EE366L/CE366L: Introduction to Robotics

Lab Handbook

Second Edition, Spring 2024

If you come across any errors, contact me at

basit.memon@sse.habib.edu.pk

Table of Contents

Preface

This handbook was developed for the companion lab to 'EE 366/CE 366/CS 380: Introduction to Robotics' course offered at Habib University in the Dhanani School of Science and Engineering. While developing this lab, I have tried to achieve three objectives: (i) provide students the experience of building complex robotic systems from constituent sub-systems; (ii) train students to adjust for the differences between theoretical models and physical systems in their system design; (iii) build the students' confidence and prepare them for building robots independently.

I believe that the best way to achieve these objectives is for the students to build the sub-systems themselves from the ground up. Admittedly, this approach limits the students to simple robotic systems given the available time, but if the students' understanding is enhanced through these simpler systems then it will be easier for them to extrapolate to more complex robotic systems. Unfamiliarity with ROS can be considered a shortcoming of this approach, and I recommend students to acquire a passing familiarity with it if they get a chance, and the understanding of homogeneous transformations acquired in this course will certainly make it convenient to understand transform trees in ROS.

Basit Memon
January 8, 2023

Acknowledgments

I would like to acknowledge the help of Mr. Waleed Bin Khalid and Mr. Hassan Shah, who were RAs in DSSE, for their assistance in setting up the robotics lab in Spring 2022. I would also like to thank the students enrolled at the time in the course, who volunteered their time to assemble the robot arms during the winter break and subsequently became the beta testers. It would not have been possible to conduct this lab timely, otherwise.

I would also acknowledge the continued support and assistance of Mr. Hassan Shah, RA for this course in Spring 22. His passion, insights, research, and debugging efforts have led to this improved second edition of this handbook.

Change Log

2024

Some typographical errors were corrected. Objectives and hardware/software requirements have been added at the beginning of each chapter. Installation guide has been included as Chapter 0. Hardware exploration has been limited to one chapter. Discussion of accuracy and repeatability has been removed. Vision has been moved to the beginning of the course. Practice for various image segmentation techniques is carried out first before the task to develop vision pipeline is assigned.

CHAPTER 0

Lab Instructions

1. Since you're working in groups, rotate among yourselves so that each member of the group has a chance to get hands-on experience.
2. Tasks marked with an [*] don't require access to the hardware and can be completed post-lab.
3. At the end of the lab, you're required to unplug the arm from the power supply and turn off the lab computer's monitor. The arm may relax and fall down as it is turned off, so be sure to catch it and carefully place it on the baseboard.
4. The weekly lab findings report is a group submission.

"It is not knowledge, but the act of learning, not possession but the act of getting there, which grants the greatest enjoyment."

- Carl Friedrich Gauss

CHAPTER 1

Getting familiar with MATLAB

As MATLAB will be used extensively throughout this course and students may have different levels of familiarity with it, today's lab is dedicated to familiarizing yourself with the MATLAB environment. For this assignment, you're required to complete the MATLAB On-ramp (https://matlabacademy.mathworks.com/details/matlab-onramp/gettingstarted) online course and upload the completion certificate.

This is a two-hours course, which provides a basic understanding of MATLAB's IDE, data storage and manipulation, and programming constructs. If you're already familiar with MATLAB, this will be a boring exercise and you can reach out to your instructors to select an advanced MATLAB course to complete for this assignment.

Task 1.1 Team Formation (10 points)

Due to limited number of lab setups, you'll be working on your arm project in groups. You are required to enroll yourself as part of a group on Canvas. If you're unfamiliar with the process of self-enrolling in a group on Canvas, you can follow a guide on Canvas.

Task 1.2 MATLAB On-Ramp (90 points)

Attach the completion certificate for the MATLAB On-Ramp course.

"And each day I learn just a little bit more.
I dont know why, but I do know what for."

– Elton John

CHAPTER 2

Getting familiar with the arm hardware

The objectives of this lab are to:

(i) get familiar with the hardware and capabilities of the Phantom X Pincher arm, shown in Figure 2.1 and manufactured by Trossen Robotics, which will be utilized for this project;

(ii) appreciate the motion and manipulation limitations of this arm;

(iii) look closely at the sensing, actuation, and the functional decomposition of our complete robotic arm system;

(iv) explain the arm's operational abilities with simple physics.

Software dependency: The utilized library functions require, at minimum:

- Arduino IDE 1.6.10
- FTDI drivers
- Interbotix ArmLink
- DYNAPose

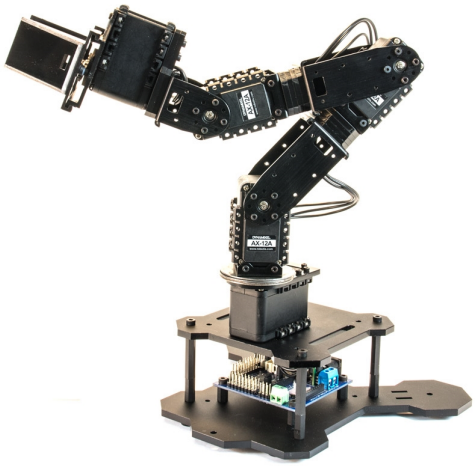


Figure 2.1: Phantom X Pincher Arm

2.1 Design of the arm

The arm has 5 motors that serve as actuators, a pinch gripper, and a microcontroller board at the bottom. We'll discuss each of these parts in this lab. A 3D interactive model of the same arm is provided by the manufacturer at this link: https://www.trossenrobotics.com/p/PhantomX-Pincher-Robot-Arm.aspx. The 3D model is built with approximately the same dimensions as physical arm, and also allows you to measure the distances and angles at different points on the robot. While the arm is **not powered up**, feel free to move the arm by hand for the following tasks. **Don't try to move the arm forcefully. If it is resisting motion, then it is powered up and the motors are providing torque to the joints. It will resist motion and forcefully moving the arm will damage the motors.**

Task 2.1 Model of the arm (25 points)

We know that a robot manipulator is mathematically modeled as a kinematic chain, made up of joints and links. Identify all the joints and links in this arm.

(a) Mark all the joints and links in Figure 2.1 or any other image of the arm.

(b) How many joints and links are in this arm? Note that the motor attached to the gripper is only responsible for opening and closing the gripper.

(c) What is the joint type? Provide a symbolic representation of the kinematic chain corresponding to this arm. Recall that a kinematic chain is symbolically represented as a sequence of joint symbols.

(d) How many degrees of freedom does this arm possess? *Hint: You can use Grubler's formula from the class slides.*

(e) Will you be able to arbitrarily position and orient this arm within its workspace?

2.2 The Arbotix-M Controller

The Arbotix-M controller, depicted in Figure 2.3, receives higher level instructions from an external processing unit and generates instructions in the format required for the servomotors (actuators) installed in this arm. In our case, a computer will serve as external processing unit and we'll pass instructions from computer to Arbotix-M, as shown in Figure 2.2.

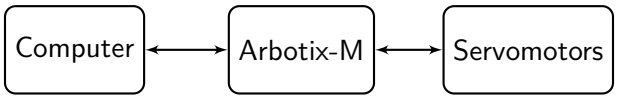


Figure 2.2: Data flow between MATLAB and servomotors

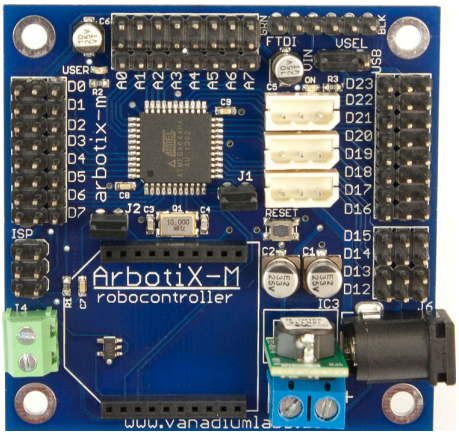


Figure 2.3: The Arbotix-M Controller

The board can also be used for additional processing. Figure 2.4 depicts the purpose of various sections on this board. It uses an ATMEGA644p AVR microcontroller as a processor, which is in the same category of microcontrollers as an Arduino. In fact, we'll be using the Arduino IDE to program the Arbotix-M in C.

For the time being, we'll only be using this board for controlling the servomotors in the arm. For this, the board can be set up as shown in Figure 2.5. We simply need to

- power the board via the DC power jack, which will provide power to all the connected motors as well;
- set up communication between the board and the computer using the provided FTDI-USB cable; this cable can only be connected in one way as shown in Figure 2.6; the port on the board also indicates which side corresponds to the green cable and which side to the black;
- connect the servomotors to the Dynamixel servo ports.

In addition to this, the required firmware files should be copied to `Documents\Arduino` folder on Windows as indicated in Appendix-A. We'll verify this in 2.3.1.

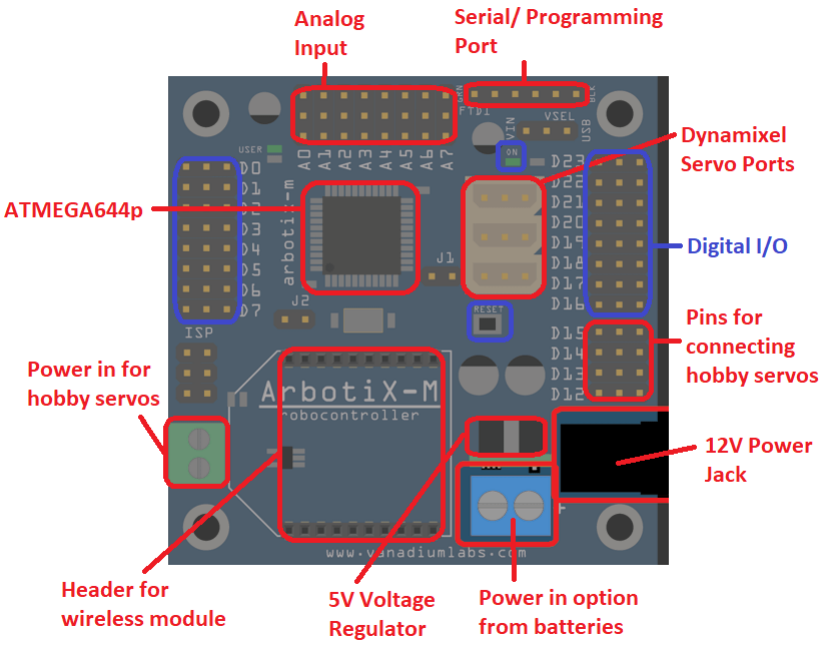


Figure 2.4: Getting to know Arbotix-M

2.2.1 Connecting motors to Arbotix-M

The motors are connected in a daisy chain, i.e. only the base motor is connected to the Arbotix, the second motor is connected to the first, and so on serially all the way to the gripper motor. Then, how does Arbotix communicate with a specific motor? Each motor is assigned an ID, Arbotix addresses each instruction message to the intended ID, and places it on the common chain/bus. The IDs assigned to the five motors on the arm are give in Figure 2.7. You can also broadcast messages intended for all motors.

2.3 Move the arm

We'll now use a simple interface, InterbotiX Arm Link Software, provided by the manufacturer to control the arm. In future, you'll write our own program to control it.

2.3.1 Setting up Armlink

1. Connect the power jack to Arbotix-M and make sure that FTDI-USB cable is plugged into a USB port in your computer. When the arm is first powered up, it may move to its 'sleep' position and then turn torque off to all the servos. Don't be alarmed by the motion and don't interrupt its execution.

2. Open the Arduino IDE. It is already installed on the lab computers. The firmware provided for the arm is only compatible with an older version of Arduino IDE, specifically ver.1.0.6.

3. We'll now need to load the appropriate firmware on Arbotix-M that will allow it to communicate with the Armlink software, assuming that the firmware is appropriately placed

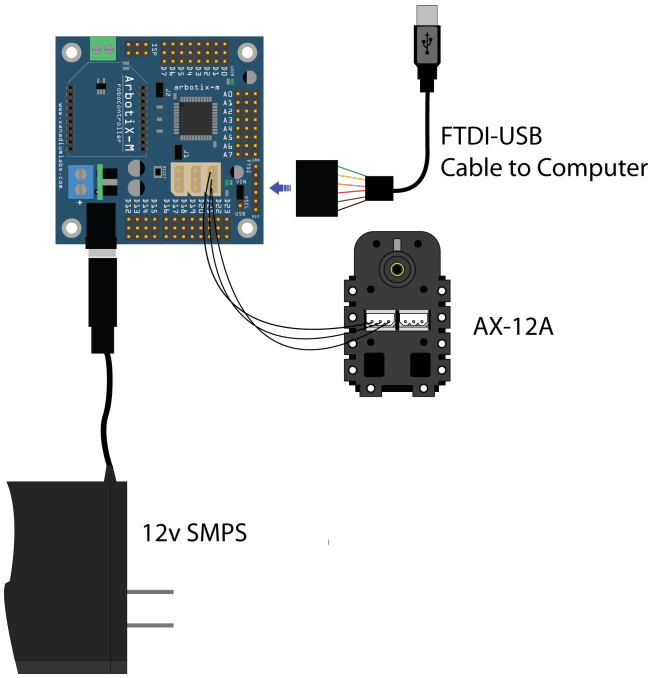


Figure 2.5: Setting up the Arbotix-M with power, servos, and programming

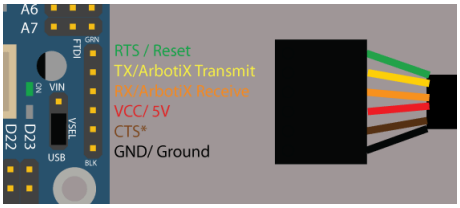


Figure 2.6: FTDI-USB Cable Connection

Figure 2.7: IDs of the servos in arm

on your computer as specified in Appendix-A. This will be verified if you're able to carry out the following steps.

(i) Verify that the libraries `ArmLink` and `Bioloid` are available under `Sketch`.

`Sketch -> ArmLink`

(ii) Select the appropriate board and programmer as follows:

`Tools -> Board -> ArbotiX`
`Tools -> Programmer -> AVRISP mkII (Serial)`

(iii) Open the `ArmLinkSerial` firmware from the Arduino IDE (Arobotix-M firmware requires Arduino 1.0.6).

`File -> Examples -> Arm Link -> InterbotixArmLinkSerial`

(iv) You have to select our arm model by uncommenting, i.e. remove `//`, from line number 60 in the code. The line should read:

```
#define ARMTYPE PINCHER
```

(v) Load this firmware onto the Arbotix-M, by clicking on the `Upload` icon, which is a right arrow, in the toolbar or from the menu,

`Sketch -> Upload`

(vi) Once the firmware is uploaded, you will see `Done Uploading` message in the green bar at the bottom of your IDE. This firmware sets up a protocol for Arbotix-M to communicate with the ArmLink software over USB, and convert received messages to instructions for motors.

4. Open the `ArmLink` application. The application is already copied to `Desktop\ArmLink_1.6_Win64` on the lab computers. When the application is launched, click on `Auto Search`. This will search for the attached arm and connect to it.

5. On a successful connection, the arm will move from its 'sleep' position to a 'home' position. This may take several seconds. Once the arm has moved to its home position and is ready for commands, the various panels will appear as shown in Figure 2.8. **Once the arm is connected to the software, don't try to move it by hand as each of the motors will exert torque.**

6. You can adjust the sliders or text panels to adjust the positions of the arm. You can send these values to Arbotix-M by clicking on `Update`, or you can check `Auto Update`, in which case instructions will be sent continuously. **Read through the next two sections carefully to be fully aware of safety instructions before playing around with the controls.** It is better to use the `Update` option so that you can keep track of cause and action.

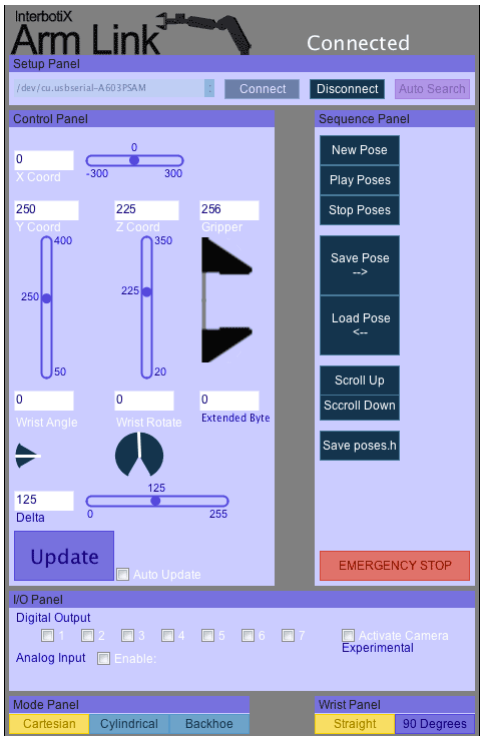


Figure 2.8: Arm Link Application Panel

2.3.2 Getting familiar with Arm Link Controls

Change the panel controls gradually so that it does not collide with itself or an object in the environment. It may appear that the arm is not moving, but it requires some time to process the received coordinates. Always be alert to stop the arm from colliding with itself or the environment. You can stop it by clicking on the 'Emergency Stop' button or unplugging power.

