EE366L/CE366L: Introduction to Robotics

Lab Handbook

Second Edition, Spring 2025

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 subsystems; (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 hardware for 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 led to the improved second edition of this handbook in 2023.

Change Log

2025

A discussion on perspective projective model in class. The image processing MATLAB course has been integrated with robotic system development tasks.

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 appendix. 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 and Safety Instructions

0.1 Installation Guides

The hardware setup instructions are included in the first chapter, a specific hardware equipment is used. MATLAB will be used as the primary development environment throughout the course. The required software has already been set up on the lab computers. A software installation guide to set up the required drivers and software on your own computer is provided on LMS and is also included as Appendix A and Appendix C. You can get a licensed version of MATLAB for your own machine under HU's license at https://www.mathworks.com/academia/tah-portal/habib-university-31522703.html

0.2 Expectations during the lab

The lab work in this course is intensive, as is typical for project-based learning. To manage your workload effectively, make the most of the allocated lab hours. You are welcome to access the lab whenever it is available outside of scheduled hours. To enhance the lab environment for extended work sessions, feel free to provide constructive suggestions to the course instructors.

Tasks marked with an [*] do not require hardware and can be completed outside the lab. Since this course involves group work, it is important to rotate responsibilities within your group to ensure that every member gains hands-on experience and contributes equally to the project. 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.

0.3 Care of the equipment (Do not skip this section)

The forces and torques generated by the robotic arm are not strong enough to cause serious harm to a human. **However, always ensure that your body remains outside the arm's workspace to minimize the risk of any potential injury.**

A collision whether between the arm and itself or between the arm and objects in the environment can **cause significant damage to the arm.** Stay attentive to the arm's motion and **immediately unplug it if a collision appears imminent.** Be mindful that once powered off, the arm may relax and fall under its own weight. **Always be prepared to catch it and gently place it back onto the baseboard.**

Each joint of the arm is driven by a motor with a delicate gear assembly. **Do not attempt to move the joints by hand when the arm is powered on, as this will damage the internal gears.** If you need to reposition the arm manually, **first power it off and then move the joints gently.** Avoid applying excessive force during manual adjustments.

"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 the manipulator and MATLAB

The objectives of this lab are to:

(i) get familiar with the hardware of the Phantom X Pincher arm, shown in Figure 1.1 and manufactured by Trossen Robotics, which will be utilized for this project;

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

(iii) explain the arm's operational abilities with simple physics;

(iv) enhance MATLAB skills.

Task 1.1 Team Formation (5 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.

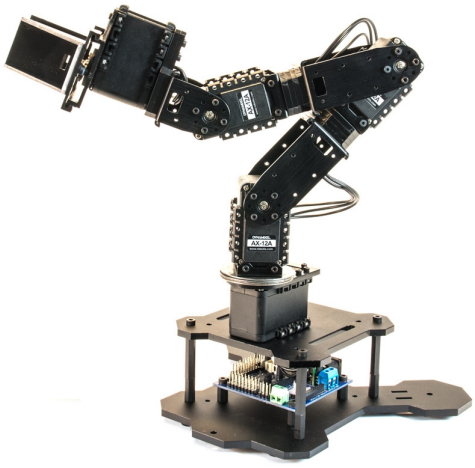


Figure 1.1: Phantom X Pincher Arm

1.1 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 1.2 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 1.1 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?

1.2 The Arbotix-M Controller

The Arbotix-M controller, depicted in Figure 1.3, 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 1.2.

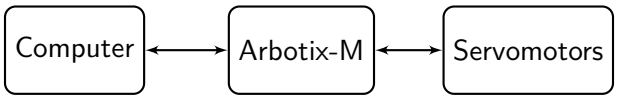


Figure 1.2: Data flow between MATLAB and servomotors

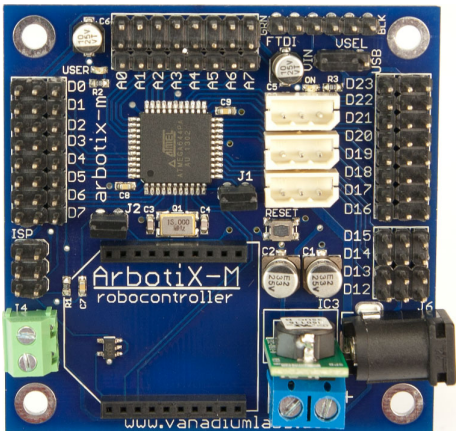


Figure 1.3: The Arbotix-M Controller

The board can also be used for additional processing. Figure 1.4 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.

1.2.1 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 1.5. You can also broadcast messages intended for all motors.

Definition 1.2.1: Sensors

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

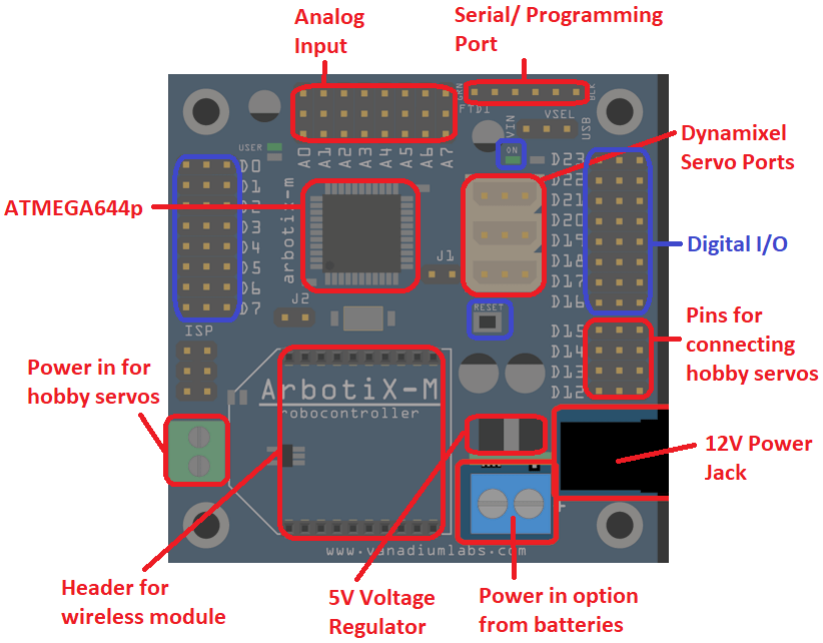


Figure 1.4: Getting to know Arbotix-M

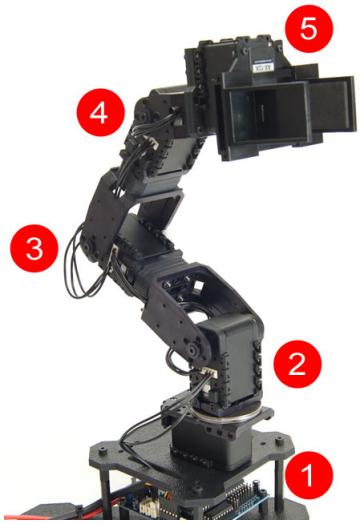


Figure 1.5: IDs of the servos in arm

Definition 1.2.2: Actuators

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

1.2.2 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 1.6.

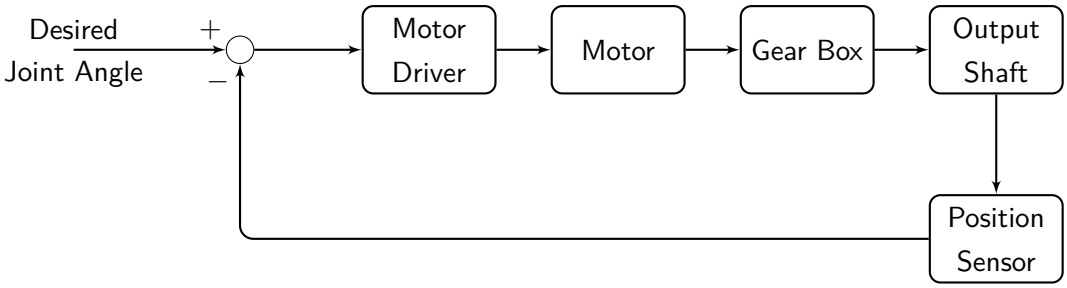


Figure 1.6: Block diagram of a servomotor

Our robot arm has five Dynamixel AX-12A servomotors. The functional components highlighted in the previous block diagram are captured in Figure 1.7 (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.

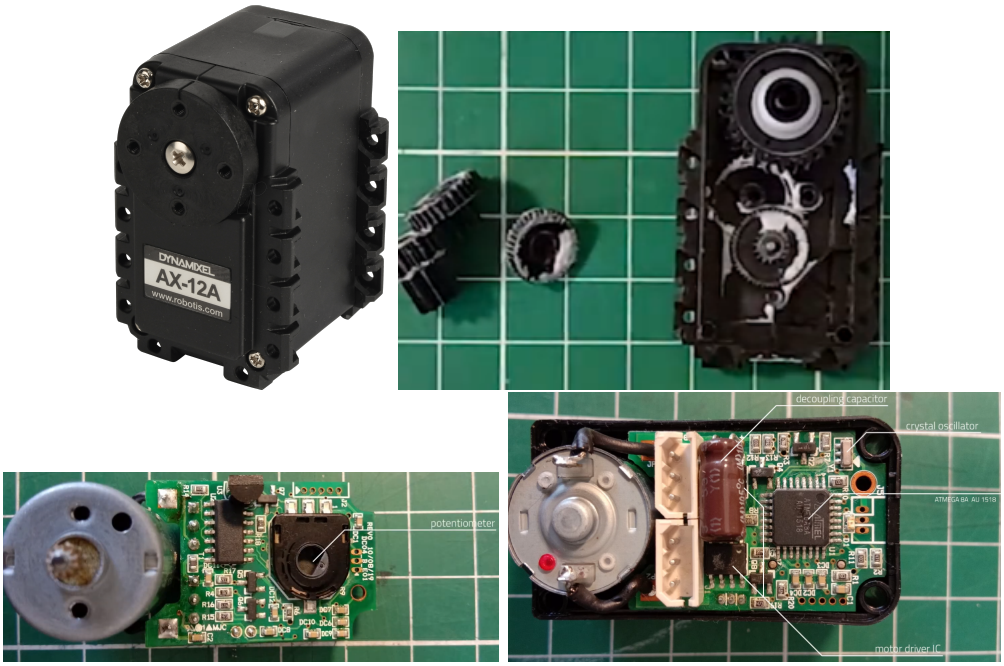


Figure 1.7: Top Left: AX-12A Motor, Top Right: Gear Box, Bottom Left: Potentiometer, Bottom Right: Microcontroller and Motor Driver

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.

1.3 Vision-based Pick and Place

Your lab objective is to design a vision-based pick and place robotic system, i.e. the system should be able to identify the object in the environment to be picked, pick the object, move to the desired location, and place the object at the desired location.

Task 1.3 Design of vision-based system (20 points)

(a) [*] Figure 1.7 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) Based on Figure 1.6, you're aware that the servomotors installed in the arm expect the desired joint angle as input. Furthermore, imagine that you have an overhead camera that has the environment in its field of view. Draw block diagram of the complete 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.

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

1.4 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 1.1 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 1.1: 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.

