EE366L/CE366L: Introduction to Robotics

Lab Handbook

Second Edition, Spring 2023

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 will recommend you to acquire a passing familiarity with it if you 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 Zaid and Mr. Hassan Shah, who were RAs in DSSE, for their assistance in setting up the robotics lab. I would also like to thank the students enrolled in 2022 who volunteered their time to assemble the robot arms during the winter break. It would not have been possible to conduct this lab timely, otherwise.

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

CHAPTER 0

Lab Instructions

1. The credentials for signing in to the lab computers are:

   ```
   Username: robo
   Password: Habib123++
   ```

2. Since you're working in groups, rotate among yourselves so that each member of the group has a chance to get hands-on experience.

3. Tasks marked with an [*] don't require access to the hardware and can be completed post-lab.

4. At the end of the lab, you're required to unplug the arm from the power supply and turn off the lab computer's monitor. The arm may relax and fall down as it is turned off, so be sure to catch it and carefully place it on the baseboard.

5. The weekly lab findings report is a group submission.

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

- Carl Friedrich Gauss

CHAPTER 1

Getting familiar with MATLAB

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

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

Task 1.1 Team Formation (10 points)

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

Task 1.2 MATLAB On-Ramp (90 points)

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

"If science is to progress, what we need is the ability to experiment."

– Richard P. Feynman

CHAPTER 2

Getting familiar with the arm hardware

The objective of this lab is to get yourself familiar with the hardware and capabilities of the Phantom X Pincher arm, shown in Figure 2.1 and manufactured by Trossen Robotics (https://www.trossenrobotics.com), which will be utilized for this project. During the course of this lab, you'll also get a sense of the motion and manipulation limitations of this arm and become familiar with the variables used to measure performance. 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 the same dimensions as physical arm, and also allows you to measure the distances and angles at different points on the robot.

2.1 Design of the arm

The arm has 5 motors that serve as actuators, a pinch grasper, and a microcontroller board at the bottom. We'll discuss each of these parts in greater detail later in this lab. 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 by hand when it is powered up, as the motors will be providing torque and it will resist motion, and you may damage the arm.**

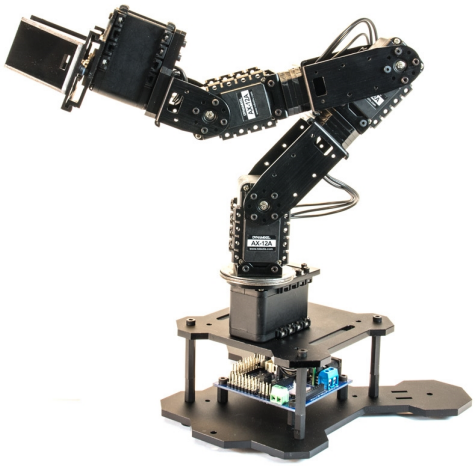


Figure 2.1: Phantom X Pincher Arm

Task 2.1 Model of the arm (20 points)

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

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

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

(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.*

2.2 The Arbotix-M Controller

The Arbotix-M controller, depicted in Figure 2.3, lies at the heart of this arm. This is an informational section that provides relevant details of this controller. This controller receives higher level instructions from an external processing unit and generates instructions in specific 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 MATLAB to Arbotix-M, as shown in Figure 2.2.

This board is capable of doing more than controlling the servomotors; some of these capabilities are outlined in this section. If you were to take out the Arbotix-M board and study it, then Figure 2.4 would depict the purpose of the various sections on this board. All the processing is actually done on this small square-shaped chip in the middle, specifically by an ATMEGA644p AVR

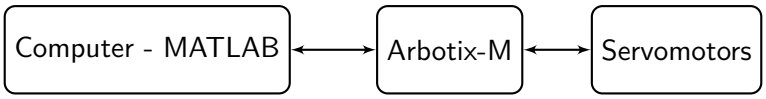


Figure 2.2: Data flow between MATLAB and servomotors

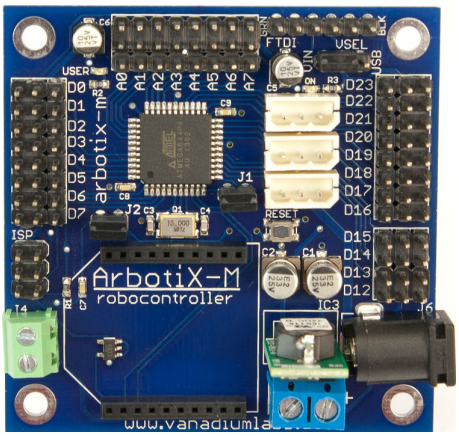


Figure 2.3: The Arbotix-M Controller

microcontroller, which is in the same category of microcontrollers as an Arduino. In fact, we'll be using the Arduino IDE to program the Arbotix-M in C. For the interested, this page https://learn.trossenrobotics.com/arbotix/arbotix-getting-started/38-arbotix-m-hardware-overview provides further hardware details.

