position), 'int32');
    joint_velocity = typecast(uint32(joint_velocity), 'int32');

    % Print joint position and velocity
    fprintf('[ID:%03d] Position: %d, Velocity: %d\n', DXL_ID(i),
joint_position, joint_velocity);
end

% Clear parameters after reading
groupBulkReadClearParam(groupread_num);
```

Position Control Mode

In this tutorial we are using position control mode to move the robot to the home configuration.

```
%% Initializing params and getting robot to home configuration using
position control mode
%% Setting joint-position limits

JOINT_LIMITS = [1000, 3050;
                 950, 3060;
                 950, 3060;
                 830, 3400;
                1850, 2700];

% Joint positions for Home config
goal_positions = [2050, 1250, 2500, 2350, 2500];
```

```
% Moving robot to Home configuration
for i = 1:length(DXL_ID)
    % Setting the operation mode to position control and enabling torque
    write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(i),
ADDR_OPERATING_MODE, POSITION_MODE);
    write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(i),
ADDR_TORQUE_ENABLE, TORQUE_ENABLE);

    % Sending the joint position commands
    param_goal_position = typecast(int32(goal_positions(i)), 'uint32');
    write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(i),
ADDR_GOAL_POSITION, param_goal_position);
end

pause(0.5) % Wait for the robot to reach Home config
```

Disabling torque to change operation mode to velocity control mode.

```
for i = 1:length(DXL_ID)

    % Disabling torque(to change operation mode, torque needs to be
disabled)
    write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(i),
ADDR_TORQUE_ENABLE, TORQUE_DISABLE);

    % Setting the operating mode to velocity control and re-enabling torque
    write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(i),
ADDR_OPERATING_MODE, VELOCITY_MODE);
    write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(i),
ADDR_TORQUE_ENABLE, TORQUE_ENABLE);

end
```

Velocity Control Mode

In Velocity Control Mode, the robot's joints move at specified speeds. This mode is useful for applications requiring continuous motion or speed adjustments.

```
%% Setting joint velocity using velocity control mode

% Example velocities for testing each motor
goal_velocities = [20, 20, -20, -20, -20];

timeout = 5;  % Duration in seconds to hold each velocity command
update_freq = 10; % in Hz
time_period = 1/update_freq;

tic;
tStart = toc;
tUpdate = toc;
```

```
% Initializing flags
FIRST_RUN = true;
STOP_FLAG = false;

while(true)
    t2 = toc;

    if (t2 - tUpdate > time_period || FIRST_RUN)
        fprintf("\n--------\nUpdating velocity!\n--------\n\n");
        for i = 1:length(DXL_ID)
            param_goal_velocity = typecast(int32(goal_velocities(i)),
'uint32');
            groupBulkWriteAddParam(groupwrite_num, DXL_ID(i),
ADDR_GOAL_VELOCITY, LEN_GOAL_VELOCITY, param_goal_velocity,
LEN_GOAL_VELOCITY);
        end

        % Writing the bulk packet
        groupBulkWriteTxPacket(groupwrite_num);
        groupBulkWriteClearParam(groupwrite_num);

        % Resetting tUpdate
        tUpdate = toc;
        FIRST_RUN = false;
    end

    % Reading the packet
    groupBulkReadTxRxPacket(groupread_num);
```

Reading the joint positions from the robot and checking for joint limit violations

```
% Reading the joint positions
    for i=1:length(DXL_ID)
        % Reading the joint positions
        val_read = groupBulkReadGetData(groupread_num, DXL_ID(i),
ADDR_PRESENT_POSITION, LEN_PRESENT_POSITION);
        fprintf('[ID:%03d] JointPos:%d\n', DXL_ID(i),
typecast(uint32(val_read), 'int32'));

        % Checking for joint limit violations
        if(val_read < JOINT_LIMITS(i, 1) || val_read > JOINT_LIMITS(i, 2))
            STOP_FLAG = true;
            break
        end
    end
    fprintf("------x------\n");
    t = toc;

    % Exit condition
    if(t - tStart > timeout || STOP_FLAG)
        if STOP_FLAG
            fprintf("\n\n==========x==========\n\nWARNING!\nJoint limit
exceeded!\n")
        else
            fprintf("\n\n==========x==========\n\nTimeout!\n");
        end
        break
    end
end

for i=1:length(DXL_ID)
    % Stop the motor after the duration by setting velocity to 0
    param_goal_velocity = typecast(int32(0), 'uint32');
    groupBulkWriteAddParam(groupwrite_num, DXL_ID(i), ADDR_GOAL_VELOCITY,
LEN_GOAL_VELOCITY, param_goal_velocity, LEN_GOAL_VELOCITY);
end

% Writing the bulk packet
groupBulkWriteTxPacket(groupwrite_num);
groupBulkWriteClearParam(groupwrite_num);
```

Disable Torque on Motors

```
prompt = "Enter any key to disable: ";
input(prompt, "s");

% Disabling the torque
for i = 1:length(DXL_ID)
    write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID(i),
ADDR_TORQUE_ENABLE, TORQUE_DISABLE);
end
```

Close port and unload library

```
% Close port and unload library
closePort(port_num);
unloadlibrary(lib_name);
```