1.4.1 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 1.8, 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

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

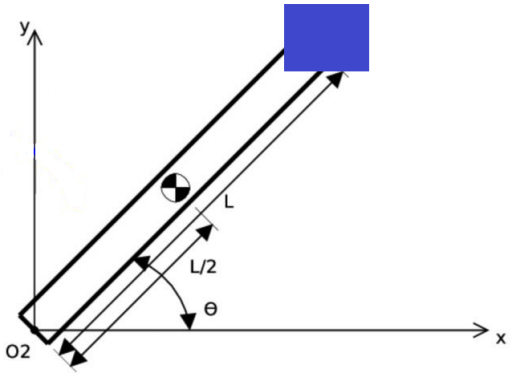


Figure 1.8: One link arm

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 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 1.9. The lengths and masses

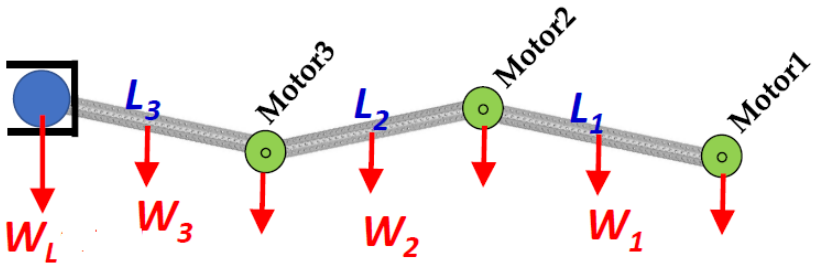


Figure 1.9: Multi-link robot arm

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.

Task 1.4 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 1.10, and physical principles outlined in Section 1.4.1, verify the wrist lift strength specified by the manufacturer. The wrist lift strength is the load that the wrist joint can support.

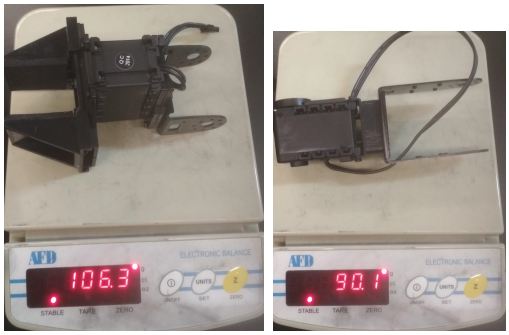


Figure 1.10: Masses of different arm components

1.5 MATLAB

As MATLAB will be used extensively throughout this course, you may want o familiarize 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, then you're suggested to complete the Stateflow On-ramp (https://matlabacademy.mathworks.com/details/stateflow-onramp/stateflow), which is useful for designing the higher-level operations of a robot.

Task 1.5 MATLAB On-Ramp (50 points)

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

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.

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

– David Marr

CHAPTER 2

Perception-I

The objectives of this lab are to:

(i) explore the camera being used;

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

(iii) design a vision pipeline for object pose estimation using RGB and depth images.

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

- Intel RealSense SDK 2.0
- MATLAB Image Processing Toolbox

2.1 System Overview

In the last lab, you may have come up with various questions when developing your block diagram, such as:

- 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?

As we progress through the semester, you'll find answers to all of these questions and you're encouraged to maintain a list of all questions. Our system development will begin with the development of the perception pipeline. The overall robotic system you'll be guided to build by this handbook is captured in Figure 2.1, which also answers some of the questions above.

Figure 2.1: 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 2.1 for now.

2.2 Determining the object frame

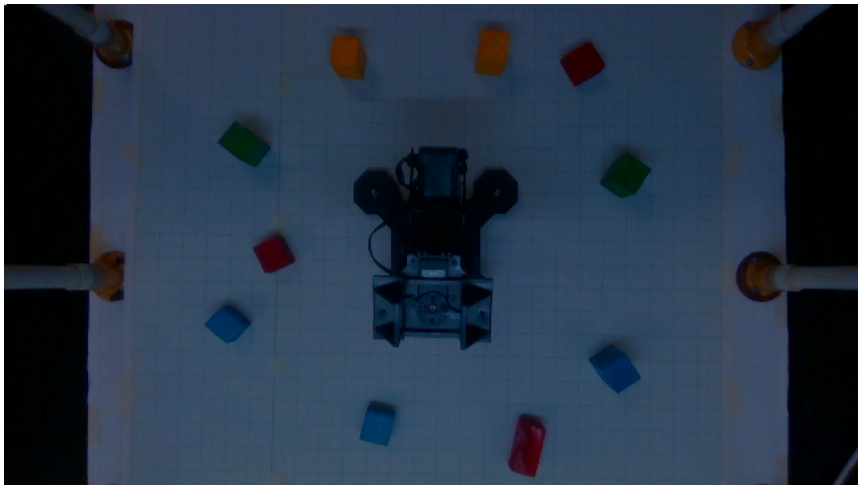


Figure 2.2: A possible task space as viewed from the camera

The first step in our pick and place pipeline is to determine the position and orientation of the objects to be picked, using image(s) of the environment from some camera(s), as illustrated in Figure 2.2. This problem can be further decomposes into three sub-problems, depicted in Figure 2.3. Our perception pipeline will have to:

- detect the objects of interest in the camera images, e.g. find all the red cubes in the image(s);

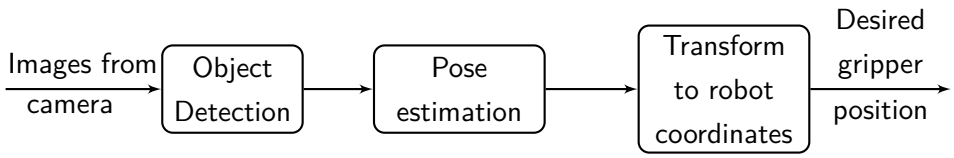


Figure 2.3: Block diagram of our vision pipeline

- estimate the pose (position and orientation) of the detected object in the camera frame (a real-world frame);
- transform the pose to the robot world frame.

The sub-problem of pose estimation can be formulated as that of determining the frame of an object of interest in the camera frame, as illustrated in Figure 2.4. The camera frame assignment

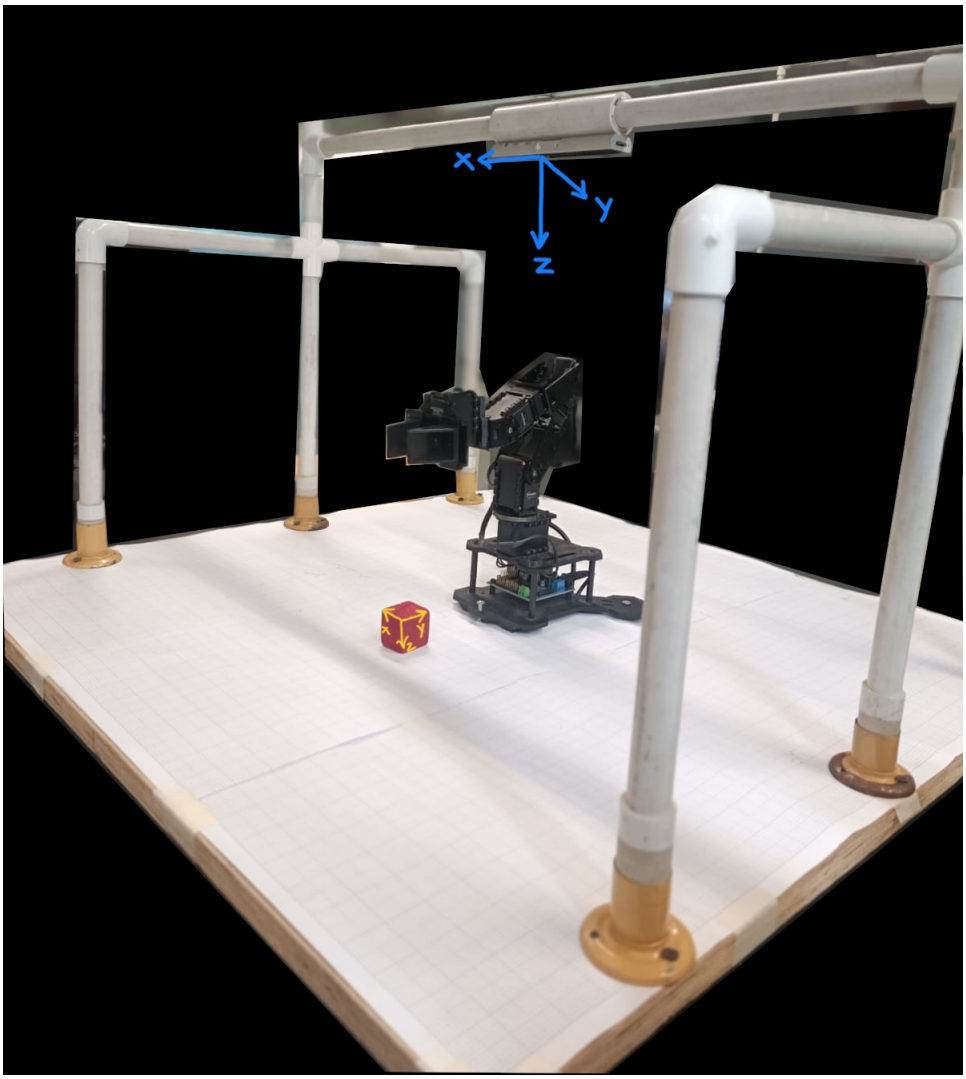


Figure 2.4: Frame assignments for pose estimation

is canonical - the origin of the 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 2.5. The object frame can be assigned arbitrarily as we find convenient.

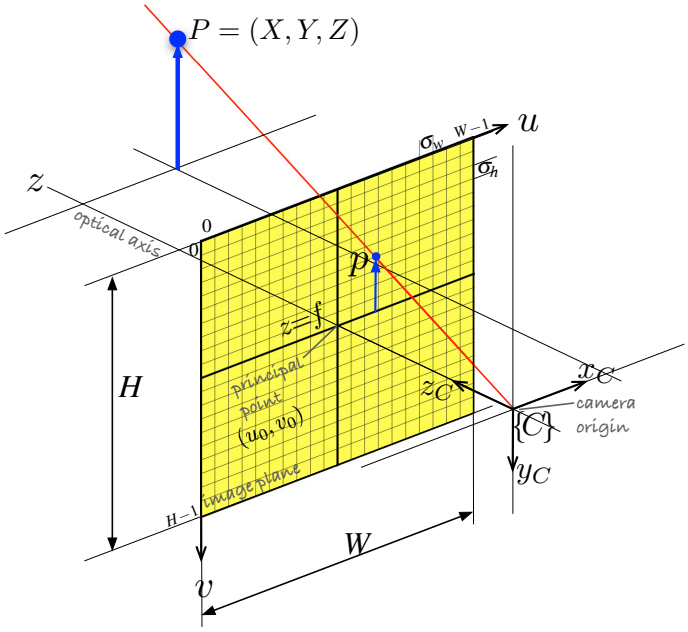


Figure 2.5: Camera Frame $\{C\}$ and the yellow Image Frame

Task 2.1 Image Generation Model (6 points)

Based on the discussions during the class, take the quiz '[L2] Image Formation' on LMS individually to test your understanding of the perspective projective model.

The perspective projective model informs us that there is a loss of information during the 3D to 2D process of image formation. Therefore, obtaining complete frame of the object requires information beyond one image of the scene. We'll consider two possible strategies here:

1. If we have prior knowledge of the object's geometry, then a perspective-n-point algorithm can be used to determine the object frame. In our case, objects are cubes of known geometry so we can determine its frame if we can determine locations of n points of the cube in the image.

2. With no prior knowledge of object's geometry, we require multiple images of the object from different perspectives or depth information.

2.3 RGB-D Cameras

You'll be using Intel RealSense SR305 camera, shown in Figure 2.6, in this project. A typical camera provides a 2D image, but this camera belongs to the class of RGB-D cameras that 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.