The software provide three modes of operation - two in the task space representation, and one in the configuration space, where you can control each joint individually . These modes can be selected from the bottom of the panel.

1. **Cartesian:** In this task-space mode, you can specify the X-Y-Z coordinates in the task space and the software will move the end-effector to that location.

2. **Cylindrical:** As the name suggests, you can specify the cylindrical coordinate for the placement of the end-effector in this mode. The panel gives you the option to set the base angle, Y, and Z coordinates.

3. **Backhoe:** In this mode, you can directly control the position of the base, shoulder, and elbow joints. It is important that you don't change the angle by a large amount (greater than $30°$) in one iteration.

In addition to this,

- The `Straight` and `90 Degrees` wrist option place the wrist in the horizontal or vertical configuration.
- Each mode allows you to rotate the wrist.
- Each mode allows you to open and close the gripper.
- The `Delta` value determines how long it will take for the arm to move from its current position to the new position. The amount of time that the move takes is calculated by multiplying `Delta` by 16. The result will give you the time interval in milliseconds.
- The right panel allow you to save different poses to create a sequence and play it on the arm.

2.3.3 Common Problems

What to do if you have provided commands from the software panel, but the arm is not moving as intended? Make sure that you have waited sufficiently, as the arm requires some time to process any received instructions. If sufficient time has passed and the arm is still not moving, check that

(i) the motor cables for any of the motors have not wrapped around, restricting the motion of the arm;

(ii) none of the motors have a flashing red indicator light; the light indicates one of these listed events: over-temperature, joint angle instruction exceeds allowed limit, excessive torque.

If you're aware of the instruction causing this error, reverse it. Or, you could reset the arm by powering it off and manually moving it to zero configuration, if needed. If it is an over-temperature event, the arm will have to be powered off for a longer duration for the arm motors to cool down.

Task 2.2 Configurations Exploration (15 points)

Play around with the different modes of motion in the software and explore the capabilities and limitations of this arm.

(a) Think about a configuration in which the arm reaches the farthest possible point. Move the arm to verify. Draw this configuration as a diagram. In this diagram, links can be represented by line segments and revolute joints by circles.

(b) In Cartesian mode, move the robot to an arbitrary (x, y, z) location. Change the wrist angle from the panel and observe what happens to the other joints of the arm. Document your observations and comment on the reasons behind what you observe.

(c) [*][a] The coordinates in the Cartesian or Cylindrical mode describe task space locations. Task space can be used to describe tasks to be carried out by the manipula-

tor, e.g. grabbing a water bottle. Give an example of a task that can be described better in Cartesian coordinates, and a task, which is best expressed in cylindrical coordinates.

[a]Tasks marked with [*] can be completed post-lab.

2.4 Establishing a coordinate system

The numbers appearing in the Arm Link panels for X, Y, Z values appear to map the physically possible range along an axis onto a subset of integers, e.g. X, onto a subset of integers between 0 and 1023[1]. So, this must be based on an underlying coordinate system and we could use the same in our design.

Task 2.3 Coordinate Axes (20 points)

(a) Determine the directions of positive x, y, and z axes and mark them on paper, in relation to the asymmetric shape of the black base.

(b) We'll set the origin of the x and y axes at the center of the shaft of the first motor, and the origin of the z axis at the level of the wooden baseboard. If 1 unit in the ArmLink system corresponds to 1 unit in the real world, identify the units being utilized in the real world and the point on the arm corresponding to these coordinates.

2.5 Vision-based Pick and Place

Definition 2.5.1: Sensors

Sensors are physical devices that enable a robot to perceive its physical environment in order to get information about itself and its surroundings.

Definition 2.5.2: Actuators

End-effectors use underlying mechanisms, such as muscles and motors, which are called actuators and which do the actual work for the robot.

Task 2.4 Manipulation (5 points)

Grab one each of the provided objects. In this task, you'll place each of these objects at a fixed location in your workspace, move the arm using `ArmLink` in Cartesian mode to that location, pick the object with the arm, and place it at another location. A ruler may be of help to you.

[1]The number range is 0-1023, because the arm servos use a 10 bit location for motor angle. Limitations on the possible range are governed by (a) the physical angular limits of the motors, (b) safety margins to avoid self-collisions of the arm.

(a) Describe the steps taken as a block diagram. How is the real world environment being sensed in this case? How is the arm motion being adjusted based on the received sensing data? Where is this processing happening?

2.5.1 Servomotors

A servomotor is a regular DC motor coupled with sensing, to measure the rotational position of the output shaft, and a controller, which uses that position feedback to precisely control the position of the motor. Advanced servos can also control the motor's angular velocity and acceleration. This feedback loop is illustrated in the block diagram in Figure 2.9.

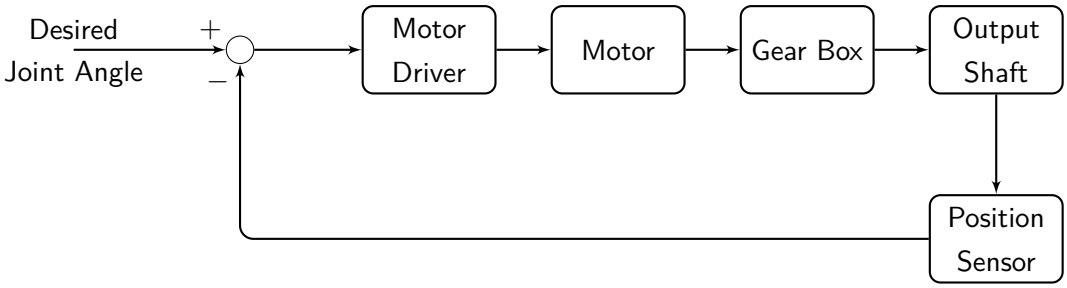


Figure 2.9: Block diagram of a servomotor

Our robot arm has five Dynamixel AX-12A servomotors. The functional components highlighted in the previous block diagram are captured in Figure 2.10 (Zoom in to see the different components labeled in the figure). A complete tear-down of a Dynamixel AX-12A motor is captured in this video: https://youtu.be/1v-vgnHDV68.

The controller on the Dynamixel AX-12A motors has a set instruction format for its read and write instructions used for reading and writing the values of its registers. The structure to be followed in every instruction message packet is completely outlined in Dynamixel reference manual. The Arbotix-M plays the role of an intermediary, allowing us to use its easier C libraries for setting and getting the position of each motor, while the libraries translate our instructions to the packet structure required by AX-12A motors.

Task 2.5 Design of vision-based system (20 points)

(a) [*] Figure 2.10 suggests that a potentiometer is being used as the shaft position sensor. Research and describe how can a potentiometer be used for this purpose.

(b) You're now aware that the servomotors installed in the arm expect the desired joint angle as input. Furthermore, imagine that you now have an overhead camera available to you and you'll not use the Armlink software. Draw a block diagram of a robotic system for a pick and place task. Each block's text should describe the block's function. Try to add as much detail as you can to your diagram.

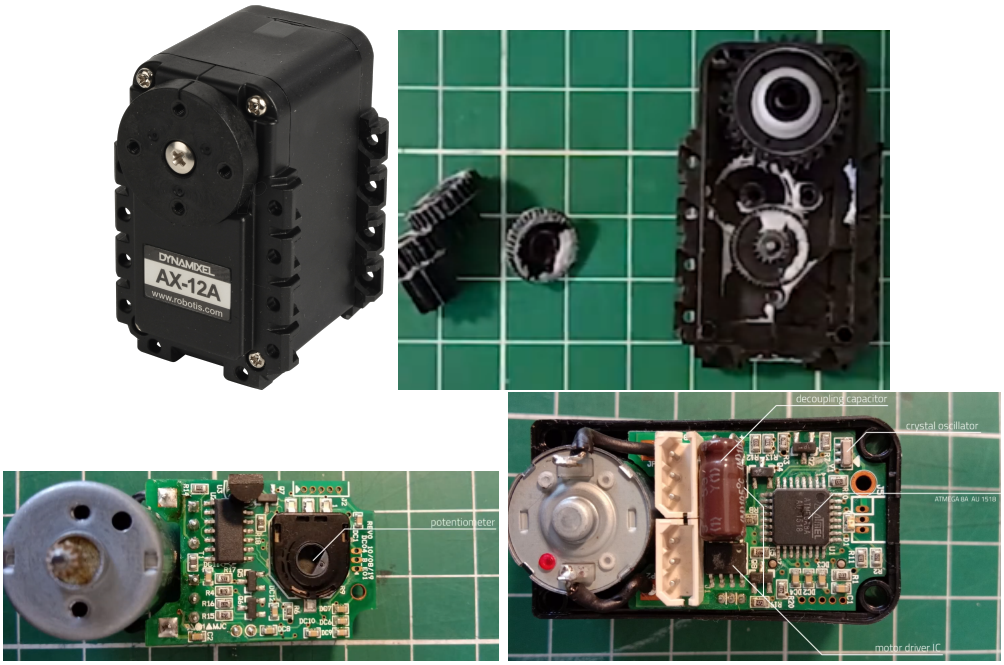


Figure 2.10: Top Left: AX-12A Motor, Top Right: Gear Box, Bottom Left: Potentiometer, Bottom Right: Microcontroller and Motor Driver

(c) [*] Identify and list all the sensors and actuators in your complete robotic system.

2.6 Limits of the manipulator

Task 2.6 Motor Specifications (15 points)

This task is about familiarizing ourselves with the Dynamixel reference manual (https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/) and some relevant specifications included in it.

(a) Find the angle rotation limits, resolution (see the definition below), speed limit, and torque limit[a] of AX-12A servo.

(b) [*] Will this motor resolution limit the possible Cartesian resolution of the end-effector? If yes, why?

(c) Will you be able to pick an object placed in any configuration in the robot's workspace? Justify your response.

[a] Note that the specifications provide the stall torque and is maximum torque the motor is capable of generating. This is the torque required to hold the load/weight connected to the motor shaft in position. For the same mass, you need torque greater than stall torque to move that mass, depending on required acceleration.

Definition 2.6.1: Resolution

This specification represents the smallest incremental joint motion that can be produced and sensed by the robot. A robotic system's resolution depends on its sensing capabilities.

2.7 Teaching Pendant

Most robot arms come with the ability to learn a motion, by allowing the user to move it by hand and storing the sensor values at various waypoints on the path. To teach a motion to the robot, we'll use another script, `DYNAPose`, as it allows us to relax the servos and move the arm by hand to teach it a pose.

(i) Follow the instructions in the *Download Code* section at https://learn.trossenrobotics.com/8-arbotix/131-dynapose-dynamixel-arbotix-pose-tool.html to upload the appropriate firmware on Arbotix-M for this task. Note the following:

- The code file is downloaded and available on the desktop on lab computers.
- The serial monitor is opened by clicking on the magnifying glass icon in the top-right corner of IDE.
- Don't forget to set the baud rate. This is the rate (bits/s) at which IDE will communicate with Arobotix-M.

(ii) The menu options provided on the webpage are incorrect and options specified in DYNA-Pose code are as follows:

```
0: Relax Servos
1: Enable Torque and Report Servo Position
2: Save Current Position to next pose(2)
3: Display All Poses in memory
4: Play Sequence Once
5: Play Sequence Repeat
6: Change Speed
7: Set Next Pose Number
8: Center All Servos
9: Scan All Servos
```

(iii) Relax the servos, which will allow us to move the arm by hand. This is option 0 in the menu.

(iv) Teach a simple motion to the robot. This could be as simple as moving to the corner of an object, which will not change its location, executed in at most 5 steps. At each step, after positioning the arm by hand, save that pose (Option 2). The final pose will be when the end-effector is at the corner of the object.

(v) Play the sequence of poses (option 4)

Task 2.7 Bonus (10 points)

Now, let's try a multi-step task. Think of a simple task, involving both positioning of arm and manipulation, which you would want to carry out with the arm right now, e.g. writing a single alphabet on paper. Teach the arm (use `DYNAPose`) to carry it out by recording poses at sufficient gaps and recreate that motion. Some questions you may have to think about are: How many save points do you need to describe your task? What poses should you save? When do you lift your arm off the page? What is the best orientation to grab a pen? For your submission, provide a written description of the task, a video of the execution, and your comments on how it can be improved.

2.8 How much weight can the arm lift? (Optional Read)

The aim of this section is to make sense of the listed strength specifications of the Phantom X Pincher arm, reproduced for convenience in Table 2.1 from the manufacturer's literature.

Strength	25 cm / 40 g; 20 cm / 70 g; 10 cm / 100 g
Gripper strength	500 g
Wrist lift strength	250 g

Table 2.1: Strength specifications of Phantom X Pincher

Towards that aim, you'll utilize your prior physics knowledge to develop the ability to perform back of the envelope calculations for strength.

2.8.1 Torque requirements for lifting

The lifting abilities of an arm are primarily constrained by the torque supported by the installed actuators. In this section, you'll calculate the torque required at each joint to hold the arms steady in horizontal position such that the arm does not drop due to its own weight or the weight of the load.

Let's start by considering a one-link arm, shown in Figure 2.11, which can rotate about the point $O2$. Assume that the link is of length L, has mass m_1, and the arm is carrying a load of mass M_L. Then the torque exerted by the load at any position of the arm is given by

$$\tau = (M_L g) L \sin\theta + (m_1 g) \left(\frac{L}{2}\right) \sin\theta,$$

where the first term is due to the weight of the load and the second term because of the weight of the link itself. It is assumed that the center of mass of the link is at its mid-point. A motor installed at joint $O2$ will have to exert the same amount of torque to maintain the load at its current position. Note that we're doing this calculation at static equilibrium, i.e. to achieve zero net generalized forces on the arm, when it is not in motion. Since we're currently only interested in the maximum torque required from the motor, we can simply consider the worst-case position

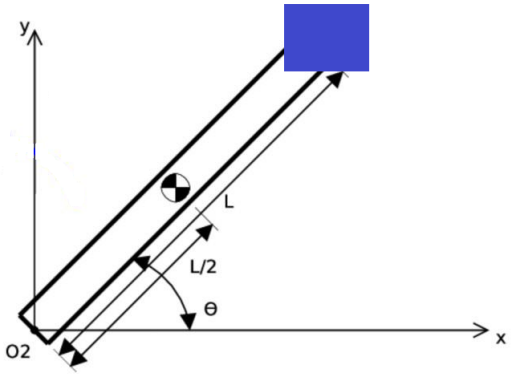


Figure 2.11: One link arm

of the arm, i.e. when the arm is horizontal. In this case,

$$\tau = M_L g L + m_1 g \frac{L}{2}.$$

Now consider the case of a multi-link robot arm shown in Figure 2.12. The lengths and masses

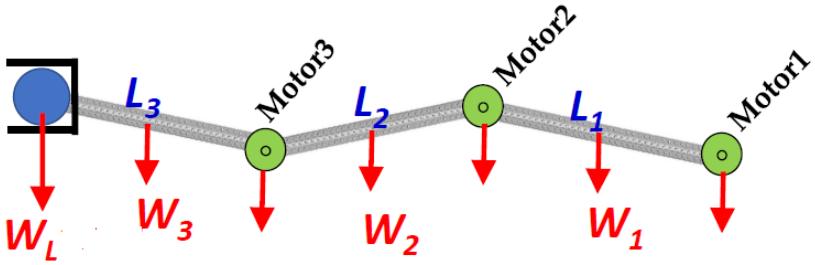


Figure 2.12: Multi-link robot arm

of the links going outwards from the base are L_i and m_i respectively. The mass of the motors attached at each joint are M_i respectively. A load of mass M_L is attached at the end of this arm, and the mass value includes the mass of the end-effector. Then, the torque demands on each of the motors can be computed as:

$$\tau_3 = M_L g L_3 + m_3 g \frac{L_3}{2}$$

$$\tau_2 = M_L g (L_2 + L_3) + m_3 g \left(L_2 + \frac{L_3}{2} \right) + M_3 g L_2 + m_2 g \frac{L_2}{2}$$

$$\tau_1 = M_L g (L_1 + L_2 + L_3) + m_3 g \left(L_1 + L_2 + \frac{L_3}{2} \right) + M_3 g (L_1 + L_2) + m_2 g \left(L_1 + \frac{L_2}{2} \right) + M_2 g L_1 + m_1 g \frac{L_1}{2}$$

Practically, the motors will be required to produce greater torque than these values as frictional and dynamic effects have not been catered in these calculations. The torque required for motion can be calculated by adding another torque term to these values based on the formula $\tau = I\alpha$, where I is the moment of inertia and α is the angular acceleration.