Figure 2.6: Intel RealSense SR305 Camera

A number of ways are being used in practice today to calculate depth, e.g. stereo images, LiDAR (time of flight), etc. The SR305 works on the principle of coded light. In addition to

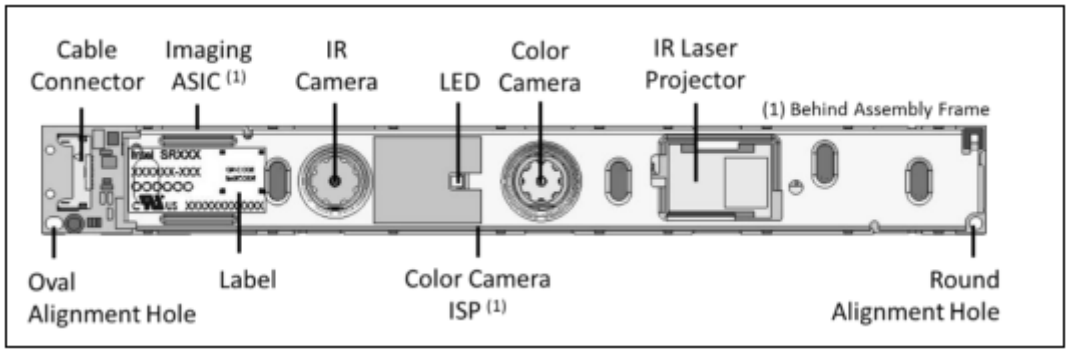


Figure 2.7: Components of an SR-305 Camera

an RGB camera, providing the usual color image, an SR305 has an IR (Infrared) camera and an IR emitter, as seen in Figure 2.7. The emitter projects patterned light, typically a pattern of

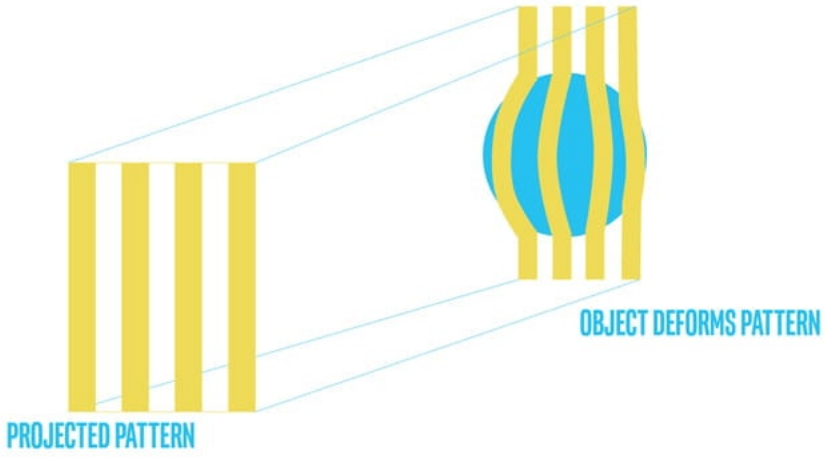


Figure 2.8: Principle of coded light

vertical bars, on to a scene, as illustrated in Figure 2.8. 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 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 2.2 Getting to know the camera (0 points)

Use `Intel RealSense Viewer` tool 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.

2.3.1 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.

2.3.2 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 2.9. Each pixel value in the RGB image is 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 2.10. Note that this distance is the normal distance from the camera plane to that object. The different colors in the depth image in Figure 2.9 correspond to different values of depth, and are obtained by mapping the depth readings in the image to a colormap.

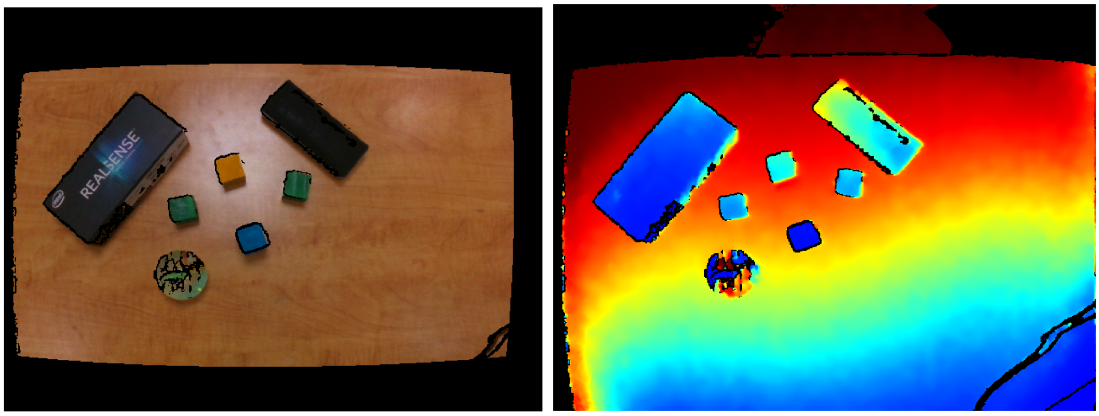


Figure 2.9: (Left) Image from RGB Camera. (Right) Depth Image

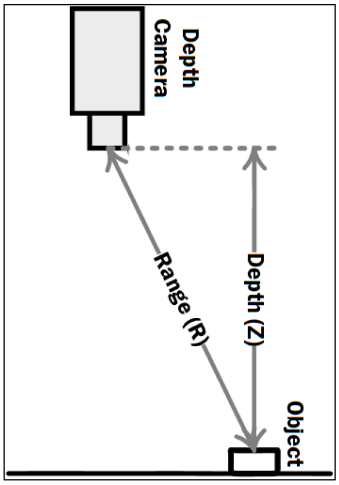


Figure 2.10: Depth value in the depth image

Noise in camera

We can notice from Figure 2.9 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 2.11.

2.4 Object Detection 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

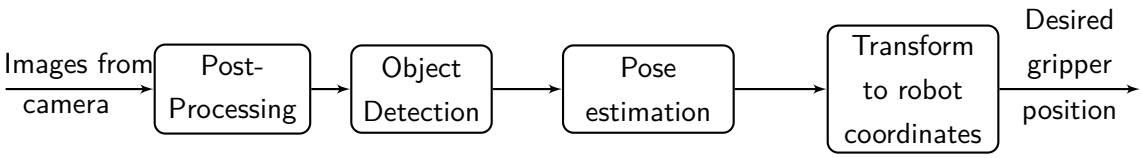


Figure 2.11: Complete flow of our vision pipeline

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. Further reading resources for this section are [3, Chapters 10, 12]

2.4.1 Image manipulation in MATLAB

The images captured by the camera will be imported in MATLAB. How are images represented and manipulated in MATLAB?

Task 2.3 Image Manipulation in MATLAB (10 points)

Complete lessons 2.1-2.4 from Module 'Working with Image Data' of the course 'Image Processing with MATLAB' (https://matlabacademy.mathworks.com/details/image-processing-with-matlab/mlip). A number of tasks in this lab will require you to complete various modules of this course, so keep it open. Your completion for all such tasks will be saved in the 'Progress Report', which you'll submit once along with your lab findings report. The 'Image Processing On-Ramp' can provide further help.

Task 2.4 Get a color image of the scene (10 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 and presets in the RealSense Viewer that will result in an aptly exposed image and accurate depth values at the highest resolution. Change the ambient lighting and find the best ambient lighting choice. Ignore the filters for now. Use [4] and [5] as reference. Save the RGB image to be used in subsequent tasks.

2.4.2 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 2.12 provides an instance of this method in use based on the MATLAB example https://www.mathworks.com/help/images/ref/imbinarize.html. The

Figure 2.12: 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 2.5 Thresholding (10 points)

Complete lesson 'Working with Binary Images' from Module 'Working with Image Data' of the course 'Image Processing with MATLAB'.

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 2.13.

Task 2.6 Color Segmentation (10 points)

Complete Module 'Color Segmentation' of the course 'Image Processing with MATLAB'.

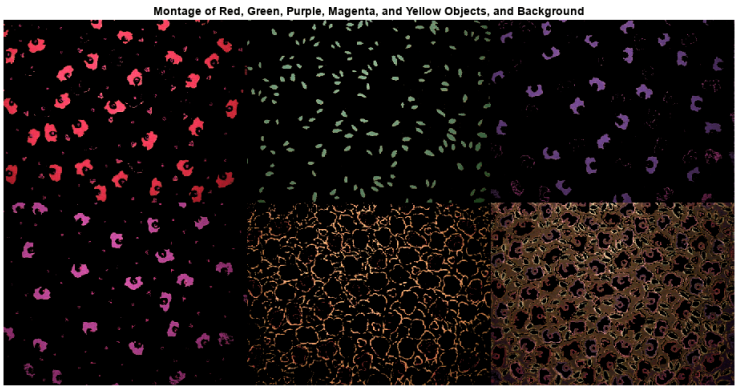


Figure 2.13: Segmentation based on color in L*a*b color space

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 automatic clustering of all data points into possibly k clusters [6]. 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.

Task 2.7 Find colored objects (20 points)

Develop a color segmentation stage for determining the pixels corresponding to all the cubes in the workspace of a single color. You are required to submit

(i) a note describing your choice of color segmentation strategy and the rationale behind it;

(ii) aptly commented code for your algorithm;

(iii) before and after images for two colors;

(iv) comments on the success of the algorithm; Are areas of the object missed (zoom in)? Does the location of the object have an effect? Does ambient lighting have an effect?

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 2.8 Connected Components (10 points)

Complete module 'Finding and Analyzing Objects' of the course 'Image Processing with MATLAB'.

Task 2.9 Labeling cubes (10 points)

Develop a labeling stage that clusters the pixels corresponding to a cube. You are required to submit

(i) aptly commented code for your algorithm;

(ii) labeled image of the run;

(iii) comments on the success of the algorithm.

You can see that the accurate detection of the cubes relies on the accuracy of the segmentation stage.

Task 2.10 Improving Segmentation (10 points)

Complete module 'Improving Segmentation' of the course 'Image Processing with MATLAB'.

Task 2.11 Improved object detection pipeline (20 points)

Based on the observed problems and knowledge acquired from the previous tasks, develop an improved pipeline from a raw RGB image to an image with labeled cubes of one color. You are required to submit

(i) a note identifying each problem, your solution for it in the new pipeline, and your rationale for why it works;

(ii) aptly commented code for your algorithm;

(iii) raw image, labeled image from the previous pipeline, labeled image of the new run.

2.5 Pose Estimation

Finding the object pose requires determining the description of the origin and the the axes of the object frame (yellow frame in Figure 2.4) in the physical world. This section will explore various ideas to obtain object pose given images, but first let's explore an alternate way to detect and locate an object in our environment.

2.5.1 AprilTags

AprilTags are artificial optical markers that can be placed on objects and then detected by a robot. The markers are designed so that objects can be identified uniquely and its pose can also be estimated. We'll use MATLAB's `readAprilTag` function to identify and determine the pose of a cube in our robot's workspace. Download a 'tagStandard41h12' family tag from `https://github.com/AprilRobotics/apriltag-imgs`, scale it appropriately to fit top face of a cube, print it, and stick it on top of a cube. The MATLAB function requires knowledge of the camera intrinsics, so let's first obtain this information.

Task 2.12 Camera Intrinsic Parameters (4 points)

The MATLAB function `determineIntrinsics()`, provided on LMS, uses Intel's SDK[a] to display the intrinsic parameters of the color camera. Elaborate each entry in light of acquired knowledge from class and https://github.com/IntelRealSense/librealsense/wiki/Projection-in-RealSense-SDK-2.0.

[a] Intel provides an SDK ([7]) for its RealSense line of cameras. We'll use the MATLAB wrapper provided by SDK. You can install the SDK by using the provided installer. Make sure to check `MATLAB Developer Package` option during installation. The package will be installed to `C:/Program Files (x86)/Intel RealSense SDK 2.0/matlab/`. Add the RealSense library to your MATLAB path by navigating to 'Set Path' in the 'Home' ribbon on MATLAB and 'Add with subfolders' the library directory outlined above.

Task 2.13 Detect AprilTags (10 points)

Use the MATLAB function `readAprilTag` to obtain the pose of an AprilTag placed on the top face of a cube. Use a ruler to physically measure the frame transformation and verify the accuracy of the result obtained from the MATLAB function.

(Optional Read) Field of View

The field of view of a camera identifies portion of the scene space that is actually projected onto the image plane and captured by the camera. The horizontal field of view, vertical field of view, and diagonal field of view are defined as illustrated in Figure 2.14. The field of view is expressed as an angle, and is typically specified as half of the fov angle in datasheets, as is the case for SR-305 too, e.g. The vertical fov is α° in Figure 2.14, and would be specified as $\alpha^\circ/2$ in its datasheet. The field of view of any camera lens is a constraint imposed by the physical size of the camera sensor and the focal length of the lens. It can be obtained as shown in Figure 2.15, where $\alpha/2 = \arctan(d/2f)$.

Task 2.14 Camera datasheet (0 points)

From the provided datasheet [8], determine the values of the following and provide an explanation for each quantity:

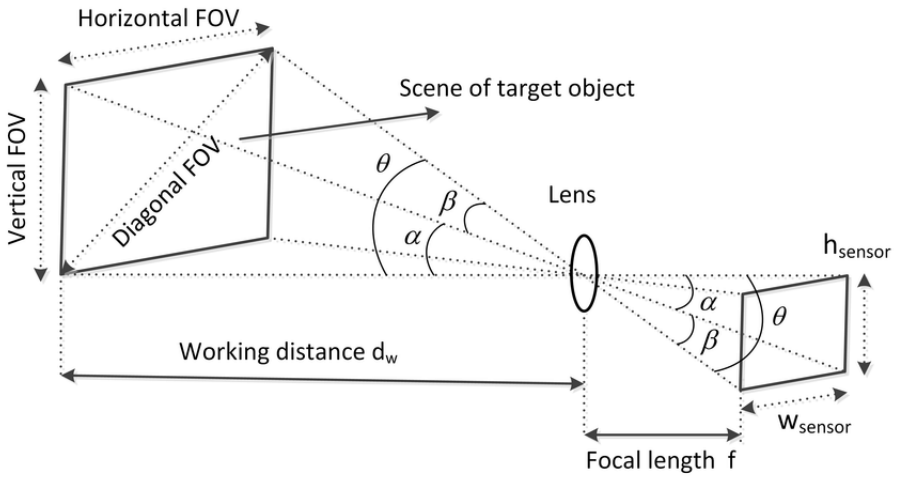


Figure 2.14: Field of view of a camera lens (Source: Ngo, T., Abdukhakimov, A., Kim, D. (2019). Long-Range Wireless Tethering Selfie Camera System Using Wireless Sensor Networks. IEEE Access.)

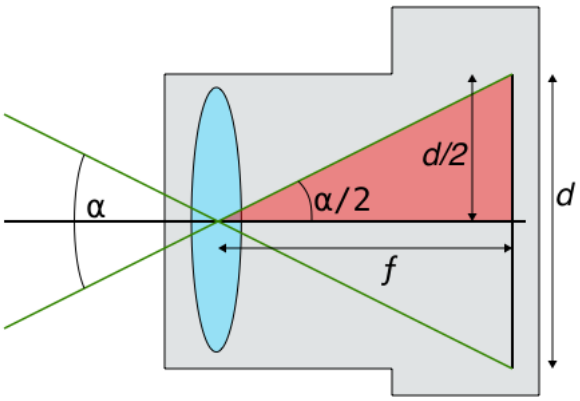


Figure 2.15: Focal length and sensor size determine fov

(a) Resolution of color camera and IR camera;

(b) Frame rates of both cameras;

(c) Depth field of view;

(d) Depth start point.

2.5.2 Pose estimation without fiducials

The previous approach requires infrastructural changes, specifically the objects of interest have to be marked by an optical fiducial marker, e.g. an AprilTag. This may not always be possible, but our approach of detecting objects in a visual scene followed by a determination of their pose is generally applicable.

The successful completion of Task 2.4.2 has enabled us to find relevant blobs in an image, e.g. given a mission, 'find the red blocks', we're able to find all pixels corresponding to each red block, or we have information at pixel level for each red block. Additionally, we also have a depth image that contains the depth value or real-world Z for every pixel in the RGB image. Using the images coordinates of all the pixels 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. Following are some possible ideas (untested) for pose estimation grouped together by the two information approaches outlined in Section 2.2. All of these approaches make use of the perspective projective model to relate pixels (point) (u, v) in an image to points (X, Y, Z) in the real world. A detailed view of the pipeline is provided in Figure 2.16.

- **Using cube's known geometry.**
  - Find three corners in RGB image and determine their corresponding real-world points using known length of the cube.
  - Determine edges in the image and make use of the knowledge that they are orthogonal.
  - Matching the edges with an edge template of the cube.
- **Using the depth image.**
  - Identify faces (planes) of the cube using depth data.
  - Identify a corner in RGB image and then determin corresponding real-world coordinates using depth data.
  - Combine RGB data and depth image into a point cloud and process that (next chapter).

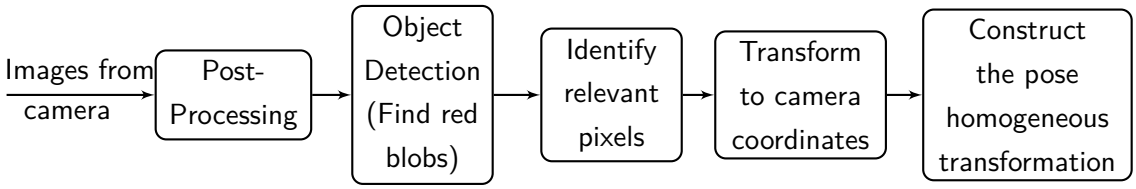


Figure 2.16: Detailed vision pipeline

Detecting geometric features

A number of these ideas require determining geometric features of objects. Typically, local features such as edges are identified first from pixel data. This local feature data is then used to identify global featues, such as lines, planes, in an image. 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 geometric 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 example https://www.mathworks.com/help/images/hough-transform.html shows how to detect lines in an image using Hough Transform.

Task 2.15 Detecting Features (10 points)

Complete module 'Detecting Edges and Shapes' of the course 'Image Processing with MATLAB'.

Further Exploration.
Module 'Aligning images with image registration' from the same course provides techniques for matching images to templates.

Task 2.16 Getting the images in MATLAB (0 points)

A MATLAB function `depth_example` is provided on LMS. Use it to obtain a color and a depth image of your setting. The file provides an example of setting the camera parameters in code before acquisition. Because the two cameras each have their own coordinate axes and the pixel coordinates in the color and depth images will be with respect to their own coordinate axes, a necessary step is to align the two images.

Task 2.17 Determining pose (40 points)

Think about which geometric features could assist you in associating a pose with the object and determine the pose of an identified cube. You are required to submit

1. your assignment of a frame to an object;
2. description of an algorithm for constructing the pose homogeneous 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;
5. images at various steps of a good test run;
6. accuracy of the pose compared to physical measurements over multiple runs.

Task 2.18 Perception Pipeline (20 points)

In this task, you'll combine `depth_example` and your previously developed algorithms to develop an automated perception pipeline that acquires an RGB image and a depth image of your scene and output poses of all the cubes of your chosen color in the scene.

References

[1] A. Zabatani et al., "Intel® 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] P. Corke, W. Jachimczyk, and R. Pillat, *Robotics, Vision and Control: Fundamental algorithms in MATLAB*. Springer, 2023.

[4] Intel. "Tuning depth cameras for best performance. "[Online]. Available: https://dev.intelrealsense.com/docs/tuning-depth-cameras-for-best-performance.

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

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

[7] Intel. "Intel realsense sdk. "[Online]. Available: https://github.com/IntelRealSense/librealsense.

[8] *Intel realsense depth camera sr300 series product family datasheet*, SR300, Rev. 2, Intel, Jun. 2019.

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

– David Marr

CHAPTER 3

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

3.1 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, 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 2.4. 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 3.1. MATLAB has a whole set of functions available for point cloud processing

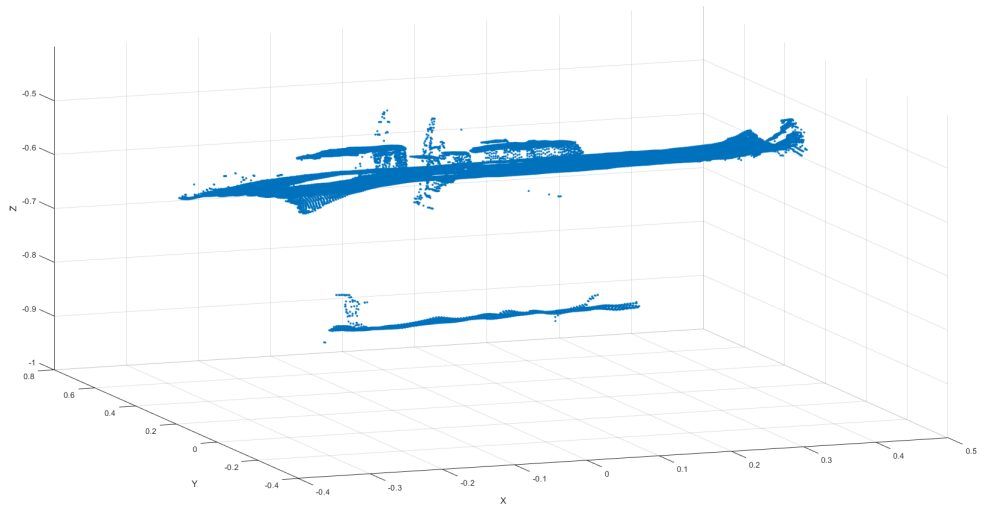


Figure 3.1: Point cloud of the same scene from Figure 2.9

that can be explored at https://www.mathworks.com/help/vision/point-cloud-processing.html.

3.1.1 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 3.1 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.

3.2 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 a relevant point cloud of an object, and then estimating the pose using geometry. This section outlines two possible ideas for this task.

3.2.1 Leveraging geometry of the scene

The objects to be detected have a well-defined geometric shape placed in a fixed scene. The robot arm will always be at a known location in the point cloud and could be removed. The

table will be the largest plane in the scene and could be filtered out by fitting a plane to the point cloud. Distinct cubes could be segmented based on Euclidean distances of neighboring points. The page [2] provides a list of MATLAB's built-in functions for point clouds processing. A detailed survey of different segmentation techniques for point clouds is [3].

3.2.2 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 3.2 Pose Estimation Pipeline (100 points)

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

1. a description of your entire pipeline, including
   - any filters/processing applied to the image frames and your process for determining the corresponding parameters,
   - segmentation strategy and rationale behind it,
   - pose estimation stage,
   - frame whose pose is being determined;
2. aptly commented code for the entire pipeline;
3. images at various steps of a good test run;
4. 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.

[3] A. Nguyen and B. Le, "3d point cloud segmentation: A survey," in *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, 2013, pp. 225–230. DOI: 10.1109/RAM.2013.6758588.

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

– Chebyshev to Sylvester, 1873

4

CHAPTER

Forward Kinematics

The objectives of this lab are to:

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

(ii) obtain forward kinematics mapping of the manipulator using standard DH convention or Product od Exponentials approach;

(iii) determine the reachable workspace of the manipulator;

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

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

- Arduino IDE 1.6.10
- FTDI drivers
- Interbotix ArmLink
- pypose
- Peter Corke's Robotics Toolbox
- MATLAB Robotic System Toolbox R2022a or later

- MATLAB Symbolic Math Toolbox

4.1 Move the arm

We'll first use a simple interface, InterbotiX Arm Link Software, provided by the manufacturer to control the arm. This will help us understand the final motion control system that we have to create in the subsequent few weeks.