2.8.2 Gripping Force

In this section, we'll review the physics behind the calculation for the gripping force needed to grasp an object. These are again calculations for the condition of static equilibrium. Assume that we have a parallel gripper, as shown in Figure 2.13. Let F be the force being exerted by

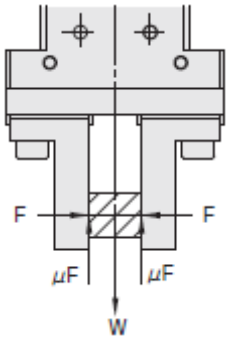


Figure 2.13: Force required to grip an object

the gripper on an object of weight W on either side. The coefficient of friction is μ, which depends on the two materials in contact. Because of the force being exerted by the gripper, an equal reaction force N is generated. For static equilibrium, the frictional force, F_s, should be equal to the weight of the object, i.e.

$$F_s = W$$
$$\mu N = W$$
$$F = \frac{W}{\mu}$$

Therefore, a gripping force of at least W/μ is required to keep holding an object of weight W against the gravitational pull.

Task 2.8 Bonus (10 points)

According to the manufacturer's provided heuristic [1], the load on any motor should not exceed $1/5$ of the stall torque. Keeping this heuristic, motor specifications, mass measurements in Figure 2.14, and physical principles outlined in Section 2.8.1, verify the wrist lift strength specified by the manufacturer. The wrist lift strength is the load that the wrist joint can support.

References

[1] L. ROBOTIS Co. "How much is n.m when converted to kgf.cm?" (), [Online]. Available: https://en.robotis.com/model/board.php?bo_table=robotis_faq_en&wr_id=33&sca=GENERAL.

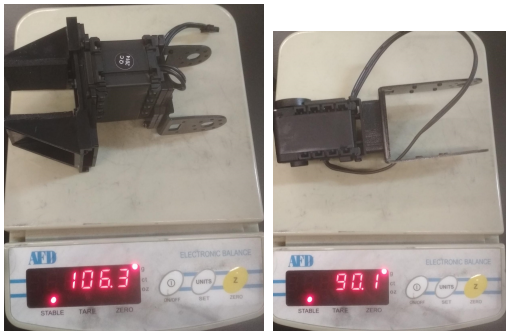


Figure 2.14: Masses of different arm components

Vision is the process of discovering from images what is present in the world and where it is.

– David Marr

CHAPTER 3

Perception-I

The objectives of this lab are to:

(i) explore the camera being used;

(ii) develop an interpretation of our complete robotic arm system in terms of frames;

(iii) gain familiarity with image manipulation and processing in MATLAB;

(iv) design a vision pipeline for object pose estimation using depth images.

Software dependency: The utilized library functions require, at minimum:

- Intel RealSense SDK 2.0
- MATLAB Image Processing Toolbox

3.1 System Overview

In the last lab, you carried out a manual pick and place operation, identifying the position the object through your eyes or a ruler, and then correspondingly adjusting the gripper position using manual control through the provided interfacing software. However, the output of your project will be an autonomous vision-based pick and place robot, for which a number of questions will have to be answered:

- Is it possible to find the location of object to be picked from an image? How?
- Given desired position of gripper, how will we find the corresponding joint angles?
- How will we find the current position of the gripper?
- How will we make sure that each joint achieves the desired angle?

You would have thought of these questions when developing your block diagram in the previous lab, and will certainly find answer to these as part of the design of your robotic system.

The overall robotic system is captured in the diagram in Figure 3.1, which answers some of these questions.

Figure 3.1: Block diagram of our vision-based pick and place robot

Note that this block diagram indicates that our system is not operating in a closed loop, i.e. there is no feedback being obtained from the camera about the positioning of the gripper. We can also be implement a complete visual feedback-based closed loop, but we'll operate our system in the open loop configuration of Figure 3.1 for now.

Task 3.1 Frame Assignment (30 points)

Given the functional blocks of Figure 3.1,

(a) assign frames to rigid bodies of interest in Figure 3.2.

(b) describe the pick and place task in terms of frame descriptions, e.g. determine ${}^{A}T_{B}$, the description of frame B with respect to frame A.

3.2 Camera

As outlined in Figure 3.1, the first step in our pick and place pipeline is to determine the frames of the objects to be picked, using an image of the environment from some camera. The frame of this object will be offset to determine the desired frame of the gripper for grasping the object, and the arm joints will be moved correspondingly to achieved this desired gripper placement.

3.2.1 Working with SR-305 Camera

You'll be using Intel RealSense SR305 camera, shown in Figure 3.3, in this project. A typical camera provides a 2D image, but this camera belongs to the class of RGB-D cameras that

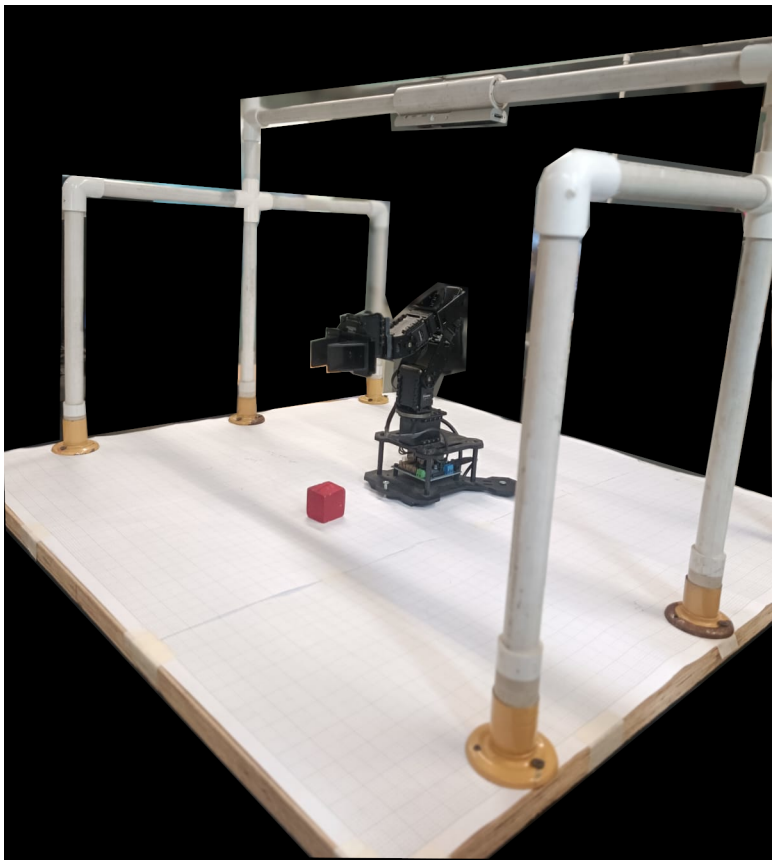


Figure 3.2: Pick and Place setup

provide a depth image, i.e. distance of objects from the camera, in addition to the usual 2D color image. This allows us to capture the 3D world in its entirety by using multiple images. A number of ways are being used in practice today to calculate depth, e.g. stereo images,

Figure 3.3: Intel RealSense SR305 Camera

LiDAR (time of flight), etc. The SR305 works on the principle of coded light. In addition to an RGB camera, providing the usual color image, an SR305 has an IR (Infrared) camera and an IR emitter, as seen in Figure 3.4. The emitter projects patterned light, typically a pattern of vertical bars, on to a scene, as illustrated in Figure 3.5. The bars illuminate certain pixels in the scene. Depth is determined by how the pattern is deformed by the object. Usually, a temporal sequence of patterns is utilized, giving each pixel a unique code in terms of the sequence, to

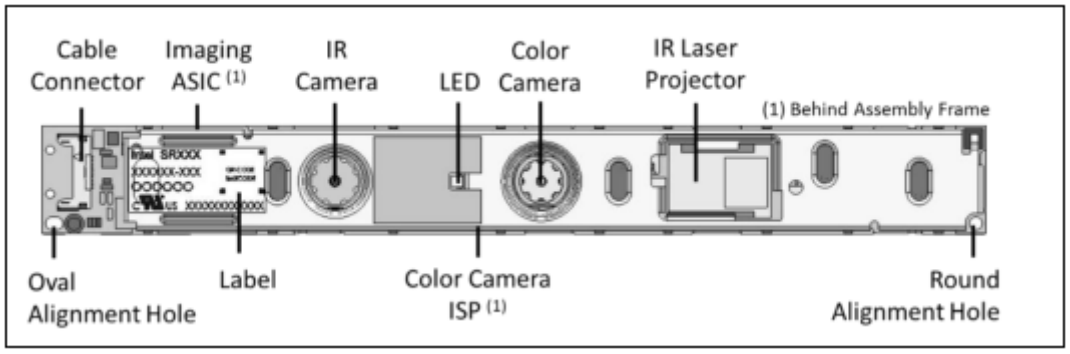


Figure 3.4: Components of an SR-305 Camera

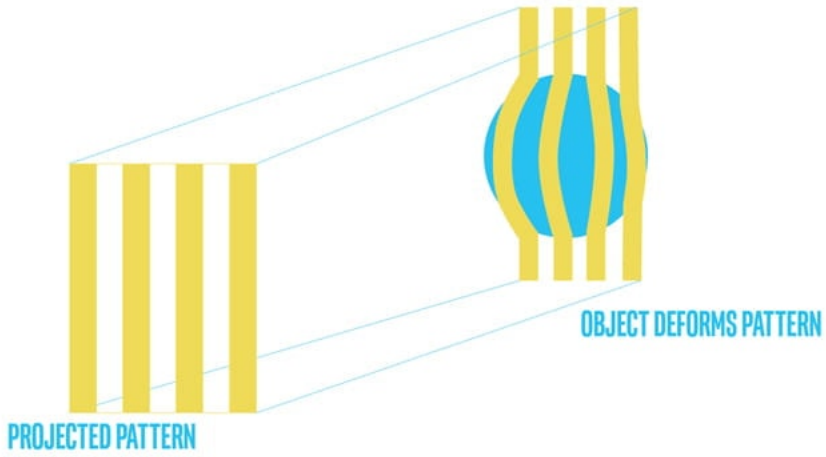


Figure 3.5: Principle of coded light

make the problem of matching illuminated pixels to their source light bars easier. The video at https://youtu.be/3S3xLUXAgHw provides a detailed explanation of the general principle of a coded-light camera. A complete functional decomposition of the SR-305 camera is provided in [1].

Task 3.2 Getting to know the camera (0 points)

Download `Intel RealSense Viewer` tool from Canvas to verify that your camera is working and to explore the various parameters. If you enable both the RGB and depth streams, you shall see live videos for both where the depth stream represents different depths in different colors. Hover over any pixel in the depth image and you shall see the depth value in meters at the bottom.

3.2.2 Where will the camera be mounted?

The SR-305 will be mounted overhead, on the provided assembly, at such a height that the robot arm and the objects in its workspace will lie in the field of view of the camera. How do we determine this height? You'll determine it experimentally to obtain the required field of view in

the Intel Viewer, but it can also be determined based on the camera parameters and geometric modeling, which you'll carry out in the future. Our perception pipeline will have to:

- detect the objects of interest in the camera images, e.g. find all the red cubes;
- estimate the pose (position and orientation) of the detected object in the camera frame;
- transform the pose to the robot world frame.

This is also illustrated in Figure 3.6.

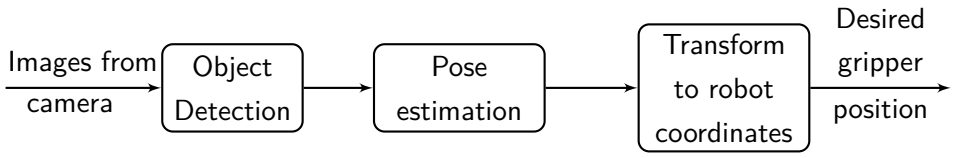


Figure 3.6: Block diagram of our vision pipeline

3.2.3 Representations

We're using cameras to create a representation of the 3D world or 3D geometry. This representation can take many forms and often we can convert between these representations, though at times the conversion is lossy. Examples of representations are triangulated surface meshes, occupancy grids/voxels, depth images, and point clouds. We'll focus on extracting information from depth images in this lab, and leave the exploration of point clouds to a subsequent lab.

Depth Images

As you have seen in the camera viewer, the SR-305 camera provides two images - one from the RGB camera and the other from the depth camera, as shown in Figure 3.7. Each pixel value

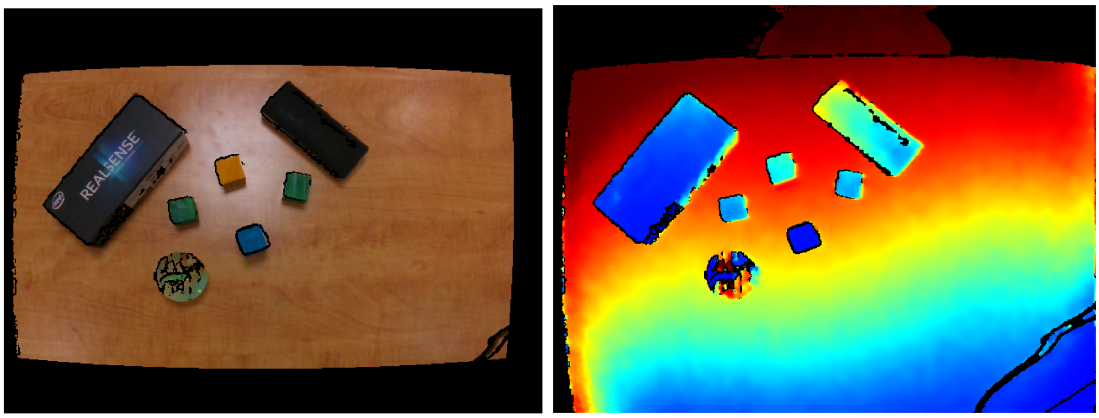


Figure 3.7: (Left) Image from RGB Camera. (Right) Depth Image

in the RGB image is typically 3-dimensional and provides information about the luminance of visible light along the pixel direction. There are different representation formats (color spaces)

for expressing this luminance information [2]. However, each pixel value in the depth image is a single number that represents the distance between the camera and nearest object along the pixel direction, as shown in Figure 3.8. Note that this distance is the normal distance from the

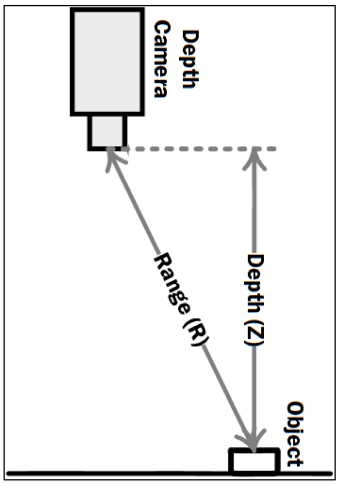


Figure 3.8: Depth value in the depth image

camera plane to that object. The different colors in the depth image in Figure 3.7 correspond to different values of depth, and are obtained by mapping the depth readings in the image to a colormap.

Noise in camera

We can notice from Figure 3.7 that the obtained images are not perfect, e.g. the depth image is showing different colors for the table when in reality the entire table is at the same depth from the camera. This is because camera sensors are not perfect, and in fact noisy. To compound things, errors in depth returns are not simple Gaussian noise, but rather are dependent on the lighting conditions, the surface normals of the object, and the visual material properties of the object, interference from other sources, among other things. The effects of these variable factors are mitigated for subsequent stages by typically adding a post-acquisition processing stage (filters), before an object detection strategy is applied, as shown in modified pipeline of Figure 3.9.

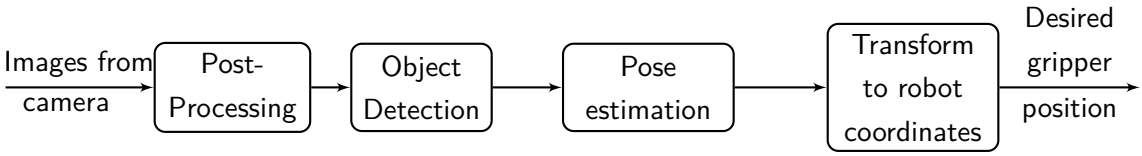


Figure 3.9: Complete flow of our vision pipeline

3.3 Extracting information from color images

Computer vision algorithms are enabling robots to tackle extremely complex environments. Our case here is comparatively easier, i.e. detecting a known object (cube) in a relatively uncluttered environment. We'll adopt the geometric approach to computer vision for determining the position of the object, as opposed to the other major approach these days, i.e. data-driven 'deep perception' approach. The geometric approach leverages the geometry of the objects, camera, and environment to assist in the vision task.