4.1.1 Getting familiar with Arm Link Controls

Set up the ArmLink connection following the instructions outlined in Appendix-B. Open the ArmLink application and the panel in Figure 4.1 will appear on display. **Read through the next two sections carefully to be fully aware of safety instructions before playing around with the controls.**

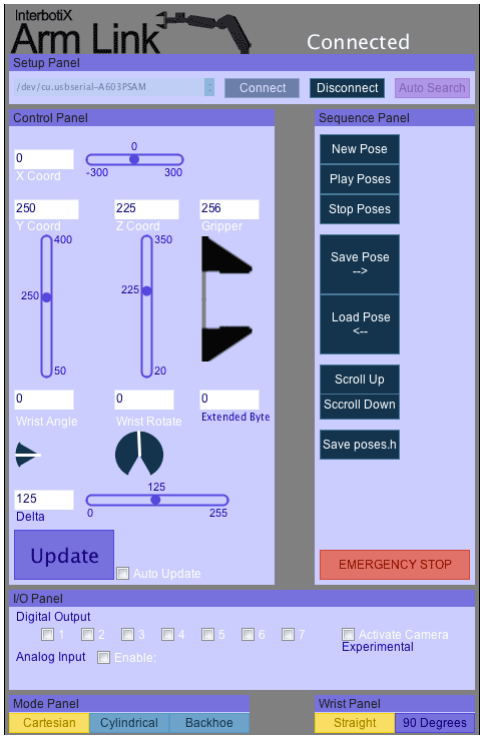


Figure 4.1: Arm Link Application Panel

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. It is better to use the `Update` option so that you can keep track of cause and action. **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.

4.1.2 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 4.1 Configurations Exploration (10 points)

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

(a) Move one joint at a time in Backhoe mode to get a sense of the motion of each joint.

(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) From the Dynamixel reference manual (https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/), find the angle rotation limits, resolution (see the definition below), speed limit, and torque limit[a] of AX-12A servo.

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

[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 4.1.1: 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.

4.2 Interacting with the arm using MATLAB

We'll now interact with the robot from MATLAB. For this, we'll make use of the `Arbotix` library written by Peter Corke as part of his Robotics toolbox [1]. Instructions for setting up the toolbox are provided in Appendix-C. Follow the steps below to get the arm set up to communicate with MATLAB.

4.2.1 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
   ```

4.2.2 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 [2]. 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.

4.3 Kinematics

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 employ either the DH convention or the product of exponentials approach to obtain 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.

4.4 Determination of forward kinematic mapping

Throughout this section, each task will have two sets of instructions - the top instructions correspond to the PoE approach and the bottom with the DH convention. We'll follow the conventions and procedures outlined in [3] for the standard Denavit-Hartenberg (DH) convention, details of which are outlined in Appendix-D.

Task 4.2 Frame Assignment (10 points)

Common Instructions: The base frame is to be assigned as follows: 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. Place the origin of the end-effector frame at the center of gripper motor horn, for convenience of measurements in upcoming tasks.

(PoE) Assign fixed and end-effector frames to the robot arm in Figure 4.2, according to the instructions above. Draw and paste each frame's to the robot body at an appropriate location. This will help your visualization in later tasks.

(DH) Using standard DH convention, assign DH frames to the robot arm in Figure 4.2. Make sure to clearly indicate the z and x axes, and the origin of each frame; drawing the y axis is optional. 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.

Task 4.3 Zero Configuration (15 points)

(PoE) Using MATLAB's connection to the arm determine the zero configuration of the arm and represent it as a homogeneous transformation, M. You'll have to physically measure the lengths. Also determine the positive direction of the axes of rotations and mark them on Figure 4.2.

Figure 4.2: DH Frame assignment

Task 4.4 Parameters (15 points)

(PoE) Determine the screw axes, $\mathcal{S}_i$, corresponding to each of the joints, i, of the robot arm, expressed in the base frame.

(DH) Annotate Figure 4.2 with DH parameters based on your frame assignments, complete Table 4.1, 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 4.1: Table of DH parameters

Task 4.5 Homogeneous Transformations (5+5+3 points)

(PoE) Use MATLAB's symbolic math toolbox to determine the rigid-motion corresponding to each of the screw axes and the resultant transformation ${}^0T_4$. See below for specific deliverables.

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

(Common Instructions)

(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.

(b) Obtain ${}^0T_4$ by multiplying the previously determined homogeneous transformations in the appropriate order.

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

Symbolic Math Toolbox Tips

- 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 convention for naming the variables storing homogeneous transformations may result in convenience later. The MATLAB functions `simplify` and `expand` may be of help in simplifying the final expressions.

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?

4.4.1 Building the functions in MATLAB

Task 4.6 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 [4].

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 [5]. The provided MATLAB script file `pincherModel.m` builds a model for Phantom X Pincher.

Task 4.7 Verification of Forward Kinematic Mapping (5 points)

PoE To be updated.

(DH) 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.

Common Instructions 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.

Task 4.8 DH and Servo Joint Angles alignment (15 points)

(DH) The range for θ_i may not be $[-150^\circ, 150^\circ]$ 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.

(a) Map the DH joint angles to the respective servomotor angles in Table 4.2 and Table 4.3. 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 4.3.

(b) 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 4.2.

4.5 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.

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 4.2: 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 4.3: Joint Limits

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 $[-150°, 150°]$.

Task 4.9 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.

4.5.1 Controlling the arm from MATLAB

We'll now query the robot from MATLAB and physically verify that our forward kinematics computations are correct.

Task 4.10 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?

References

[1] P. Corke. "Robotics toolbox. "[Online]. Available: https://petercorke.com/toolboxes/robotics-toolbox/.

[2] P. Corke. "Dynamixel ax12a servos. "[Online]. Available: https://petercorke.com/robotics/dynamixel-ax12a-servos/.

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

[4] Mathworks. "Functions in matlab. "[Online]. Available: https://www.mathworks.com/help/matlab/functions.html.

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

CHAPTER 5

Resolved Rate Motion Control

The objectives of this chapter are to:

(i) determine the Jacobian and singularar configurations for our manipulator;

(ii) apply resolved rate motion control - a Jacobian-based approach to move our manipulator to the desired end-effector configuration.

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

- Arduino IDE 1.6.10
- FTDI drivers
- Interbotix ArmLink
- pypose
- Peter Corke's Robotics Toolbox
- MATLAB Robotic System Toolbox R2022a or later
- MATLAB Symbolic Math Toolbox

In the previous chapter, you constructed the forward kinematic mapping of the manipulator, which relates the end-effector pose to the joint variables. The speeds of the manipulator's

joints or end-effector were not considered. For some tasks, e.g. spray painting, not only do we care about precision but also desire the task to be carried out at uniform speed. In this lab, you'll determine Jacobian of this manipulator, which relates the joint velocities to end-effector velocities. Having determined the Jacobian, you'll identify the kinematic singularities of this manipulator. Finally, you'll utilize the Jacobian to implement the resolved rate motion control algorithm and move the manipulator to the desired end-effector configuration.

5.1 Determination of the manipulator Jacobian

Definition 5.1.1: Jacobian

The geometric Jacobian, J, of a manipulator is a $6 \times n$ matrix that relates the twist of an end-effector to the joint velocities, i.e. $\mathcal{V} = J(q)\,\dot{q}$, where $q \in \mathbb{R}^n$ is a vector of joint variables. MATLAB provides a function `jacobian` to compute the geometric Jacobian.

The Jacobian of this manipulator will be a 6×4 matrix, $\begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$, where J_v is the Jacobian for linear velocity and J_ω for the angular velocity. The Jacobian matrix can be determined column by column as outlined in the book [1], where each column corresponds to the respective joint of the manipulator.

Task 5.1 Manipulator Jacobian (20 points)

Use the homogeneous transformation, ${}^0T_4$, obtained in the previous lab to find either the space Jacobian or the body Jacobian[a] for the manipulator in the lab.

[a] Singularity analysis in the next task is easier with the body Jacobian.

Task 5.2 Rank of Jacobian (5 points)

(a) What is the maximum possible rank of this Jacobian?

(b) Will we able to achieve arbitrary linear velocity and arbitrary angular velocity of the end-effector?

5.2 Singularity Analysis

Definition 5.2.1: Kinematic Singularities

Kinematic singularities are configurations of a manipulator where the rank of the Jacobian is less than its maximum possible rank. At singularities, manipulator loses its abilities to generate velocities in certain directions and consequently its ability to move instantaneously in those directions. You may also need large joint velocities near singularities to achieve even small end-effector velocities.

Determining the conditions at which the Jacobian drops rank is not an easy task. The process

is made simple if we leverage the fact that the Jacobian singularities are an inherent property of the physical manipulator and are unaffected by our choice of the base or end-effector frames. So, we can place the end-effector frame at a position that yields a simpler Jacobian and simplifies our singularity analysis. This new Jacobian is only for the purpose of singularity analysis and will not be the best candidate for determining velocities.

Task 5.3 Singular Configurations (20 points)

Determine the conditions at which the rank of Jacobian drops below its maximal rank and the corresponding singular configurations.

- We'll choose the origin of our end-effector frame on the axis of our last joint. If you chose it earlier at the center of the end-effector, you can effectively shift it by setting the last length to be zero.
- You can use the function `subs` to substitute $a_4 = 0$, e.g. if the Jacobian is stored in `J`, then we use `subs(J,'a_4',0)`.
- Remember to make judicious use of `det`, `simplify` and `expand`.

(a) Show that the Jacobian loses rank when $\sin\theta_3 = 0$.

(b) How would you describe the shape of the arm at singular configurations? At singularity, in which direction is the arm unable to move?

Now that you have determined the singularities, keep them in mind when planning the motion of your manipulator.

5.3 Grasp Pose and Pre-grasp Pose

Let's think about our 'pick and place problem' again and determine where do we stand right now (Refer to Figure 2.1). We have previously developed

(i) a vision pipeline that determines the pose of an object, to be picked, with respect to the overhead-camera frame, ${}^{c}T_{o}$;

(ii) a forward kinematics mapping that determines the pose of the end-effector with respect to the fixed base frame, ${}^{s}T_{e}$.

Knowing that the camera placement relative to the base frame is fixed, ${}^{s}T_{c}$, one can easily determine the pose of an object of interest with respect to the base frame, ${}^{s}T_{o}$. The required pose of the end-effector, ${}^{s}T_{e}^{d}$ to grasp an object at its current configuration is called the **grasp pose** [2], e.g. if a cube frame's origin is at the center of the cube you may want your gripper frame's origin to be vertically above it by some length. What would be a suitable grasp pose for our manipulator and objects?

As you have determined, the 4 DoF manipulator in our lab is incapable of achieving arbitrary spatial positions and orientations of the end-effector. We can set a desired position of the end-effector arbitrarily, but with the available only one orientation degree of freedom, the most we can do is tilt the wrist. 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 an axis of the enf-effector frame fixed along the $\hat{z}$-axis of the base frame or the axis lying in the $\hat{x} - \hat{y}$ plane of the base frame.

To avoid collisions with other objects in the workspace, it is usually the practice to move the end-effector safely through the free space above all the objects to a pose directly above the object to be picked and then move straight vertically downwards. This intermediate pose is called a **pre-grasp pose**.

Task 5.4 Grasp and Pre-grasp poses (10 points)

Determine an appropriate grasp pose, ${}^{s}T_{g}$, and a pre-grasp pose, ${}^{s}T_{pg}$, in terms of a given object pose ${}^{s}T_{o}$.

5.4 Resolved Rate Motion Control

Having determined the pre-grasp pose of the end-effector, our goal is to move the end-effector in a straight line from its current pose to the pre-grasp pose at a constant speed. The resolved-rate motion control algorithm is a simple and elegant way to execute this motion [3]. Recall that

$$\dot{q} = {}^{s}J(q)^{-1}\,\mathcal{V}_s,$$

which can determine the required joint velocities for a desired end-effector twist. The required joint velocities can then be integrated to determined the required joint angles. However, we have two problems: (i) J can only be inverted if it is a square invertible matrix, which is not the case here, and (ii) How do we determine $\mathcal{V}_s$ for our case?.

In situations when J is not invertible, we can use the Moore-Penrose pseudoinverse,

$$\dot{q} = {}^{s}J(q)^{+}\,\mathcal{V}_s.$$

The pseudoinverse will determine the least squares solution, i.e. determine the joint velocities that minimize $\|{}^{s}J(q)\dot{q} - \mathcal{V}_s\|^2$.

How do we determine $\mathcal{V}_s$ for our case? The twist comprises of an angular velocity and a linear velocity. Owing to the limited dof of our manipulator and having fixed the wrist angle, we only need to set the position of our end-effector and correspondingly the linear velocity.

Task 5.5 Desired end-effector velocity (10 points)

Formulate an expression to determine the desired end-effector velocity, ${}^{s}v_e$, for straight-line motion, given the current position of the origin of our end-effector frame in the fixed frame, ${}^{s}p_{fk}$, as determined by our forward kinematics mapping and the desired position, ${}^{s}p_{pg}$, as determined by ${}^{s}T_{pg}$.

Provide a MATLAB `function ve = makeVE(p_cur,p_des,speed)` that accepts the current position, the desired position, and the linear speed as arguments and returns the desired end-effector velocity.

Since we're only interested in setting v_e, we will use the $J_v \in \mathbb{R}^{3\times 4}$ part of the Jacobian only, ${}^{s}v_s = {}^{s}J_v\dot{q}$ or ${}^{b}v_e = {}^{b}J_v\dot{q}$. Recall that we have also fixed the wrist angle, θ_4, e.g. the gripper is pointing vertically downward, and so $\dot{\theta}_4 = 0$. Consequently, only the first three columns of J_v contribute to determining the linear velocity, i.e.

$$v_a = \underbrace{\begin{bmatrix} J_{v1} & J_{v2} & J_{v3} \end{bmatrix}}_{{}^{a}J'_v} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix}, \quad \text{where } a \in \{s, b\}.$$

This results in a simplification: $J'_v(q) \in \mathbb{R}^{3\times 3}$ and as long as q is not singular, J'^{-1}_v exists.

Do we use the space Jacobian or the body Jacobian? Note that you have determined the desired end-effector velcoity in the fixed base frame, i.e. ${}^{s}v_e$. If you're using the body Jacobian, then

$$\dot{q} = {}^{b}J_v^{-1}\,{}^{b}v_e = {}^{b}J_v^{-1}(q)\;{}^{b}R_s(q)\;{}^{s}v_e.$$

Would using ${}^{s}J_v$ be straightforward? Sadly, no. Recall that space Jacobian maps to linear velocity v_s, which is not ${}^{s}v_e$, but $v_s = {}^{s}v_e - {}^{s}\omega_e \times {}^{s}p_e$. Therefore,

$$\begin{aligned}
\dot{q} &= {}^{s}J_v^{-1}\,{}^{s}v_s \\
&= {}^{s}J_v^{-1}\left({}^{s}v_e - {}^{s}\omega_e \times {}^{s}p_e\right) \\
&= {}^{s}J_v^{-1}(q)\,{}^{s}v_e - {}^{s}J_v^{-1}\left({}^{s}\omega_e \times {}^{s}p_e\right) \\
&= {}^{s}J_v^{-1}(q)\,{}^{s}v_e - {}^{s}J_v^{-1}\left({}^{s}J_\omega(q)\dot{q} \times {}^{s}p_e\right) \\
&= {}^{s}J_v^{-1}(q)\,{}^{s}v_e + {}^{s}J_v^{-1}\left({}^{s}p_e \times {}^{s}J_\omega\,\dot{q}\right) \\
\dot{q} - {}^{s}J_v^{-1}\left({}^{s}p_e \times {}^{s}J_\omega\dot{q}\right) &= {}^{s}J_v^{-1}(q)\,{}^{s}v_e \\
\dot{q} - {}^{s}J_v^{-1}\left[{}^{s}p_e\right]{}^{s}J_\omega\dot{q} &= {}^{s}J_v^{-1}(q)\,{}^{s}v_e \\
\left(I - {}^{s}J_v^{-1}\left[{}^{s}p_e\right]{}^{s}J_\omega\right)\dot{q} &= {}^{s}J_v^{-1}(q)\,{}^{s}v_e \\
\dot{q} &= \left(I - {}^{s}J_v^{-1}(q)\,\left[{}^{s}p_e(q)\right]\,{}^{s}J_\omega(q)\right)^{-1}\,{}^{s}J_v^{-1}(q)\,{}^{s}v_e
\end{aligned}$$

If we want to use the space Jacobian, then we have to make use of a complicated expression. However, another way to obtain J_v is by directly differentiating ${}^{s}p_e(q)$ obtained from the forward

kinematics mapping ${}^sT_e$, using the chain rule:

$$
\begin{aligned}
{}^sv_e = {}^s\dot{p}_e &= \frac{\partial\, {}^sp_e}{\partial\theta_1}\,\dot{\theta}_1 + \frac{\partial\, {}^sp_e}{\partial\theta_2}\,\dot{\theta}_2 + + \frac{\partial\, {}^sp_e}{\partial\theta_3}\,\dot{\theta}_3 + \frac{\partial\, {}^sp_e}{\partial\theta_4}\,\underbrace{\dot{\theta}_4}_{=0} \\
&= \underbrace{\begin{bmatrix} \frac{\partial\, {}^sp_e}{\partial\theta_1} & \frac{\partial\, {}^sp_e}{\partial\theta_2} & \frac{\partial\, {}^sp_e}{\partial\theta_3} \end{bmatrix}}_{\bar{J}_v(q)} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \\
\Rightarrow \dot{q} &= \bar{J}_v^{-1}(q)\ {}^sv_e.
\end{aligned}
$$

Tips for computing $\bar{J}_v$ in MATLAB

- Define your joint variables θ_i as functions of time, so that you can differentiate them.