3.3.1 Image manipulation in MATLAB

The images captured by the camera will be imported in MATLAB so that objects of interest can be identified and their pose can be determined using machine vision algorithms. How are images represented and manipulated in MATLAB?

Task 3.3 Image Manipulation in MATLAB (10 points)

Complete lessons 2.1-2.4 from Module 2 of the course 'Image Processing with MATLAB' (https://matlabacademy.mathworks.com/details/image-processing-with-matlab/mlip#module=2). Your completion will be saved in the 'Progress Report', which you'll submit along with your lab findings report. A textual quick reference guide for this course is available at https://matlabacademy.mathworks.com/artifacts/quick-reference.html?course=mlip&language=en&release=R2023b. Also, the module https://matlabacademy.mathworks.com/details/image-processing-onramp/imageprocessing#module=2 can provide further help with this task.

3.3.2 Segmentation

Images contain a vast amount of data that tends to overwhelm what is significant to us in the image. *Segmentation* is the process of collecting together pixels into summary representations that emphasize some important, interesting, or distinctive properties, e.g. pixels of the same color. There are two main threads in segmentation - clustering together pixels based on local information to form regions, e.g. close by red pixels, or pixels are assembled together based on global relations, e.g. pixels believed to lie on the same line. In this section, some methods are shared from both threads.

Thresholding

Thresholding is an extremely simple idea of dividing pixels into two clusters based on a single property, e.g. the pixel intensity in a grayscale image can divide image into foreground and background. Pixels with property value greater than threshold lie in one cluster and values less than threshold in the other cluster. A single global threshold can be defined by the designer or an adaptive algorithm can be utilized. The designer can use histogram of the property of interest to determine the threshold. Figure 3.10 provides an instance of this method in use based on the MATLAB example https://www.mathworks.com/help/images/ref/imbinarize.html. The

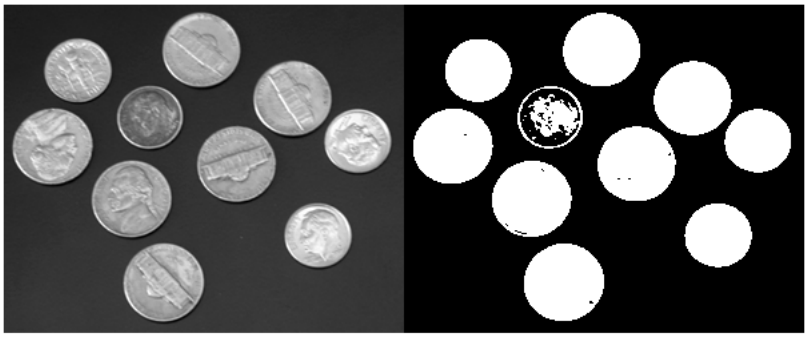


Figure 3.10: Segmentation using thresholding

method can also be applied to depth images. The MATLAB commands `imbinarize`, `imhist`, `graythresh`, `histogram`, `otsuthresh`, `rgb2gray` may be of use in applying this method.

Task 3.4 Thresholding (20 points)

Complete lesson 2.5 from Module 2 of the course 'Image Processing with MATLAB' (https://matlabacademy.mathworks.com/details/image-processing-with-matlab/mlip#module=2). Also, the module https://matlabacademy.mathworks.com/details/image-processing-onramp/imageprocessing#module=3 can provide further help with this task.

Color Segmentation

Color segmentation based on a specific color would also result in two clusters, e.g. red pixels and other colored pixels. Similar to thresholding, the image is transformed to a color space that allows better distinguishing of colors, e.g. HSV space or Lab, and then identify all pixels close to a specified color value. An example of this is provided by the MATLAB example https://www.mathworks.com/help/images/color-based-segmentation-using-the-l-a-b-color-space.html, and also illustrated in Figure 3.11.

Task 3.5 Color Segmentation (20 points)

Complete Module 4 of the course 'Image Processing with MATLAB' (`https://matlabacademy.mathworks.com/details/image-processing-with-matlab/mlip#module=4`).

K-means Clustering

K-means clustering is a beautiful algorithm that adaptively determines cluster centers and allocates each data point to a cluster based on distance from the center, resulting in auto-

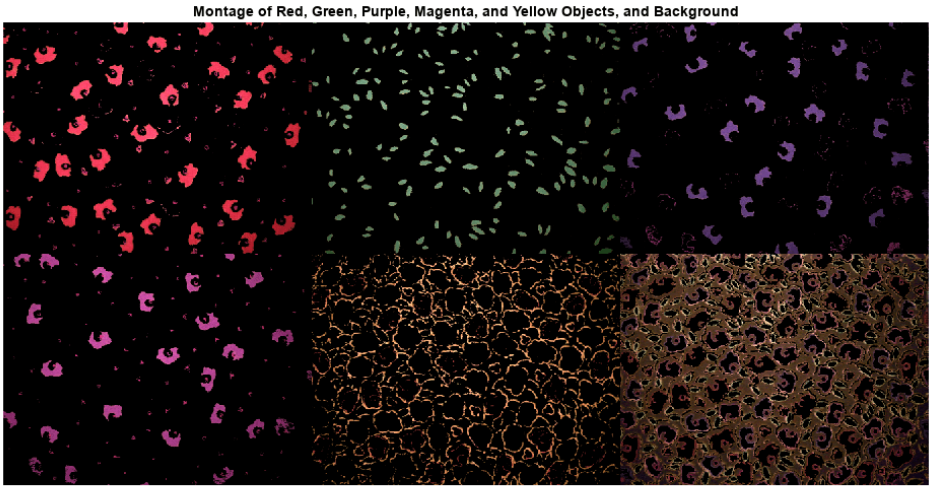


Figure 3.11: Segmentation based on color in L*a*b color space

matic clustering of all data points into possibly k clusters [3]. The method can be applied to any kind of data, and can be used for image segmentation as well. The MATLAB example https://www.mathworks.com/help/images/color-based-segmentation-using-k-means-clustering.html uses k-means clustering to automatically segment a medical image into three clusters based on color.

Connected components

Finding connected components in a binary image (pixel values are either 0 or 1) divides pixels in an image based on local connectivity, i.e. two pixels with value 1 are part of the same region if their edges touch (https://www.mathworks.com/help/images/label-and-measure-objects-in-a-binary-image.html). This is typically the second step once pixels with desired property have been identifies, e.g. mark disconnected red regions in the image to identify all the red cubes. The MATLAB functions `bwconncomp` and `label2rgb` help with this task.

Task 3.6 Connected Components (20 points)

Complete Lessons 7.1, 7.2, and 7.4 from Module 7 of the course 'Image Processing with MATLAB' (https://matlabacademy.mathworks.com/details/image-processing-with-matlab/mlip#module=7).

Fitting geometric model

A possible segmentation at a global level is to discover geometric artifacts in the images or point clouds, e.g. lines, planes, etc. Algorithms in this domain use some parametric model of the geometric figure and then classify each pixel/point in the image to be either part of this geometric figure or not. This is done based on the distance of the point/pixel from the geometric artifact. Note that the algorithms will have to determine both the values of the parameters in the geomet-

ric model, e.g. what is the equation of the line in the image?, and identify all the points in the image that constitute that geometric model, e.g. which points of the image are on that line. Two algorithms falling in this category are the Hough transform and RANSAC (random sample consensus). The MATLAB example https://www.mathworks.com/help/vision/ref/pcfitplane.html shows how planes can be detected in a point cloud using RANSAC, reproduced in Figure 3.12. The example https://www.mathworks.com/help/images/hough-transform.html shows how to detect lines in an image using Hough Transform.

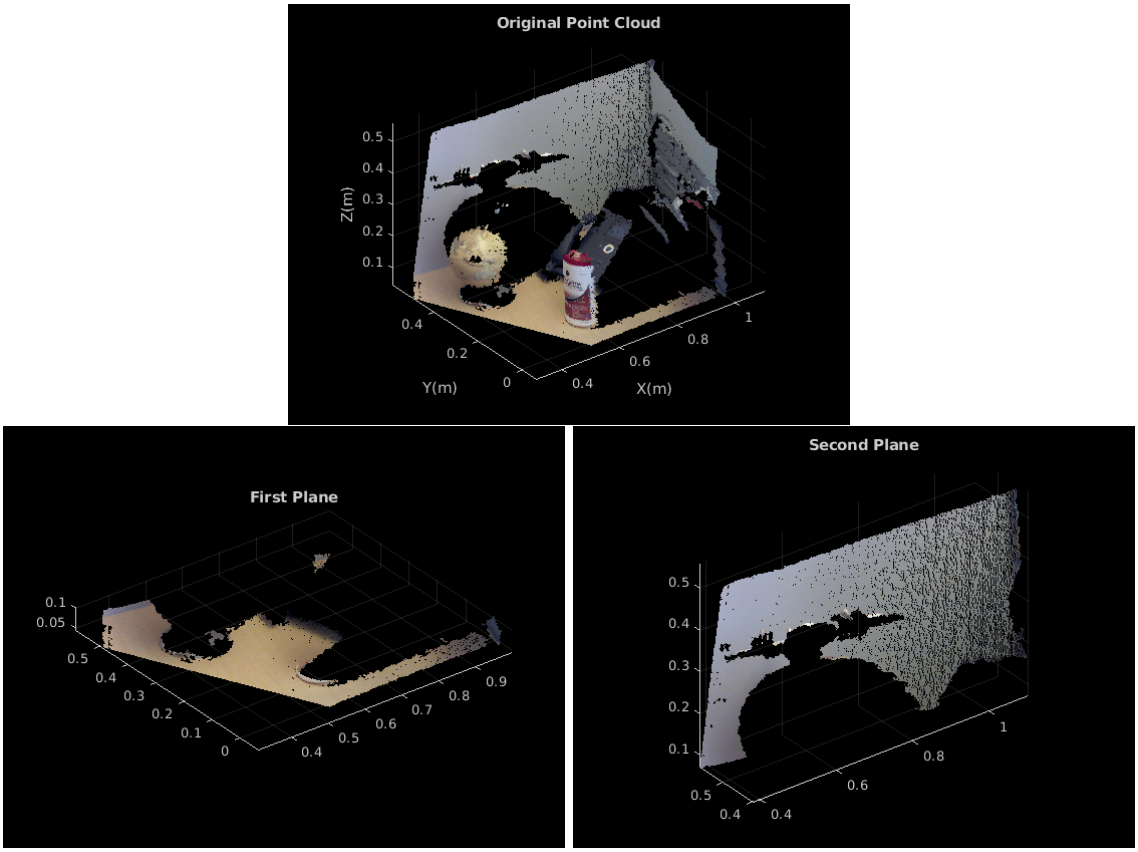


Figure 3.12: Finding planes in point cloud

Further Exploration

Modules 6, 7.3, and 8 from the same course are recommended for enhancing your algorithms based on the previous techniques.

3.3.3 Detecting the blocks

Intel provides an SDK ([4]) for its RealSense line of cameras. The SDK includes a MATLAB wrapper too and we'll be making use of it in our project.

The MATLAB code `depth_example` obtains the RGB and depth images in MATLAB and applies desired post-processing to make them suitable for object detection. You'll have to edit this file to suit your pipeline based on the post-processing parameters that you'll determine

later. All of this is done using the Intel provided SDK[1] (See [4] for more details). One of the post-processing steps is to align the two images so that their origins coincide and they're of the same size. This is why you see the black border around color image in Figure 3.7 as the native resolution of the RGB camera is higher than the IR camera, so it has to be downsampled.

Task 3.7 Find colored objects (60 points)

Set up cubes of various colors in the workspace of the robot. It can be assumed that the cubes are of a known size and the workspace is relatively uncluttered.

Tune, determine, and note down the camera parameters, presets, and filters in the RealSense Viewer that will result in an aptly exposed image and accurate depth values. Use [5] and [6] as reference. A MATLAB function `depth_example` is provided on LMS. Use it to obtain a color and a depth image of your setting[a]. The file provides an example of setting the camera parameters in code before acquisition.

Develop an object detection algorithm for determining the pixels corresponding to each object in the workspace. You are required to submit

1. a note describing your object detection strategy and the rationale behind it;
2. a list of the post-processing steps and parameters;
3. aptly commented code for your algorithm;
4. images at various steps of a good test run;
5. statistics across multiple trials, including
   - number of objects of each color in the workspace to be detected
   - number of objects of each color correctly detected
   - accuracy rate for each color.

[a] You will have to add the RealSense library to your MATLAB path. You can do this by navigating to 'Set Path' in the 'Home' ribbon on MATLAB and 'Add with subfolders' the library directory `C:/Program Files (x86)/Intel RealSense SDK 2.0/matlab/`.

The successful completion of previous task has enabled you to find relevant blobs in an image, e.g. given a mission, 'find the red blocks', you'll be able to find all pixels corresponding to each red block, or you have information at pixel level for each red block. But the mission requires you to find the red blocks in the physical world, i.e. determine the pose of each red block with respect to the camera frame. The camera frame is canonically defined, but the frame attached to the object is arbitrary; it is assigned by the system designer, you, e.g. the yellow frame

[1] You can install the SDK by using the provided installer. Make sure to check the `MATLAB Developer Package` during installation. The package will be installed to `C:/Program Files (x86)/Intel RealSense SDK 2.0/matlab/realsense/`.

assigned to the red cube in Figure 3.13.

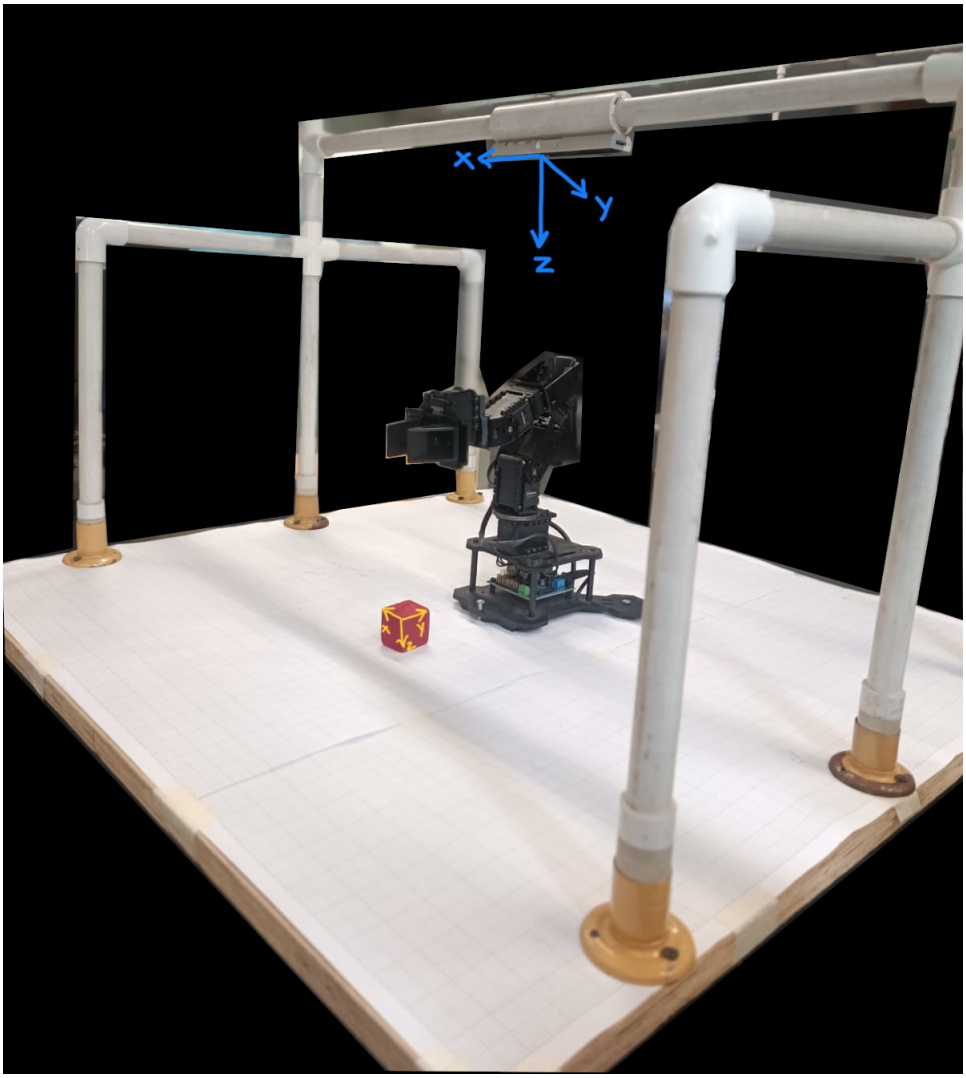


Figure 3.13: Definition of the camera frame

The canonical definition of the camera frame is illustrated in Figure 3.13. The origin of this frame is placed at the center of the lens, z-axis is perpendicular to the plane of the lens and points out of the camera, the x-axis and the y-axis are aligned with the horizontal and vertical axes of the image plane, e.g. u and v in Figure 3.14.

Using the images coordinates of all the points of the red cube, in the field of view of the camera, and (a) the depth image and (b) the known geometry of the blocks, the pose of the object is to be determined. Finding the object pose requires determining the description of the origin and the the axes of the object frame (yellow frame) in the physical world. Note that every pixel (point) (u, v) in an image corresponds to a point (x, y, z) in the real world. The depth image provides the value of z for each pixel in the image[2]. Knowing z, a unique mapping

[2]Make sure that two images, color and depth, are aligned before querying for z value as the images are acquired by different cameras. The MATLAB code `depth_example` includes an aligning step.

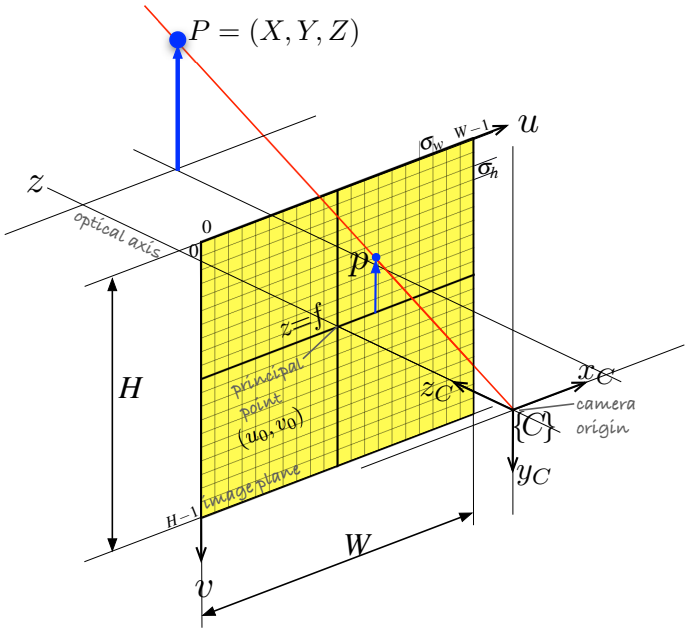


Figure 3.14: Camera Frame $\{C\}$ and the yellow Image Frame

$(u, v) \rightarrow (x, y)$ exists. So, you may assume that you can find real world coordinates of any point in the image. Then, the task becomes to determine the pixel coordinates of the relevant pixels in the image.

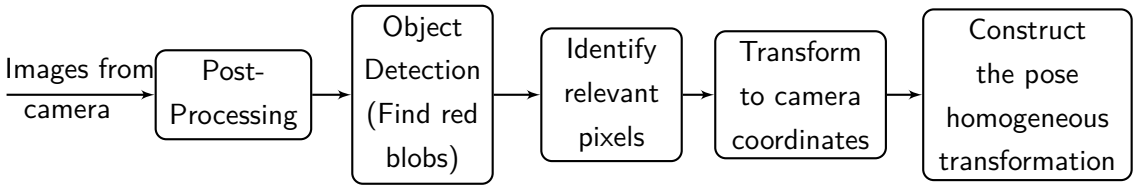


Figure 3.15: Detailed vision pipeline

Task 3.8 Determining geometric features (40 points)

Think about which geometric features could assist you in associating a pose with the object, e.g. you could detect the planes of the object, or the edges, or the corners, etc. You are required to submit

1. your assignment of a frame to an object;
2. description of an algorithm for constructing the pose homogeneour transformation (how do you determine the transformation from your identified geometric features?);
3. description of an algorithm for determining the required geometric features (determining the pixel coordinates of relevant pixels of the blob);
4. aptly commented code for the previous part;
5. images at various steps of a good test run.

References

[1] A. Zabatani, V. Surazhsky, E. Sperling, *et al.*, "Intelő realsense sr300 coded light depth camera," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 10, pp. 2333–2345, 2019.

[2] Mathworks. "Understanding color spaces and color space conversion." (), [Online]. Available: https://www.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html.

[3] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Prentice Hall, 2002.

[4] Intel. "Intel realsense sdk." (2023), [Online]. Available: https://github.com/IntelRealSense/librealsense.

[5] A. Grunnet-Jepsen, J. N. Sweetser, and J. Woodfill, "Best-known-methods for tuning intelő realsense d400 depth cameras for best performance," *Intel Corporation: Santa Clara, CA, USA*, vol. 1, 2018.

[6] Intel. "Intel realsense post-processing filters." (2021), [Online]. Available: https://github.com/IntelRealSense/librealsense/blob/master/doc/post-processing-filters.md.

Vision is the process of discovering from images what is present in the world and where it is.

– David Marr

4

CHAPTER

Perception-II

The objectives of this lab are to:

(i) understand point clouds as an alternate representation for scenes in robotics;

(ii) develop a vision pipeline for pose estimation using point clouds.

Software dependency: The utilized library functions require, at minimum:

- Intel RealSense SDK 2.0
- MATLAB Computer Vision Toolbox R2022b

4.1 Point Clouds

Recall that an RGB-D camera provides a depth image, where each pixel value is the distance between the camera and the nearest object in the environment along the pixel direction. If this information from the depth image is combined with information about the camera's intrinsic parameters, e.g. focal length (we'll study this in class), then it is possible to transform this 2D depth image representation into a collection of 3D points described in the camera frame. This collection is called a 'point cloud'. Note that each of these points has three coordinates (x, y, z) that are in fact distances of the point in the real world along the three axes

indicated in Figure 3.13. It is easy to convert this representation to the robot base frame too. A point cloud of the same scene from last chapter is shown in Figure 4.1. MAT-

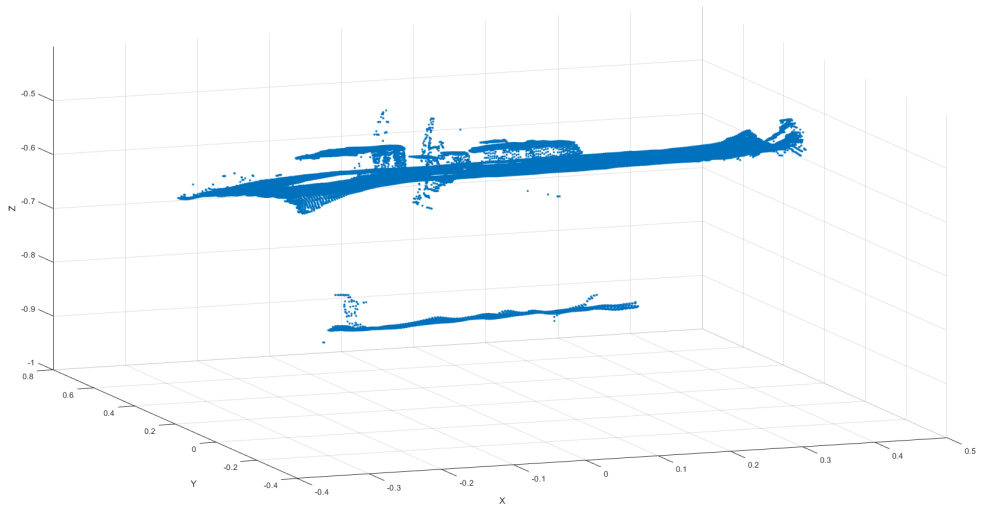


Figure 4.1: Point cloud of the same scene from Figure 3.7

LAB has a whole set of functions available for point cloud processing that can be explored at https://www.mathworks.com/help/vision/point-cloud-processing.html.

4.1.1 Getting the scene point cloud

As stated in the previous section, a point cloud is created from a depth image. Each set of pixel coordinates in the depth image, (u, v), along with the corresponding depth value, d, is used to reconstruct the original point in the physical world, (x, y, z), in the camera coordinates. This de-projection function uses the depth image and the depth camera's intrinsic parameters, which include the focal length and principal point. For more details, see [1].

Task 4.1 Explore point clouds (0 points)

The MATLAB file `pointcloud_example.m` provides code to generate a point cloud. Run the code and explore the obtained point cloud. Study the provided code.

4.2 Pose Estimation using Point Clouds

Our objective is to determine the pose of known objects in a relatively uncluttered workspace. The stages of this pipeline will be similar to the ones used with depth images, i.e. segmenting the raw point cloud of the scene to extract point cloud from an object, e.g. based on color or k-means clustering, and then estimating the pose using geometry. This section outlines two possible ways to achieve this task.

4.2.1 Fitting to Geometric Models

The objects in this lab have a well-defined geometric shape. So, a possible step towards determining the pose is to fit geometrical models, e.g. planes to the point clouds. MATLAB provides some builtin functions for fitting point clouds to geometric models. The page [2] provides a list of these functions.

4.2.2 Point cloud registration

Registration is the process of finding a transformation that takes one set of points to another, e.g. given a model point cloud describing a cube and an incoming scene point cloud, the process of point cloud registration will find the homogeneous transformation between the fixed point cloud and new point cloud, i.e. it will find the translation and rotation to achieve new point cloud from the model. These algorithms are based on ICP (Iterative Closest Point) algorithm. An example for this is `https://www.mathworks.com/help/vision/ref/pcregistericp.html`.

How to build the model point cloud?

Before using point cloud registration, we'll need a point cloud of the model to compare against. This can be done in a number of ways: (i) we can use tools that convert 3D representation into point clouds; (ii) we can create a scene with just our object, read, and save the resulting point cloud after any required manual tweaking/processing; (iii) since our object is simply a cube of known dimension, we can construct a point cloud programmatically, i.e. create a set of uniformly spaced points along the dimensions of our cube. You may be wondering how will we match the inter-point spacing with the scene point cloud to be collected later through the camera? We can match the spacing of our later scene point cloud by downsampling it using the builtin function `pcdownsample`.

Task 4.2 Pose Estimation Pipeline (100 points)

Design and develop a point cloud based vision pipeline for determining the pose of all the cubes of different color in the workspace of your robot. Assume that the workspace is relatively uncluttered and all the cubes have known uniform size. You're required to submit

1. your strategy for assigning frames to the cubes;
2. a description of any filters/processing applied to the image frames and your process for determining the corresponding parameters;
3. a description of segmentation strategy and rationale behind it;
4. a description of the pose estimation stage;
5. aptly commented code for the entire pipeline;
6. images at various steps of a good test run;
7. design of experiment(s) and results for determining the accuracy of your pipeline.

References

[1] Intel, "Projection, Texture-Mapping and Occlusion with Intel RealSense Depth Cameras," [Online]. Available: https://dev.intelrealsense.com/docs/projection-texture-mapping-and-occlusion-with-intel-realsense-depth-cameras.

[2] Mathworks. "Point cloud processing in matlab." (), [Online]. Available: https://www.mathworks.com/help/vision/point-cloud-processing.html.

Take to Kinematics. It will repay you. It is more fecund than geometry; it adds a fourth dimension to space.

– Chebyshev to Sylvester, 1873

5

CHAPTER

Forward Kinematics

The objectives of this lab are to:

(i) obtain forward kinematics mapping of the manipulator using standard DH convention and create its MATLAB function;

(ii) determine the reachable workspace of the manipulator;

(iii) create MATLAB function for determining the current pose of the manipulator.

Software dependency: The utilized library functions require, at minimum:

- Arduino IDE
- pypose
- Peter Corke's Robotics Toolbox
- MATLAB Robotic System Toolbox R2022a
- MATLAB Symbolic Math Toolbox

Kinematics refers to the geometric and time-based properties of the motion of an object without considering the forces and moments that cause the motion. In this lab, you will study

this relationship in the context of position and orientation of end-effector with respect to joint angles. This aspect of kinematics is called forward position kinematics. You'll follow the DH convention to assign frames to our manipulator, determine DH parameters, and the homogeneous transformation from the base frame to the end-effector. Having established the forward kinematics mapping, you'll verify its accuracy physically and use it to determine the workspace of the manipulator. Your forward kinematics function will be carried forward with you throughout the future labs. You'll use MATLAB to write your kinematics function as well as to control the arm.

5.1 Determination of forward kinematic mapping

Throughout this section, we'll follow the conventions and procedures outlined in [1] for the standard Denavit-Hartenberg (DH) convention.

Definition 5.1.1: Standard DH Convention

1. Links are numbered from 0 to n and joints are numbered from 1 to n.
2. z_i is chosen along the axis of rotation or translation of joint $i+1$.
3. Axis x_i is chosen along the direction of the common normal, pointing from joint $i-1$ to joint i.
   - For frame 0, the axis x_i can be arbitrarily selected.
   - If z_i is parallel to z_{i-1}, then x_i can be chosen along any of the infinite possible common normals. Typically, it is chosen along common normal passing through O_{i-1}.
   - If z_i intersects z_{i-1}, then axis x_i is chosen normal to the plane formed by z_i and z_{i-1}.
4. The origin, O_i, of frame i is chosen at the intersection of axis z_i and axis x_i.
   - If z_i intersects z_{i-1}, then O_i is at the intersection of z_i and z_{i-1}
5. Axis y_i is chosen to form a right-handed frame.
6. The end-effector frame, n, can be arbitrarily chosen.

Task 5.1 DH Frame Assignment (15 points)

Using standard DH convention, assign DH frames to the robot arm in Figure 5.1. Make sure to clearly indicate the z and x axes, and the origin of each frame; drawing the y axis is optional. Place the origin of the end-effector frame at the center of gripper motor horn, for convenience of measurements in upcoming tasks. Draw and paste each frame's z and x-axis on the motor or link bodies of the robot. This will help your visualization in

later tasks.

Definition 5.1.2: DH Parameters

The four DH parameters are:

- **Link Offset ($\mathbf{d_i}$):** Distance from the origin O_{i-1} to the intersection of x_i with z_{i-1}, measured along z_{i-1};
- **Joint Angle ($\boldsymbol{\theta_i}$)**: Angle from x_{i-1} to x_i, measured about z_{i-1};
- **Link Length ($\mathbf{a_i}$)**: Distance between axes z_{i-1} and z_i, measured along the axis x_i;
- **Link Twist ($\boldsymbol{\alpha_i}$)**: Angle from z_{i-1} to z_i, measured about x_i.

Task 5.2 DH Parameters (15 points)

Annotate Figure 5.1 with DH parameters based on your frame assignments, complete Table 5.1, and explain your process for determining the parameters where needed. You'll have to physically measure the values of some parameters.

Link	$\mathbf{a_i}$	$\boldsymbol{\alpha_i}$	$\mathbf{d_i}$	$\boldsymbol{\theta_i}$
1				
2				
3				
4				

Table 5.1: Table of DH parameters

Definition 5.1.3: Homogeneous Transformation using DH parameters

The homogeneous transformation, A_i, given the DH parameters for link i is determined as:

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 5.1: DH Frame assignment

Task 5.3 Homogeneous Transformations (5+5+3 points)

Use MATLAB's symbolic math toolbox to determine the intermediate homogeneous transformations ${}^0T_1$, ${}^1T_2$, ${}^2T_3$, ${}^3T_4$, and the resultant transformation ${}^0T_4$.

(a) Write a MATLAB script to create symbolic matrices for all the homogeneous transformations listed above. Note that one of the parameters will be a joint variable.

- You can create a symbolic variable in MATLAB using `syms` function, e.g. `syms('theta_1')` will create a symbolic variable θ_1 in MATLAB. In case of a live script, the variable will also be displayed in Greek alphabet.
- The MATLAB functions `cos` and `sin` expect arguments in radians, while `cosd` and `sind` in degrees.
- Using the standard naming convention for the variables storing homogeneous transformations may result in convenience later, e.g. `T01` or `T_01`.

(b) Obtain ${}^0T_4$ by multiplying the previously determined homogeneous transformations in the appropriate order. The MATLAB functions `simplify` and `expand` may be of help in simplifying the final expressions.

(c) Provide expressions for the position and orientation of the end-effector frame with respect to the base frame.

A remark about describing the orientation: The rotation matrix provides us the orientation of the end-effector frame with respect to the base frame, but it may be difficult for us to verify its correctness quickly. However, notice that the geometry of this manipulator is such that the gripper orientation can be described by the pair (θ_1, ϕ), where θ_1 is angle of joint 1 and tells us the radial direction in which the arm is pointed and ϕ is the angle the approach direction of gripper makes with the x-axis of frame 1. How could you find ϕ?

5.1.1 Building the functions in MATLAB

Task 5.4 FK Function (10 points)

Provide a MATLAB `function [x,y,z,R] = pincherFK(jointAngles)` or `function [x,y,z,R,theta,phi] = pincherFK(jointAngles)` that accepts joint angles of Phantom X Pincher and returns the end-effector position and orientation in the specified order. Make sure to add comments describing the arguments and corresponding units.

The choice between the provided function definitions depends on which strategy you want to adopt for describing orientation, as outlined in the previous remark. You can also decide whether your function will accept arguments in degrees or radians. You can find help on how to create MATLAB functions at [2].