```
syms theta_1(t) theta_2(t) theta_3(t) theta_4(t)
A1 = createA(theta_1,'d_1',0,-pi/2)
```

- The homogeneous transform you'll obtain will be a 4×4 matrix function of t. To extract a particular entry of this matrix, you'll have to first evaluate it at a value of t and save it in an intermediate variable. For example, if `B` is a matrix symbolic function and you want to find matrix entry $(1, 2)$, then use:

```
tempVar = B(t);
entry = tempVar(1,2);
```

- You can find derivative of a symbolic expression using the MATLAB function `diff`. For example, `diff(f,x)` computes $\frac{\partial f}{\partial x}$.
- To make your life easier, remember chain rule.

Definition 5.4.1: RRMC Algorithm

Let the expression for determining $\dot{q}$ be $\dot{q} = g(q, v_e)$. Then,

$$
\begin{aligned}
\dot{q}_k &= g\left(q_k, v_e\right) \\
q_{k+1} &= q_k + \dot{q}_k \Delta t
\end{aligned}
$$

[4]

Task 5.6 Desired joint angles (20 points)

Provide a MATLAB `function q = makeQ(ve,q0,dt,T)` that applies the RRMC algorithm to determine the desired joint angles vector, q, when it is provided a desired end-effector velocity, v_e, the current joints position vector, q_0, the time step, Δt, and the final time, T. It's a good idea to first determine the symbolic expression for g and save it to be computed at any q.

5.4.1 Controlling the arm

You will now write a helper function to control the physical arm's joint angles.

Task 5.7 Communicating with motors (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 servo joint angles, 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 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.

Task 5.8 Executing straight-line motion (20 points)

Write a MATLAB function `function errorCode = pickObject(pose_obj)` that moves the end-effector from its current configuration to the pre-grasp pose in a straight-line motion. Be mindful of the following:

- Choose a small value for the speed of motion first, e.g. 0.01 m/s, and then gradually increase it. This is because we have not yet derived the maximum possible end-effector speed from the maximum possible joint speeds.
- Choose a reasonably big timestep first, e.g. $\Delta t = 3s$, and then test the effects of decreasing it. If the timestep is too small, then you're sending frequent updates to the Arbotix controller and clogging bandwidth and compute; if the timestep is too big, then the joints slow down to reach q_k and then again accelerate towards q_{k+1}, resulting in non-smooth or jerky motion.
- The task requires commands to be sent to the Arbotix controller at a fixed real rate. The MATLAB object `rateControl` can be used for such loops.
- Verify that no joint angle, θ_i, in the generated joint angles trajectory, q, exceeds the angular bounds of the servo motors. Also, verify that the joint trajectory is not passing through singularities. You may have to plot q.
- We've not yet implemented a collision-checker and the generated joint trajectory could result in collisions with the objects in the workspace, e.g. the base board, or

in self-collisions, i.e. a link colliding with another link. Be alert and immediately unplug the arm if it is heading towards a collision.

References

[1] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge University Press, 2017.

[2] R. Tedrake, *Robotic Manipulation, Perception, Planning, and Control*. 2024. [Online]. Available: http://manipulation.mit.edu.

[3] J. Haviland and P. Corke, "Manipulator differential kinematics: Part i: Kinematics, velocity, and applications," *IEEE Robotics & Automation Magazine*, 2023.

[4] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Transactions on man-machine systems*, vol. 10, no. 2, pp. 47–53, 1969.

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 projection of arm on the floor, i.e. $\phi = \theta_2 + \theta_3 + \theta_4$. When the gripper is parallel to the base board, then $\phi = 0°$[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 algebra or geometry 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 6.4.

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'll have to resolve the following issues:

- It may be usefult to write a helper function `checkJointLimits` to make sure that a solution is realizable. Recall that the motor limits are in the interval $[-150°, 150°]$.
- Furthermore, the `setPosition` function created by you in the previous chapter expexts servo angles that may be defined differently from the definition of θ_i used by you when deriving the inverse kinematics above. So, you may have to add offsets to your inverse kinematics solution to align the angle definition to the servo angle definition.
- It's a good idea to create 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.

6.3 Accuracy

Definition 6.3.1: Accuracy

This is the ability of a robot to position its end-effector at a preprogrammed location in space. Robot accuracy is important in the performance of non-repetitive types of tasks.

Typically, there is no direct measurement of the end-effector position and orientation, and instead one relies on the assumed geometry of manipulator and its rigidity to calculate end-effector position from measured joint positions (we install sensors to measure joint angles). Therefore, accuracy is affected by computational errors, machining accuracy in the construction of manipulator, flexibility effects such as bending of the links, gear backlash, other static and dynamic effects, and the accuracy of the solution routine.

Task 6.4 Determining accuracy (10 points)

In this task, we'll determine the positioning accuracy at different points in the workspace. Accuracy at a point, p, in the workspace is specified in terms of a length, which is the radius of a sphere centered at p. The robot is accurate enough if it will always land in that sphere when instructed to reach p in the workspace.