Using the DH parameters, we can build a model of our manipulator in MATLAB as well. MATLAB treats robots in exactly the same way as we have done in class, i.e. a chain of joints and rigid bodies [3]. The provided MATLAB script file `pincherModel.m` builds a model for Phantom X Pincher.

Task 5.5 Verification of Forward Kinematic Mapping (5 points)

Enter your DH parameters from the previous task in `pincherModel.m`. The file should display a skeleton of the robot with frames. If you set your desired configuration, i.e. joint angles as the value of the `configNow` variable at the bottom of the file, the script returns the end-effector position and orientation with respect to the base frame, and displays the configuration graphically.

Select 4-5 random configurations for the manipulator and share the end-effector position and orientation, as determined by the provided `pincherModel` and your own `pincherFK` function. Make sure that they match. MATLAB command `randomConfiguration(robot)` can also generate a random configuration for `robot` in MATLAB workspace.

5.2 Identifying reachable workspace

If the forward kinematic mapping is f, then the reachable workspace is $f(\mathcal{D})$ where $\mathcal{D}$ is the joint space, which is the product space of the ranges of all four joint angles, (θ_1, θ_2, θ_3, θ_4), of our arm, e.g. if all joint angles lie in $[-150^\circ, 150^\circ]$, then the product space is

$$\mathcal{D} = [-150^\circ, 150^\circ] \times [-150^\circ, 150^\circ] \times [-150^\circ, 150^\circ] \times [-150^\circ, 150^\circ].$$

This is a difficult task to carry out geometrically, but we can use our MATLAB FK function to run a Monte Carlo simulation to get a sense of the reachable workspace of our manipulator.

Depending on our computational resources we can either uniformly sample or randomly sample our joint space to obtain a set of configurations in the joint space. The end-effector position corresponding to each joint configuration is obtained using FK function, and the resulting end-effector positions are all plotted on the same plot. We can get a reasonable representation of the reachable workspace by this plot, if the sampling is dense.

In determining the workspace, we can make use of the joint limits for Dynamixel AX-12A motors. According to the motor specifications, each motor angle lies in $[-150^\circ, 150^\circ]$. But, this may not be the range for θ_i as your choice of axes during DH frame assignment may not align with manufacturer's choice for the definition of joint angles, e.g. the manufacturer may have chosen positive joint axis as coming out of the motor while you may have chosen it to go inside the motor, resulting in opposite directions for positive joint angle.

Joint ID	DH Joint Angle (θ_i)	Servo Angle ψ_i	Aligned directions of rotation (Yes/No)
1	0°		
2	0°		
3	0°		
4	0°		

Table 5.2: Linear mapping between servo angles and DH angles

Joint ID	Minimum Joint Angle		Maximum Joint Angle	
	Servo angle	DH Joint Angle	Servo Angle	DH Joint Angle
1	-150°		150°	
2	-150°		150°	
3	-150°		150°	
4	-150°		150°	

Table 5.3: Joint Limits

Task 5.6 DH and Servo Joint Angles alignment (10 points)

Map the DH joint angles to the respective servomotor angles in Table 5.2 and Table 5.3. You'll have to determine (i) the possible angular shift between 0° of each DH joint angle (see the definition of joint angle in DH parameters) and the joint position when 0° command is sent to the corresponding servomotor, and (ii) whether the positive directions of rotation in the two cases are aligned. The determined shifts can be used to determine transform motor joint limits to DH specifications in Table 5.3.

Task 5.7 Mapping servo angles to DH angles (5 points)

Provide a MATLAB `function dhJointAngles = servo2dh(jointAngles)` that accepts joint angles of Phantom X Pincher, as understood by the servomotors, and convert them to your corresponding DH-assignment based joint angles. The function should be properly commented.

- The `jointAngles` vector contains joint angles, received from motor encoders, in order from the base to the wrist. The angles should either be in radians or angles.
- You need to find out the appropriate mapping function based on Table 5.2.

Task 5.8 Identifying reachable workspace (10 points)

Use the outlined idea of determining end-effector positions for selected joint configurations (uniform or random) to plot the reachable workspace of our Phantom X Pincher robot arm. Provide an isometric view of the workspace as well as a top-view, i.e. a projection of your workspace onto $X-Y$ plane of your base frame. Remember to mark axes in your plots. What is the maximum horizontal reach according to your identified workspace?

- The MATLAB function `rand` generates uniform pseudorandom numbers in $[0, 1]$. We can generate N samples for a joint angle θ_i, with lower bound θ_i^{min} and upper bound θ_i^{max} using the expression:

$$\theta_i = \theta_i^{\text{min}} + \left(\theta_i^{\text{max}} - \theta_i^{\text{min}}\right) \times \texttt{rand(N,1)}.$$

- The MATLAB functions `linspace`, `ndgrid`, and `scatter3` may be of help for this task.

5.3 Interacting with the arm

We'll now control the robot from MATLAB and physically verify that our forward kinematics computations are correct. For this, we'll make use of the `Arbotix` library written by Peter Corke as part of his Robotics toolbox [4]. Instructions for setting up the toolbox are provided in Appendix-B. Follow the steps below to get the arm set up to communicate with MATLAB.

5.3.1 Setting up communication between MATLAB and arm

1. From the `Arduino IDE`, upload the sketch `pypose` on the Arbotix-M.
2. Connect the arm to your computer and open MATLAB.
3. Identify the COM port to which the arm is connected by investigating the list of open ports at `Control Panel> Device Manager> Ports (COM LPT)`. For the next steps it is assumed that COM5 has been identified.
4. Enter the following on the MATLAB command prompt:

```
>> arb = Arbotix('port', 'COM5', 'nservos', 5)
```

If the connection is successful then a variable `arb` will be created, connecting to the robot and the following message will be displayed.

```
arb = Arbotix chain on serPort COM5 (open)
5 servos in chain
```

5.3.2 Library of Arbotix Functions

Almost all capabilities provided by the manufacturer in Dynamixel AX-12A library have been captured in this MATLAB library. Following is a collection of methods available in the library, which are of use to us throughout this class.

- `arb.gettemp(id)` returns the temperature of the servomotor `id`, or the temperature of all motors if no argument is provided, i.e. `arb.gettemp`.
- `arb.getpos(id)` returns the angle of the servomotor `id` in **radians**, or the angular positions of all motors if no argument is provided. **Remember that the range of motion of motors on the arm is constrained to** $[-150°, 150°]$**.**
- `arb.setpos(id,pos,speed)` sets the goal position of the servomotor `id` to `pos` **radians**. The motor will then start moving to the goal position.

  The argument `speed` is optional, and accepts values between 0 and 1023. 0 means the motor does not control its speed and uses maximum possible speed. Each unit of speed is about 0.111 rpm.

  `arb.setpos(pos,speed)` sets the positions and speeds of servos 1-N to corresponding elements in the vectors `pos` and `speed` respectively.
- `arb.relax(id, status)` causes the servo `id` to enter zero-torque (relaxed) state if `status` argument is missing or `TRUE`. If the `status` is `FALSE`, then the servo starts providing torque. To relax all motors `arb.relax()` or `arb.relax([])` can be used, and the relaxed mode can be ended with `arb.relax([],FALSE)`.

It is also worthwhile to know that it takes between $180 - 760\mu$s to read/write a single parameter to an AX-12A motor [5]. So, it could take up to 3ms for a read/write operation to all motors to be completed. In short, they are slow to process instructions.

5.3.3 Controlling the arm from MATLAB

You will now write some helper functions that will assist us in controlling the physical arm in the future.

Task 5.9 Communicating with motors (7 points)

Provide a MATLAB `function [x,y,z,R] = findPincher()` or `function [x,y,z,R,theta,phi] = findPincher()` that queries the current servo angles from Phantom X Pincher motor encoders and returns the current end-effector position and orientation in the specified order. The function should be properly commented.

- Physically measure and note the end-effector position and orientation. How does it

compare to the pose returned by your function?

- Do you think the pose returned by this process will ever be accurate? If not, what do you think are the sources of error?

Task 5.10 Mapping DH angles to servo angles (10 points)

Write a helper MATLAB
`function [servoJointAngles,errorCode] = dh2servo(jointAngles)` that accepts DH joint angles of Phantom X Pincher and convert them to corresponding servomotor angles. The function should be properly commented.

- The `jointAngles` vector contains joint angles, according to DH frame assignment, in order from the base to the wrist. The angles should either be in radians or angles.
- The angle limits for the motors are $[-150^\circ, 150^\circ]$, and your function should return an empty array for `servoJointAngles` and an appropriate error code if the provided joint angles map to angles outside this limit.
- You will have to map the computed servo angle θ_i to its corresponding value in $[-\pi, \pi]$ to compare against the allowed joint limits. One way to find the equivalent value of θ in the interval $[-\pi, \pi]$ is `angle = mod(theta+`π`,2`π`)-`π.
- You need to find out the appropriate mapping function, based on Table 5.2.
- You have freedom to choose complexity of the error reporting system. It could be as simple as `errorCode=0` if all angles are within limits, and `errorCode=1`, if they're not.

References

[1] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. John Wiley & Sons, 2020.

[2] Mathworks. "Functions in matlab." (2023), [Online]. Available: https://www.mathworks.com/help/matlab/functions.html.

[3] Mathworks. "Rigid body tree robot model in matlab." (2023), [Online]. Available: https://www.mathworks.com/help/robotics/ug/rigid-body-tree-robot-model.html.

[4] P. Corke. "Robotics toolbox." (2017), [Online]. Available: https://petercorke.com/toolboxes/robotics-toolbox/.

[5] P. Corke. "Dynamixel ax12a servos." (2019), [Online]. Available: https://petercorke.com/robotics/dynamixel-ax12a-servos/.

Take to Kinematics. It will repay you. It is more fecund than geometry; it adds a fourth dimension to space.

– Chebyshev to Sylvester, 1873

6

CHAPTER

Inverse Kinematics

The objectives of this lab are to:

(i) obtain closed-form expressions for the inverse kinematics mapping of the manipulator and create its MATLAB function

Software dependency: The utilized library functions require, at minimum:

- Arduino IDE
- pypose
- Peter Corke's Robotics Toolbox
- MATLAB Robotic System Toolbox R2022a
- MATLAB Symbolic Math Toolbox

The purpose of this lab is to derive and implement a solution to the inverse kinematics problem for the Phantom X Pincher. Recall that the objective of the inverse kinematics mapping is to determine the joint coordinates, given the end-effector position and orientation. This is an absolutely vital step for our pick and place operation. In general, the existence and uniqueness of solution is not certain in inverse kinematics problems. You will encounter this ambiguity

in your computations today, and you must choose one solution (based on motor angle limits, continuity, shortest route, obstacle avoidance, etc.). The inverse kinematics problem could be solved using numerical approach or by obtaining closed-form expression. We will obtain closed-form expressions as they are better for fast and efficient real-time control.

6.1 Finding all IK solutions

The fundamental IK problem is to find values for each of the joint variables, given a desired position, (x, y, z) of the end-effector and a desired orientation, specified in terms of a 3×3 rotation matrix. However, the 4 DoF manipulator in our lab is incapable of achieving arbitrary spatial positions and orientations of the end-effector. We can specify the desired position of the end-effector, but with the available only one orientation degree of freedom, it doesn't make sense to specify a complete rotation matrix. So, how do we leverage this available orientation degree of freedom?

One way is to fix the orientation of the gripper jaws, e.g. we set the gripper jaws to always point straight downwards, or the jaws are always horizontal. Analytically, this will correspond to fixing some entries of the rotation matrix, e.g., if the gripper is to always point downwards and the $\hat{x}$-axis of the end-effector frame is assigned along the length of the last link, then we need to set the first column of rotation matrix as $(0, 0, -1)^T$. Another possible way is to allow the user to specify an angle ϕ, which is the angle made by gripper with either vertical axis of the world frame or horizontal axis of frame 1.

Task 6.1 Inverse Kinematics Solutions (60 points)

Given a desired position, (x, y, z), of the end-effector and orientation, ϕ, find mathematical expressions for all solutions to this inverse kinematics problem. Show all steps and specifically state how many solutions exist? Assuming that direction of $\hat{x}$ of the last frame or the end-effector frame is along the length of the last link, ϕ is the angle it makes with the x-axis of frame 1, i.e. $\phi = \theta_2 + \theta_3 + \theta_4$. When the gripper is parallel to the base board, then $\phi = 0^{\circ}$[a].

[a] See the remarks below for further explanation

In the problems discussed in class, the manipulators have a spherical wrist allowing for a tidy decoupling of the inverse position and inverse orientation problems. Unfortunately, we have less than 6 DoF and no spherical wrist here. We will solve for the joint variables in the order that may not be immediately obvious, but is a consequence of the construction of the robot. At each step, you can either use the algebraic or geometric method to find expressions for that joint variable. For the algebraic method, you can make use of your Forward Kinematics expressions. A sketch for deriving the IK expressions is provided in Section 6.3.

Task 6.2 Inverse Kinematics MATLAB function (10 points)

Say there are N possible solutions to the IK problem of our manipulator, in general. Write a MATLAB function `findJointAngles(x,y,z,phi)`, which accepts the position and orientation of end-effector as arguments and returns an $N \times 4$ matrix containing all the IK solutions. Row i of this matrix corresponds to solution i, and column j of the matrix contains the values for θ_j.

Select few points (x, y, z, ϕ) in the workspace of the robot and find all the IK solutions, as determined by your `findJointAngles` function. Verify the correctness of the determined solutions by plugging them into the created Forward Kinematics function from the last chapter.

6.2 Choosing an IK solution

As we have realized, there are multiple solutions to the inverse kinematics problem, in general. What should be our criterion to select one solution out of all possible solutions? The choice of a solution depends on the relevant factors at that instant. Discuss with your lab mates possible strategies for choosing an IK solution, given (x, y, z, ϕ) and the current joint angles.

Task 6.3 Optimal Solution (20 points)

Write a MATLAB function `findOptimalSolution(x,y,z,phi,currentConfig)`, which accepts the desired position, desired orientation, and the current joint angles as arguments and returns a vector `[theta1,theta2,theta3,theta4]` corresponding to the optimal and realizable inverse kinematics solution. A realizable solution is within the joint limits of the servos. An optimal solution, in our sense, is the IK solution closest to the current configuration of the robot, i.e. minimize $b_1|\Delta\theta_1| + b_2|\Delta\theta_2| + b_3|\Delta\theta_3| + b_4|\Delta\theta_4|$. You can choose $b_i = 1$. (See below)

To complete this task, you may have to write helper function `checkJointLimits` to make sure that a solution is realizable. Recall that the motor limits are in the interval $[-150°, 150°]$ and you can use your previous helper function `dh2servo` to convert the DH angles obtained from IK solutions to servo angles. You may also need a helper function `getCurrentPose` to get the current configuration of the robot. This can make use of the MATLAB Arbotix methods. In minimizing the objective function, remember that angles wrap around, i.e. $\psi + 2n\pi = \psi$, when determining the angular difference. You can use `mod(angle+`π`,`2π`)-`π to rewrite an angle in the interval $[-150°, 150°]$ before computing the angular difference.

The weights b_i can be chosen non-uniformly to penalize change in some angles more than others, e.g. $b_1 > b_j$ for $j \in \{2, 3, 4\}$ causes the positioning to be realized by moving the small joints as opposed to large joint 1, if possible.

Task 6.4 Controlling the arm from MATLAB (10 points)

Provide a MATLAB `function errorCode = setPosition(jointAngles)` that accepts joint angles of Phantom X Pincher as argument, and sets them as goal positions for the respective motors in the arm. The function should be properly commented, especially the error codes should be explained in detail.

- The `jointAngles` vector contains joint angles, according to DH frame assignment, in order from the base to the wrist. The angles should either be in radians or angles.
- Remember that the library method `arb.setpos` expects angles in radians.
- The angle limits for the motors are $[-150°, 150°]$, and your function should output an error and stop execution, if a provided joint angle is outside this limit. You can make use of `dh2servo` function created in the previous chapter.
- You have freedom to choose complexity of the error reporting system. It could be as simple as `errorCode=0` if the instructions are being executed by the motors, and `errorCode=1`, if they're not.

Function Test: Test your function by executing some joint angles tuples returned by `findOptimalSolution` function. Before passing any joint angles to `setPosition` function, make sure that they'll not result in a self-collision by first simulating it in `pincherModel.m`

6.3 Sketch for deriving IK expressions

6.3.1 Geometric Approach

The expressions provided in these steps are with respect to our frame assignments and your expressions may be different.

1. Write down the expressions for x, y, and z of the end-effector from your forward kinematics mapping.
2. Solve for θ_1, which is dependent on desired position only.

$$\theta_1 = \arctan 2(y, x)$$

3. Are there any other solutions for θ_1?
4. Find the coordinates of the wrist center (r', s'), i.e. the origin of frame 3.

$$(\bar{r}, \bar{s}) = (r - a_4 \cos\phi, s - a_4 \sin\phi)\,,$$

where $r = \sqrt{x^2 + y^2}$ and $s = z - d_1$.

5. Solve for θ_3 and θ_2, which are dependent on the wrist center.

$$\cos\theta_3 = \frac{\bar{r}^2 + \bar{s}^2 - a_2^2 - a_3^2}{2a_2a_3}$$
$$\theta_2 = \arctan 2(\bar{s}, \bar{r}) - \arctan 2(a_3 \sin\theta_3, a_2 + a_3\cos\theta_3)$$

6. Are there any other solutions for (θ_2, θ_3)?

7. Solve for θ_4, which is dependent on the desired orientation and joint angles θ_2 and θ_3

CHAPTER 7

A complete motion control system

It is time to put together everything you have built till now to have a functional motion control system. The goal of this lab is to build an accurate pick and place system. The system will receive two locations - pick location (x_1, y_1, z_1) and a place location (x_2, y_2, z_2) from the user. Having received the locations, it will move the arm from its present location to (x_1, y_1, z_1), pick a known object from that location, move to new location (x_2, y_2, z_2), and place it there.

This will not be a completely directed activity and the manual will leave some details unspecified for you to figure out your own. But, your competency and all the elements required for this system have been built in the previous labs and you have to maximally utilize your previous learning, observations, mathematical expressions, and code. Before you get to completing the above system, you'll be directed to build some sub-systems you'll require later.

7.1 Our complete motion control system

We need a way to model our complete system and a finite state machines (FSM) or deterministic finite automata is one such way. You can read up on FSMs online to refresh your understanding of the topic.

Task 7.1 FSM (50 points)

Draw a state-transition diagram of an FSM corresponding to the following scenario:

- System is in idle state till it receives pick location, (x_1, y_1, z_1, ϕ_1) and place location, (x_2, y_2, z_2, ϕ_2).
- Geometry of the object to be picked and placed, including its orientation, is known before hand.
- The locations can be assumed to lie in the interior of the manipulator's workspace, and the object is in an orientation so that it can be picked.
- System should verify the final placement location, before determining that the task has concluded.
- Smooth motion and accurate placement[a] is desirable.

[a] You'll have to plan your gripper picking and releasing strategy, considering the accuracy of your system, determined in earlier labs.

Task 7.2 FSM Implementation (50 points)

Implement the system described by the previous FSM in MATLAB[a] for Phantom X Pincher and the cube object. In addition to your implementation code, submit an explanation of your strategy, especially functions that were not developed previously, a video of your best execution, and identify and comment on points of improvement.

[a] You can implement the FSM using usual text-based programming, or Stateflow, a graphical programming environment for implementing FSMs in MATLAB. To learn further about Staeflow, see https://www.mathworks.com/help/stateflow/gs/finite-state-machines.html.

CHAPTER A

Setting up Arbotix-M Software

ArbotiX Getting Started Guide / Arduino IDE 1.6.X Setup

The Arbotix 1.6 Hardware/Library files are currently in public beta. Please contact us with any problems or submit a Github Issue

This guide will show you how to install the Arduino IDE (Integrated Developer Environment) on your computer and then use the Arduino IDE to program your ArbotiX-M. Completing this g critical to being successful with your InterbotiX kit.

The ArbotiX-M is an Arduino-Compatible micrcocontroller that is compatible with the Arduino IDE and any standard Arduino code. The ArbotiX-M has built in support for 3-Pin DYNAMIXE

This guide covers Arduino IDE version 1.6.X+. If you are looking for a guide covering the older version 1.0.6 Arduino IDE, you can find it here.

Contents:

Step 1: Setting up the Arduino Software

Click on your operating system to expand instructions to guide you through the getting started process.

Click here to expand Windows Instruction Click here to expand Mac OS X Instruction Click here to expand Linux Instruction

Windows

Before you can build and use your InterbotiX Kit, you will need to be able to load programs onto your ArbotiX-M board. These programs are called 'sketches'. To load these sketches onto yo you will use the Arduino Integrated Developer Environment (IDE). To install the Arduino IDE do the following:

- Download the Arduino IDE 1.6.X (This guide was last tested with **Arduino IDE Version 1.6.10**.)

  1. Click on the download link for Windows Installer
  2. This will bring you to a page asking you to support the Arduino Software. The Arduino IDE is Free and Open Source, so it is not necessary to pay for it. If you'd like to give a contr further Arduino development, you can click Contribute and Download, otherwise, click Just Download.
  3. Navigate to your download folder and open the file you just downloaded.
- Install the Arduino IDE on your computer, following the prompts during installation
- Once installed, run the Arduino IDE to create the Arduino folders
- Close the Arduino IDE to get ready for the Tools and Library installation

If you have any problems with this setup, try downloading the zip file instead of the installer, and following the directions from there. The official Arduino website also has a g "Getting Started with Arduino" guide for Windows. If you are using a ArbotiX-M you will select "ArbotiX Std" when you select your board.

Back t

Step 2: Installing the FTDI drivers

Click on your operating system to expand instructions to guide you through the getting started process.

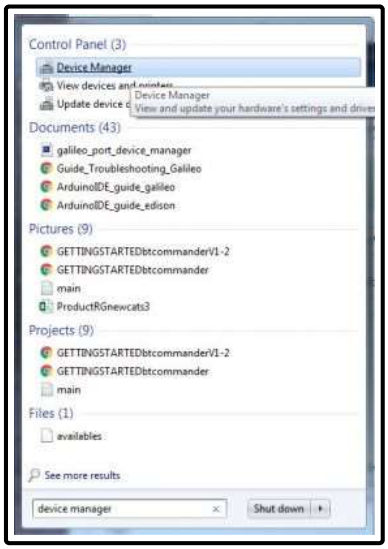
Click here to expand Windows Instruction Click here to expand Mac OS X Instruction Click here to expand Linux Instruction

Windows

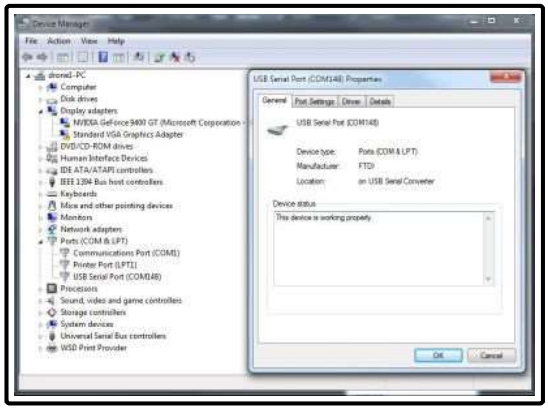
Now you will need to install FTDI drivers. These drivers will allow you to communicate with your ArbotiX-M via the USB port. Some modern Operating Systems either have these drivers or ca automatically find them. To install these drivers:

1. Plug your ArbotiX-M into your PC via USB
2. The device driver should start installing automatically. If you click on the notification in the taskbar, the Driver Software Installation window will pop up, showing you that it is installing t drivers, a USB Serial Converter, and a USB Serial Port. Next to the USB Serial Port in parenthesis, it shows you the COM port number.

3. It should install automatically, once both devices have the green check mark by them, you can check the installation by going to your file manager. To get to the device manager, press Windows key on your keyboard, and start typing "Device Manager"



4. In the list, you should see Device Manager. Click on it.



5. Now that you're in the device manager, click on ports, and the USB Serial Port should be listed with the COM port number we saw earlier. If you're not sure that it's the right one, unplu ArbotiX-M from USB and it should disappear.

If Windows did not automatically install the drivers or you are unsure, you can find the FTDI drivers here, and a guide to installing them here. You will be installing the **VCP drivers** onto your *You do not need to install the D2XX drivers mentioned in the guide*. You will need to restart your computer after installing the drivers.

Note: Windows users can download the the drivers and install them through the windows hardware wizard, or click on the 'setup executable' link automate the process. If you having problems, please see the FTDI Driver Guide for Windows on this page

Back t

Step 3: Setting up the InterbotiX Tools and Libraries

Click on your operating system to expand instructions to guide you through the getting started process.

Click here to expand Windows Instruction Click here to expand Mac OS X Instruction Click here to expand Linux Instruction

Windows

INTERBOTIX TOOLS AND LIBRARIES

The InterbotiX Tools and Libraries Download offers a variety of sketches and libraries for working with InterbotiX Robot Kits. First download the Tools and Libraries ZIP file. In this .zip file, the two folders

- **libraries**-this folder contains libraries that will add functionality to your Arduino.
- **hardware**-this folder contains the hardware definitions that the Arduino IDE needs to communicate with the ArbotiX-M

To install these files you will move these 2 folders into your 'Arduino' user folder. This is **NOT** the folder where the Arduino IDE itself is located. The location of this folder will be different base operating system.

If you're having trouble finding your 'Arduino' folder, open the Arduino IDE and open the 'Preferences' panel (File->Preferences). Here you will find a file path under 'Sketchbo location:'. This is the path to your 'Arduino' folder.

Windows XP

```
My Documents\Arduino\
```

Windows Vista/7

```
Documents\Arduino\
```

If you already have a `libraries` folder, simply copy the contents of the InterbotiX `libraries` folder into the `libraries` folder in your `Arduino` folder.Your folder structure s look like the one shown above, along with your pre-installed files.

When you are done, your file path should look like this. (Click on folders to expand them)

To check if installation was successful, open the Arduino IDE again and open

```
File -> Examples -> ArbotiX -> libraryTest
```

If you do not see `ArbotiX` under `Examples` then you have not installed the files correctly.

Once you have the `libraryTest` sketch open, click on the 'Verify' Button ✔ (the green check in the upper left). This will attempt to compile the sketch. If all of the software is installed prope see a 'Done Compiling' below the editor. If you get any errors, the library files have not been placed properly.

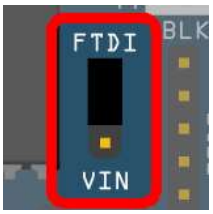
Back t

Step 4 : Connecting The Arbotix Robocontroller to your Computer

To program the ArbotiX-M Robocontroller from your computer you will need an Serial-to-USB device. We've tested the ArbotiX-M with a variety of FTDI USB-Serial converters, though it s work with other adaptors(prolific, SIlabs CH30, etc). We recommend either the UartSBee or the FTDI-USB Cable.

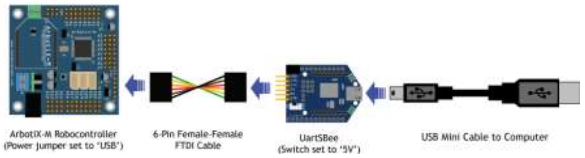
- In this example we will power the ArbotiX-M from the FTDI port. Move the power jumper so that it connects the middle pin and the 'USB' pin ('FTDI' for the original ArbotiX).

Power Jumper for ArbotiX-M

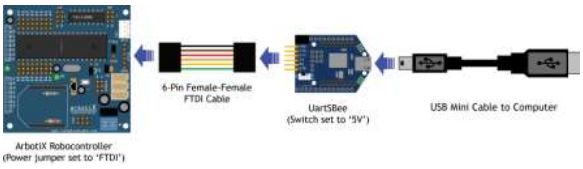


Power Jumper for ArbotiX

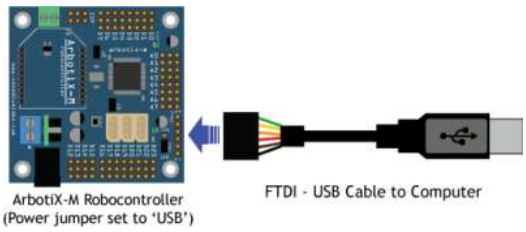
- The orientation of the FTDI cable is very important - it is possible to plug in the cable backwards and you will not be able to program your board. The top FTDI pin with the mark 'BLK' always connect to the black FTDI cable. This pin is a ground or 'GND' pin. The bottom pin with the mark 'GRN' will always connect to the Green FTDI cable. Click here to learn mor powering the ArbotiX-M
- If you are using the UartSBee, the 'BLK' and 'GRN' marking are on the underside of the board.
- If you are using a UartSBee, make sure that the switch is set to '5v' so that the unit is running at 5v like the ArbotiX-M
- Click here to learn about more options to program the ArbotiX-M



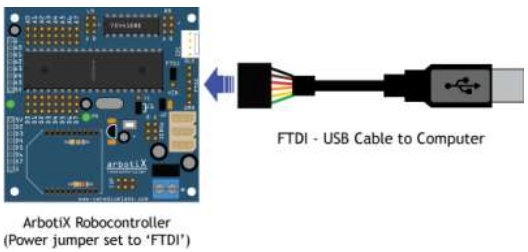
ArbotiX-M to UartSBee Connection



ArbotiX to UartSBee Connection



ArbotiX-M to FTDI-USB Connection



ArbotiX to FTDI-USB Connection

You cannot program the ArbotiX while an XBee is plugged into the ArbotiX or the UartSBee. You must unplug any XBees from the ArbotiX-M or UartSBee while programm This is because the XBee and the FTDI cable are connected to the same serial port. If you wish to program the board while an XBee is plugges in, you must use ISP progra .

Back t

Step 5: Program the ArbotiX-M Robocontroller to Blink

Now that your ArbotiX-M is hooked up to your computer, you will need to pick the `ArbotiX` board from the boards menu. Select the proper board:

```
Tools->Board ->ArbotiX Std
```

Now pick the serial port. Go to

```
Tools ->Serial Port
```

and pick the serial port for the FTDI device.

- On Windows, the serial port will be the text `COM` followed by a number, like `COM3`
- On Mac, the serial port will be the text `/dev/cu.usbserial` followed by a random number, like `/dev/cu.usbserial-AL4223`
- If you have multiple serial ports and you are not sure which one is the ArbotiX-M, unplug the FTDI device from the computer, then re-open the Serial Port menu. The serial port tha disappeared is the serial port with the ArbotiX-M attached.
- Mac and Linux users may have 2 ports - one marked 'cu.' and one marked 'tty.' Either will work.

Once you have set the board and serial port, you can open the 'ArbotixBlink' sketch.

```
File -> Examples -> Libraries -> ArbotiX -> arbotix -> ArbotiXBlink
```

Click on the 'Verify' Button ✔ (the green check in the upper left). This will attempt to compile the sketch. If all of the software is installed properly you will see a 'Done Compiling' below. Now click on the 'Upload' button ➔ (the green arrow button next to the verify button). This will compile the sketch, and then load it onto the ArbotiX-M. If the hardware is connected proper see the green user light flicker while the Arduino IDE displays an 'Uploading' message. When the Arduino IDE displays 'Done Uploading' the user LED should blink on and off in a 1 secor Congratulations, you just programmed your ArbotiX-M Board!

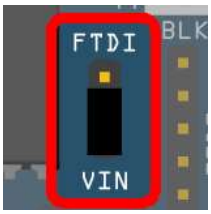
If you see any red text or error messages, then the sketch has not been loaded properly. Make sure you have installed the FTDI drivers, chosen the correct board and seria and that your USB connection is secure.

Back t

Step 6: Program the ArbotiX-M Robocontroller to Control a DYNAMIXEL Servo

Move the power jumper towards 'VIN'. This allow you to power the ArbotiX-M from the external power supply.

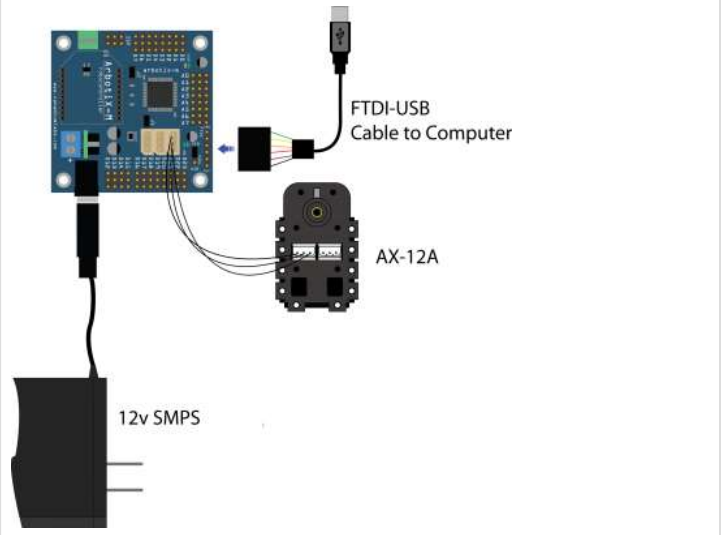
Power Jumper for ArbotiX-M



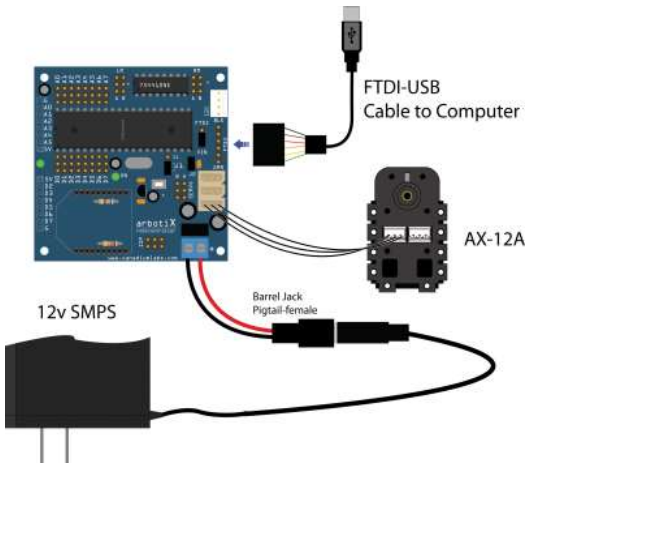
Power Jumper for ArbotiX

Next, connect your servo to the ArbotiX-M Robocontroller with a 3-pin DYNAMIXEL cable. All three DYNAMIXEL ports on the ArbotiX-M are identical, so you can plug the servo into any c **Note**: This example assumes that you are using a new AX-12A or AX-18A. All new AX servos are set to ID #1, which this example will target. This example is not intended for MX servos set to IDs other than '1'.