1. Randomly select five points (x, y, z, ϕ) in the workspace of the robot manipulator. Note that these will be in physical units, e.g., mm, cm, etc.
2. Execute the optimal solution for each point, as determined by your `findOptimalSolution` function. We've not yet implemented a collision-checker and the generated joint trajectory could result in collisions with the objects in the workspace, e.g. the base board, or in self-collisions, i.e. a link colliding with another link. Be alert and immediately unplug the arm if it is heading towards a collision.
3. Physically measure the actual location of the end-effector, and note it down.
4. Compute the absolute Euclidean error between the specified location and actual achieved location, and find the mean absolute error over all samples, which represents accuracy of the robot. Don't forget the units.
5. If there is any error, what are the possible source(s) of error?

You are to provide your data in tabular form for global accuracy calculation in 4.

6.4 Sketch for deriving IK expressions

6.4.1 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

6.4.2 Analytical Approach

Say the obtained forward kinematics expressions are:

$$\begin{aligned}
x &= \cos\theta_1 \left[a_2 \cos\theta_2 + a_3 \cos(\theta_2 + \theta_3) + a_4 \cos(\theta_2 + \theta_3 + \theta_4)\right] \\
y &= \sin\theta_1 \left[a_2 \cos\theta_2 + a_3 \cos(\theta_2 + \theta_3) + a_4 \cos(\theta_2 + \theta_3 + \theta_4)\right] \\
z &= d_1 + a_2 \sin\theta_2 + a_3 \sin(\theta_2 + \theta_3) + a_4 \sin(\theta_2 + \theta_3 + \theta_4) \\
\phi &= \theta_2 + \theta_3 + \theta_4
\end{aligned}$$

From these expressions, it is clear that

$$\begin{aligned}
{}^1\theta_1 &= \arctan 2(y, x) \\
{}^2\theta_1 &= \arctan 2(-y, -x)
\end{aligned}$$

Let

$$\begin{aligned}
\bar{x} &= \frac{x}{\cos\theta_1} - a_4 \cos\phi \\
\bar{y} &= \frac{y}{\sin\theta_1} - a_4 \cos\phi \\
\bar{z} &= z - d_1 - a_4 \sin\phi
\end{aligned}$$

Then,

$$\begin{aligned}
\bar{x}^2 + \bar{z}^2 &= a_2^2 + 2a_2a_3\cos\theta_3 + a_3^2 \\
\Rightarrow \cos\theta_3 &= \frac{\bar{x}^2 + \bar{z}^2 - a_2^2 - a_3^2}{2a_2a_3} = u \\
\Rightarrow {}^{a,b}\theta_3 &= \arctan 2(\pm\sqrt{1-u^2}, u)
\end{aligned}$$

For θ_2 then,

$$\begin{aligned}
\cos\theta_2 &= \frac{\bar{y}(a_2 + a_3u) \pm \bar{z}(a_3\sqrt{1-u^2})}{a_2^2 + 2a_2a_3u + a_3^2} \\
\sin\theta_2 &= \frac{\bar{z}(a_2 + a_3u) \mp \bar{y}(a_3\sqrt{1-u^2})}{a_2^2 + 2a_2a_3u + a_3^2} \\
\Rightarrow {}^{a,b}\theta_2 &= \arctan 2(\sin\theta_2, \cos\theta_2)
\end{aligned}$$

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.

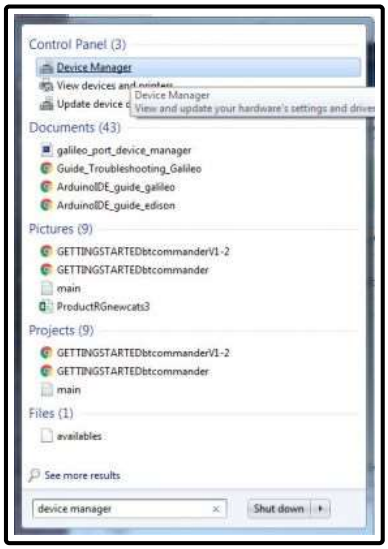
Click here to expand Windows Instruction Click here to expand Mac OS X Instruction Click here to expand Linux Instruction

Windows

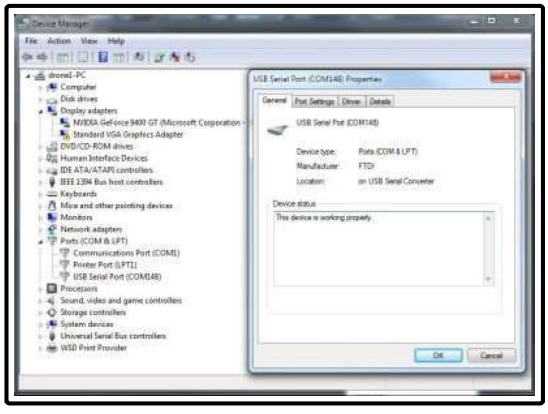
Now you will need to install FTDI drivers. These drivers will allow you to communicate with your ArbotiX-M via the USB port. Some modern Operating Systems either have these drivers or ca automatically find them. To install these drivers:

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.

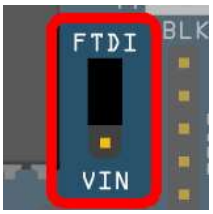
Back t

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.

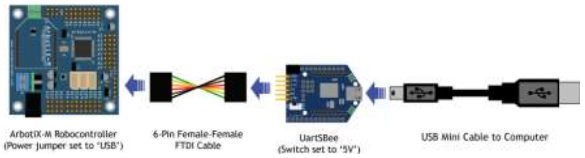
- In this example we will power the ArbotiX-M from the FTDI port. Move the power jumper so that it connects the middle pin and the 'USB' pin ('FTDI' for the original ArbotiX).

Power Jumper for ArbotiX-M

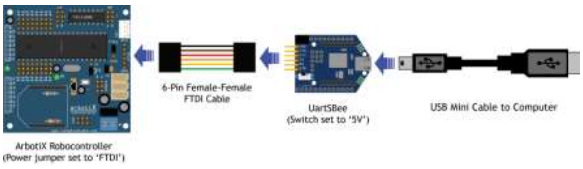


Power Jumper for ArbotiX

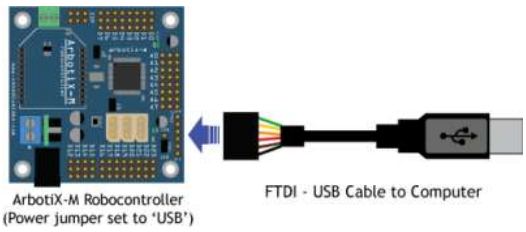
- The orientation of the FTDI cable is very important - it is possible to plug in the cable backwards and you will not be able to program your board. The top FTDI pin with the mark 'BLK' always connect to the black FTDI cable. This pin is a ground or 'GND' pin. The bottom pin with the mark 'GRN' will 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



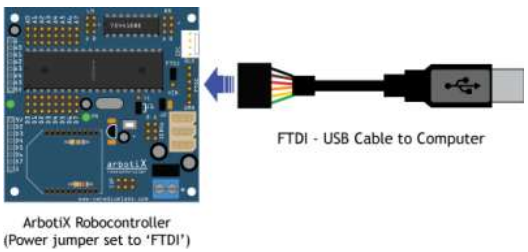
ArbotiX-M to UartSBee Connection



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!

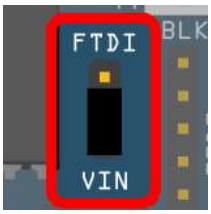
If you see any red text or error messages, then the sketch has not been loaded properly. Make sure you have installed the FTDI drivers, chosen the correct board and seria and that your USB connection is secure.

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.

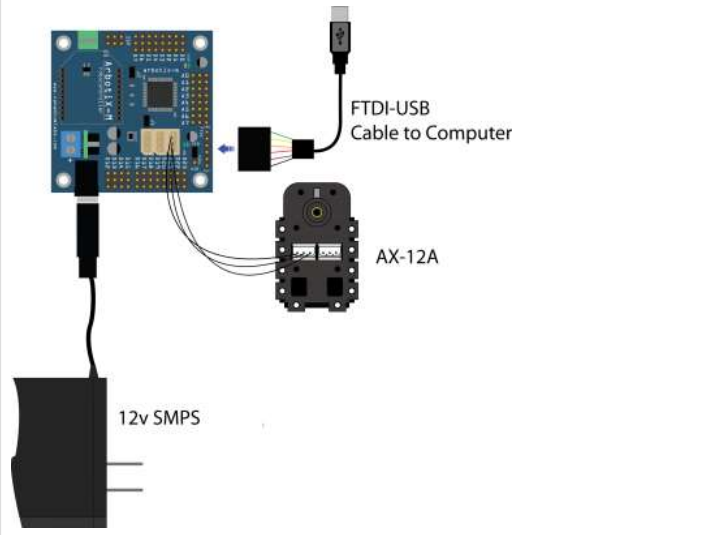
Power Jumper for ArbotiX-M



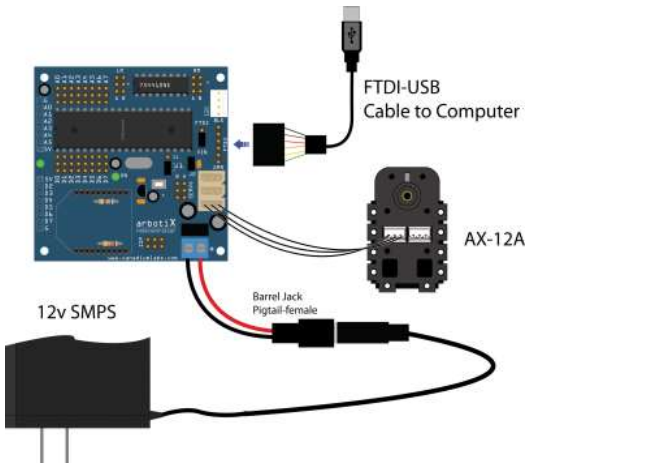
Power Jumper for ArbotiX

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

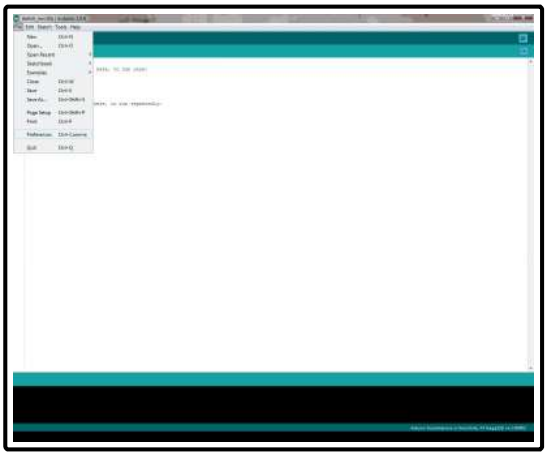
If you are working with MX servos, we reccomend you test them in the next step, Setting DYNAMIXEL IDs with the DynaManager. You can also use `File -> Examples -> A -> Test Sketchs -> MXSimpleTest`, though you may need to make some adjustments to the baud rate.

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
```

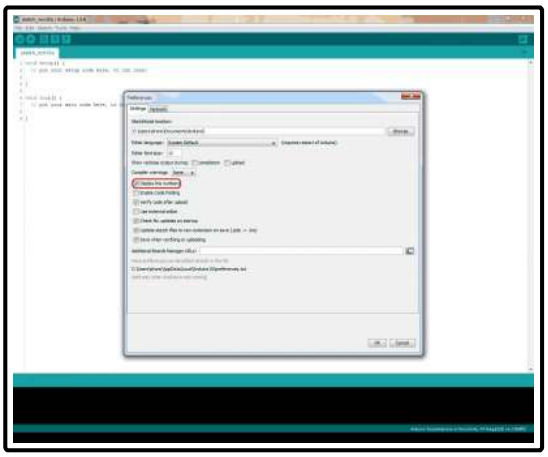
Mac users will find their preferences under `Arduino -> Preferences`



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 ➔.

CHAPTER B

Setting up ArmLink

1. 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 B.2; the port on the board also indicates which side corresponds to the green cable and which side to the black.

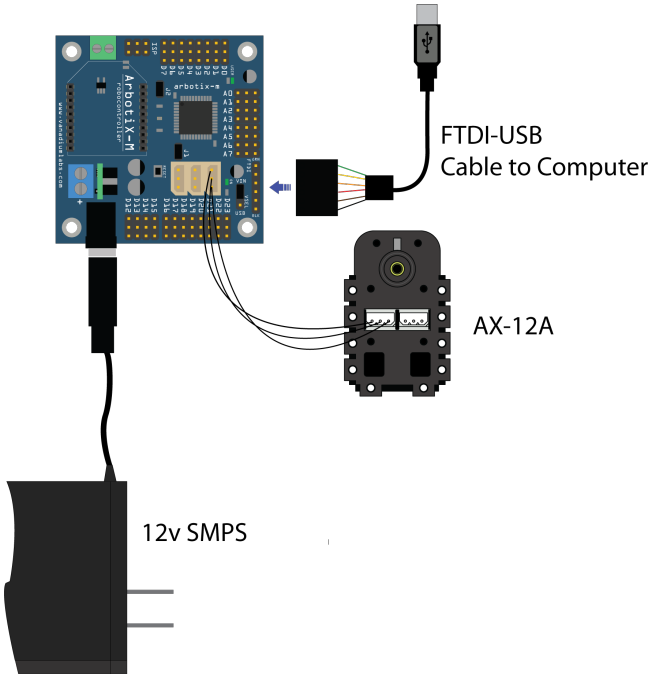


Figure B.1: Setting up the Arbotix-M with power, servos, and programming

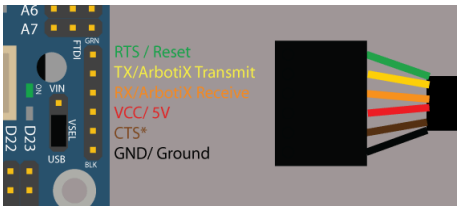


Figure B.2: FTDI-USB Cable Connection

2. Connect the servomotors to the Dynamixel servo ports (See Figure 1.4).

3. Connect the power jack to Arbotix-M. 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.

4. 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.