Now you will need to connect an external power supply to the ArbotiX-M board. In this example we'll be using a 12v Switched Mode Power Supply connected to the Arbotix's power termi through a Barrel Jack Pigtail. You can also use a charged LiPo battery connected via a LiPo battery wiring harness.



Single Servo Setup for ArbotiX-M



Single Servo Setup for ArbotiX

Now load the following sketch onto the ArbotiX-M Robocontroller

```
File -> Examples -> ArbotiX -> Test Sketchs -> AXSimpleTest
```

The servo should now begin to rotate counter clockwise, then clockwise. It will repeat this behavior until you power off the ArbotiX-M or re-program it.

Back t

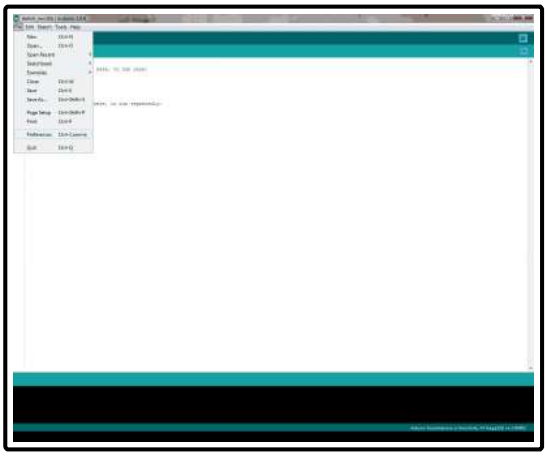
If you are working with MX servos, we reccomend you test them in the next step, Setting DYNAMIXEL IDs with the DynaManager. You can also use `File -> Examples -> A -> Test Sketchs -> MXSimpleTest`, though you may need to make some adjustments to the baud rate.

Step 7: Setting Your Preferences

Arduino has many features and preferences to customize your experience. Before you get started, let's set a couple of the basic preference . Open up your preferences menu:

```
File -> Preferences
```

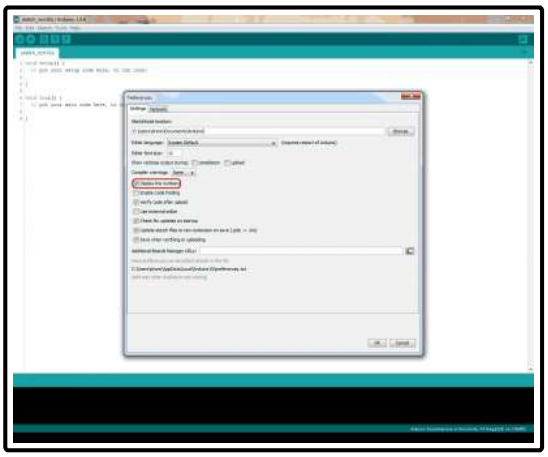
Mac users will find their preferences under `Arduino -> Preferences`



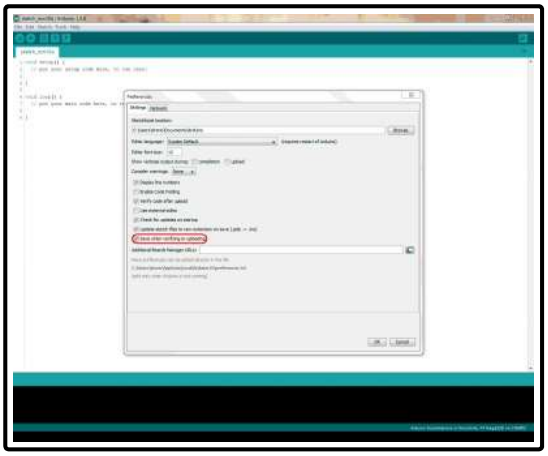
Now take a look at the settings tab:

```
Preferences -> Settings
```

There is a column of checkboxes. We recommend checking "Display line numbers" for easy debugging.



There is also a default setting, "Save when verifying or uploading". We recommend that you keep this checked. This will save your sketch automatically when you click on the Verify Button ✔ Upload Button ➔.



Congratulations, you just programmed your ArbotiX-M! Go back to the Getting Started Guide for your kit to proceed.

Back t

We are still updating learn.trossenrobotics.com to accomadate the new 1.6 hardware and library files. Some documetnation may still reference

```
File -> Sketchbook -> ArbotiX Sketches
```

This folder has now moved to be included in the ArbotiX Library Exaamples

```
File -> Examples -> ArbotiX
```

Next: Set DYNAMIXEL IDs

Next : Setting DYNAMIXEL IDs with the DynaManager

Back t

CHAPTER

B

Setting up Peter Corke's Robotics Toolbox

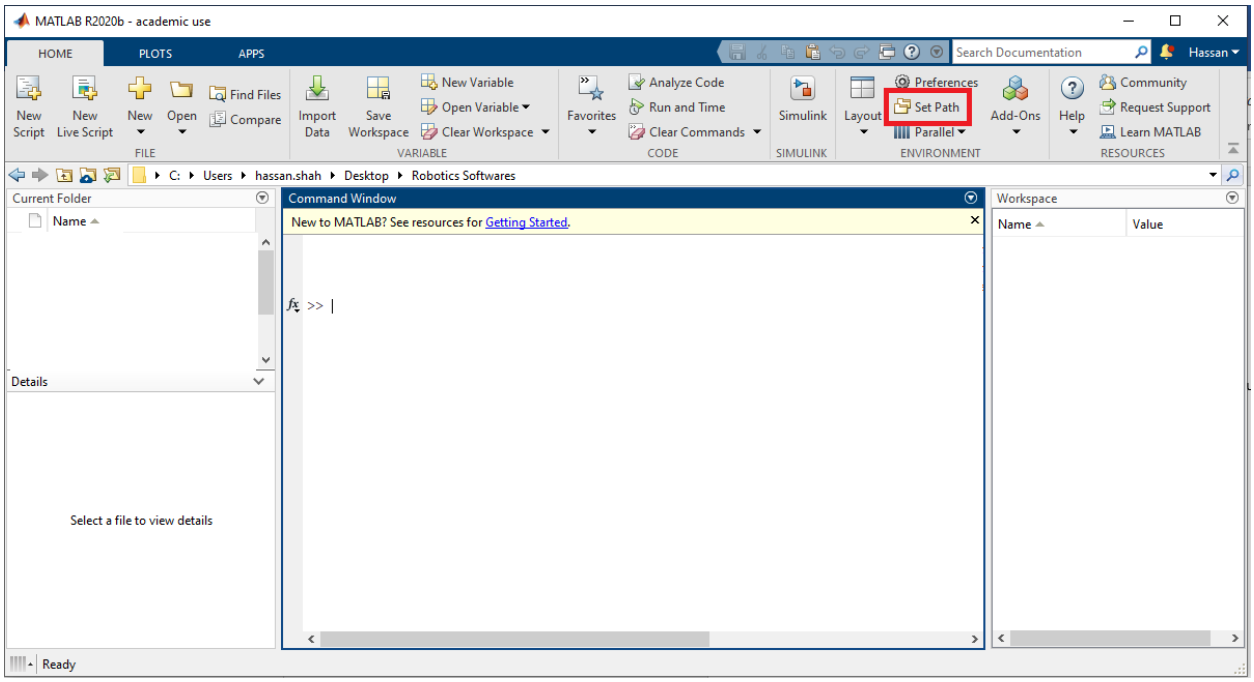
Install Tool Box

1. Download RTB10.4.mltbx from following link.
   https://petercorke.com/toolboxes/robotics-toolbox/

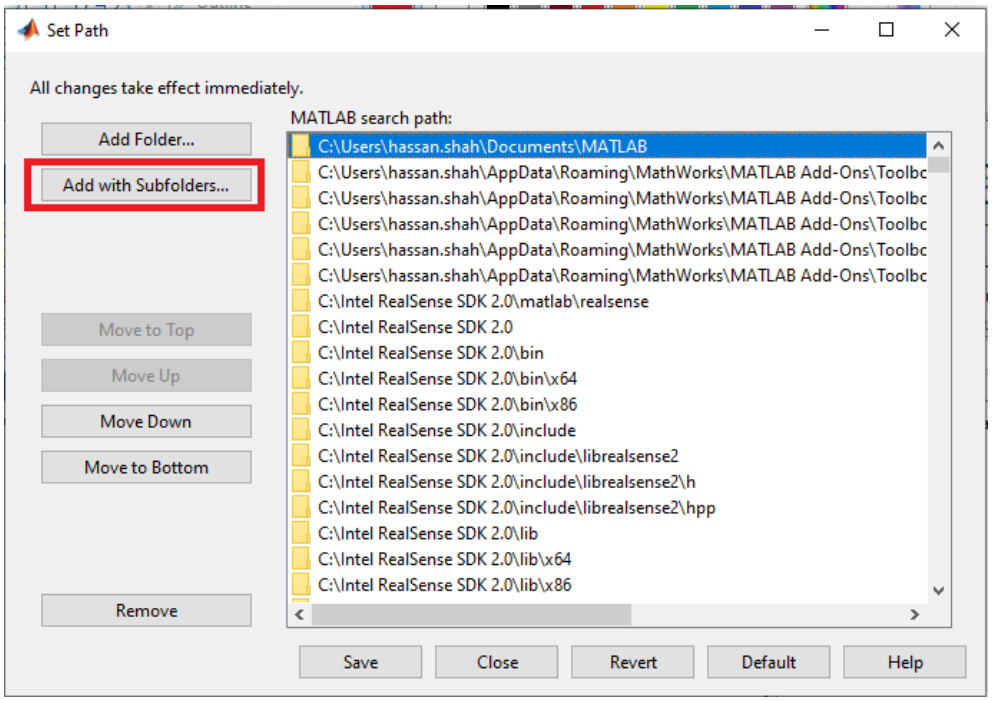
2. Navigate to the ".mltbx" file from file explorer in matlab and then double click to install.

Add Path

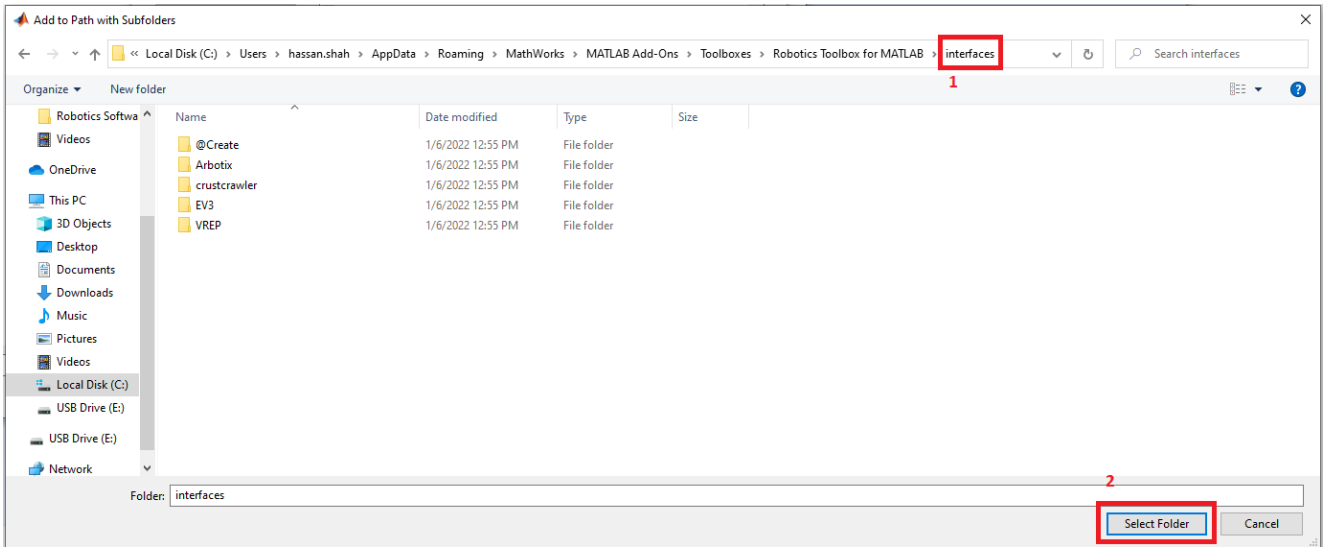
3. Click on Set Path.



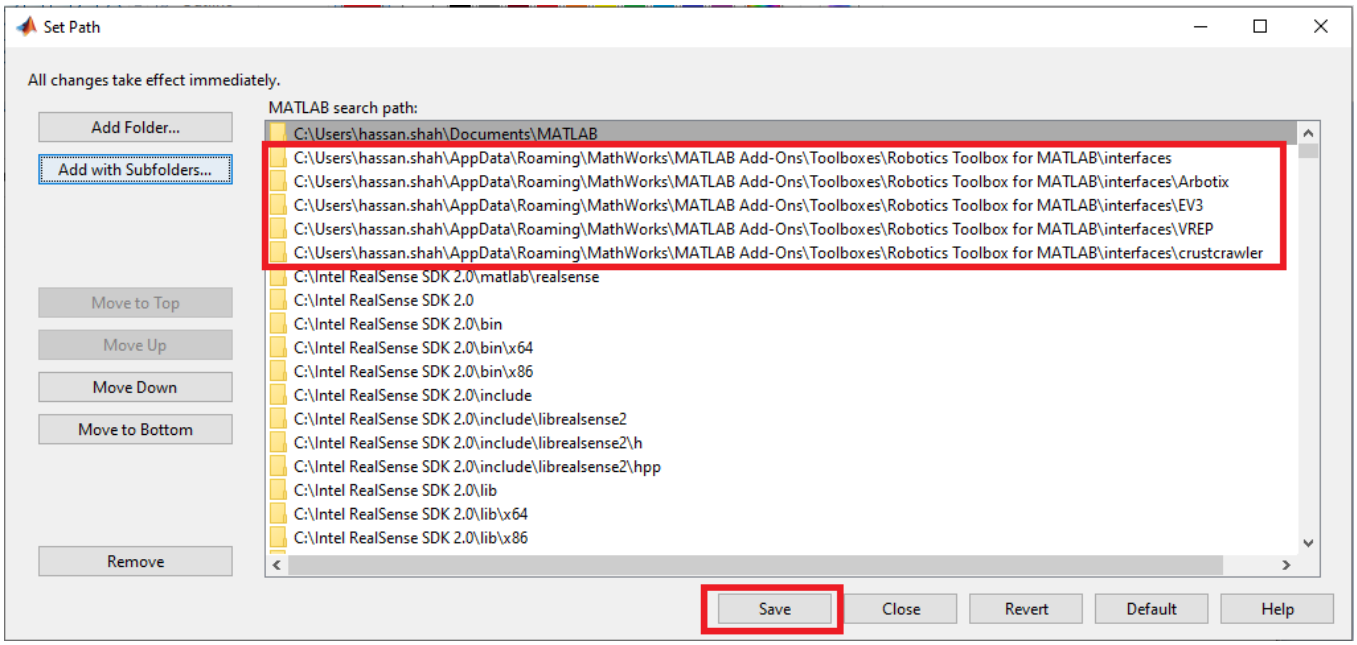
4. Click on Add with subfolder.



5. Navigate to Robotics Toolbox for Matlab/interfaces and click select folder.



6. Path to 5 folders will be added, click save.



7. Download rtb_real_robot.pdf from following link.
   https://petercorke.com/matlab/interfacing-a-hobby-robot-arm-to-matlab/

8. Follow heading 3.1 to test your robot.