5. 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

on your computer[1]. 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.

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

7. 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 on the display. **Once the arm is connected to the software, don't try to move it by hand as each of the motors will exert torque.**

8. 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.

[1]The required firmware files should be copied to `Documents/Arduino` folder on Windows as indicated in Appendix-A.

CHAPTER

Setting up Peter Corke's Robotics Toolbox

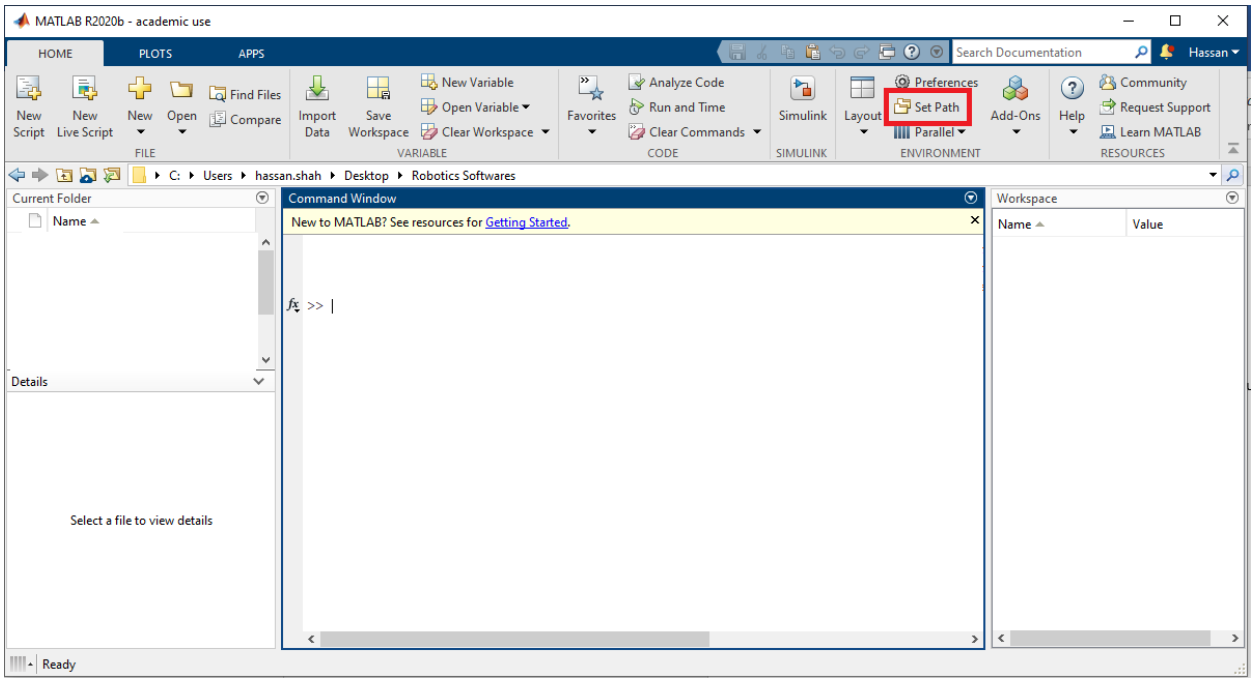
Install Tool Box

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

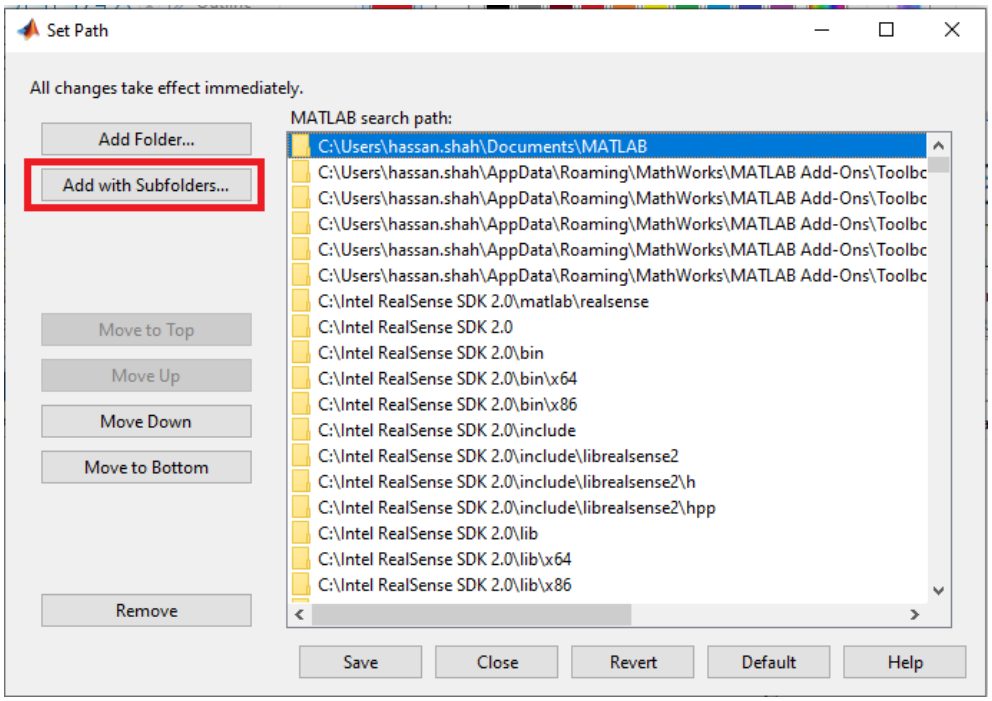
2. Navigate to the ".mltbx" file from file explorer in matlab and then double click to install.

Add Path

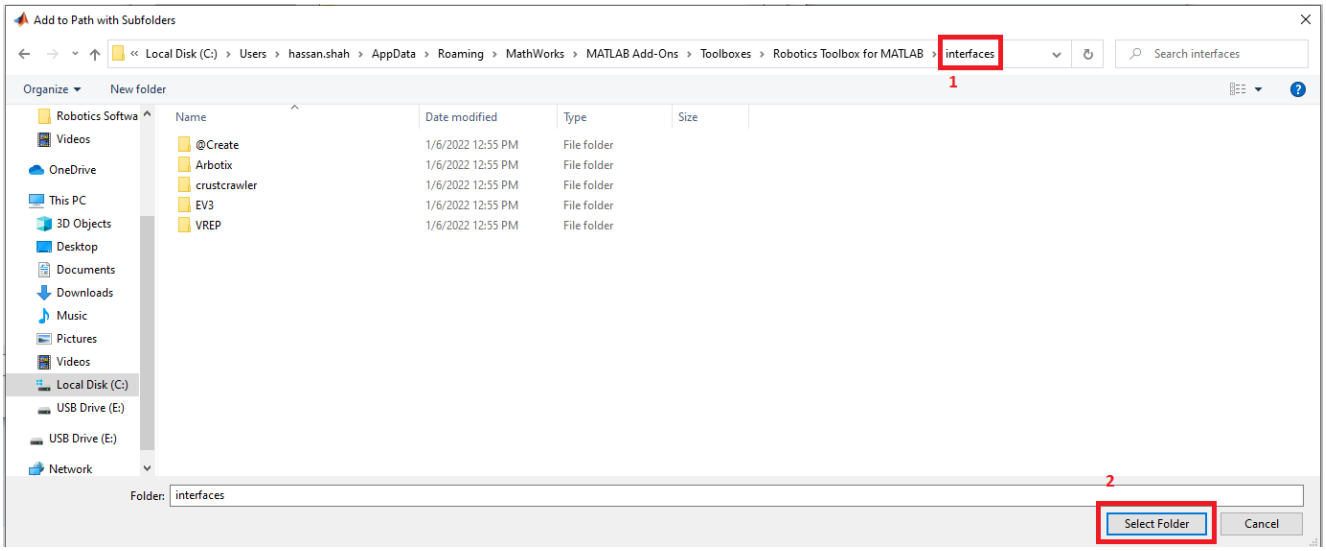
3. Click on Set Path.



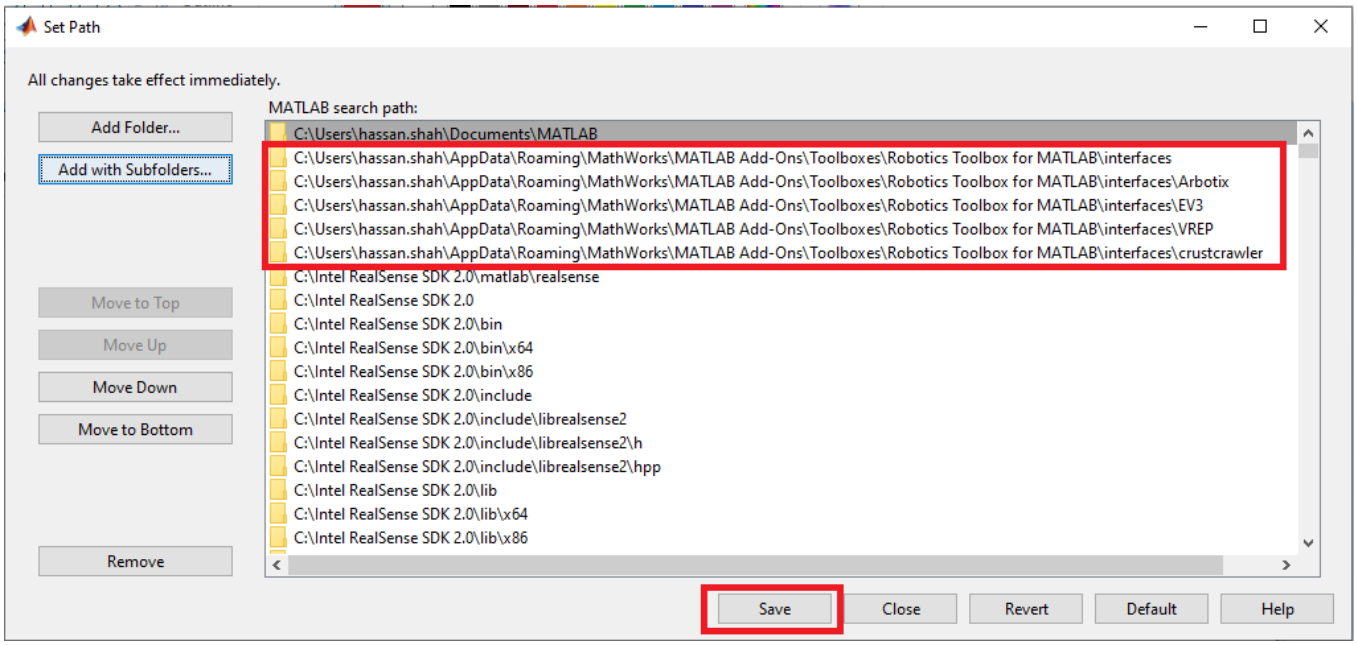
4. Click on Add with subfolder.



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.

CHAPTER D

The Standard DH Convention

Definition D.0.1: 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.

Definition D.0.2: 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.

Definition D.0.3: Homogeneous Transformation using DH parameters

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

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