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

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

In addition to this, the required firmware files should be copied to `Documents\Arduino` folder on Windows as indicated in [4]. We'll verify this in 2.3.1.

2.2.1 Connecting motors to Arbotix-M

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

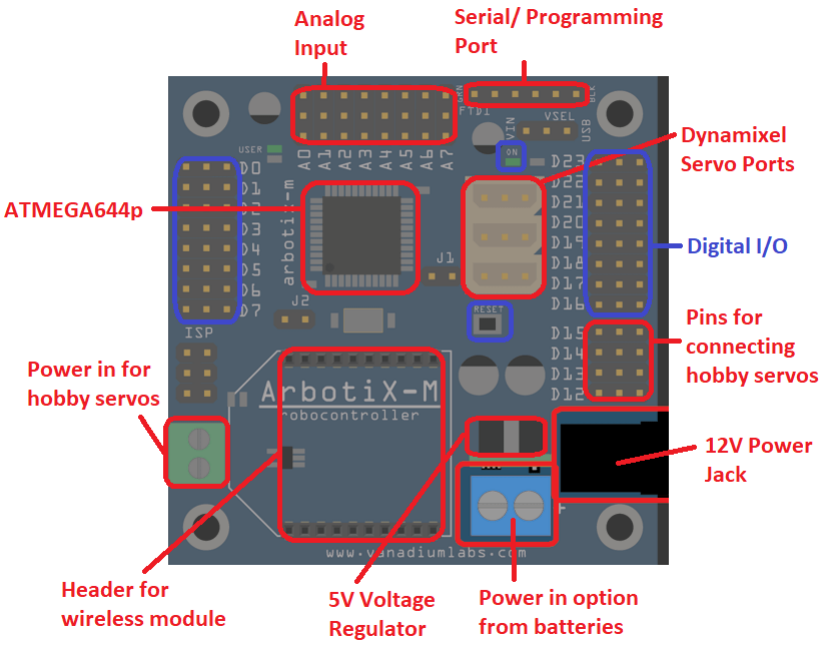


Figure 2.4: Getting to know Arbotix-M

2.3 Move the arm

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

2.3.1 Setting up Armlink

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

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

3. We'll now need to load the appropriate firmware on Arbotix-M that will allow it to communicate with the Armlink software, assuming that the firmware is appropriately placed on your computer as specified in [4]. 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)
   ```

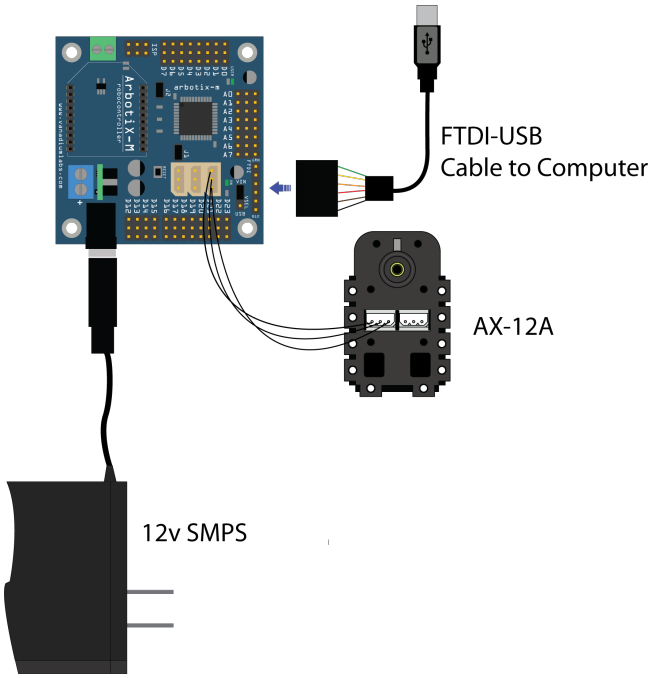


Figure 2.5: Setting up the Arbotix-M with power, servos, and programming

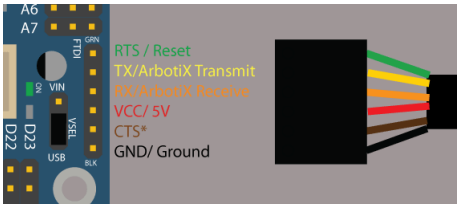


Figure 2.6: FTDI-USB Cable Connection

Figure 2.7: IDs of the servos in arm

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

File -> Examples -> Arm Link -> InterbotixArmLinkSerial

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

```
#define ARMTYPE PINCHER
```

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

Sketch -> Upload

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

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

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

6. You can adjust the sliders or text panels to adjust the positions of the arm. You can send these values to Arbotix-M by clicking on Update, or you can check Auto Update, in which case instructions will be sent continuously.

2.3.2 Getting familiar with Arm Link Controls

Read through this and the next section to be fully aware of safety instructions before playing around with the controls. Change the panel coordinates 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 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.

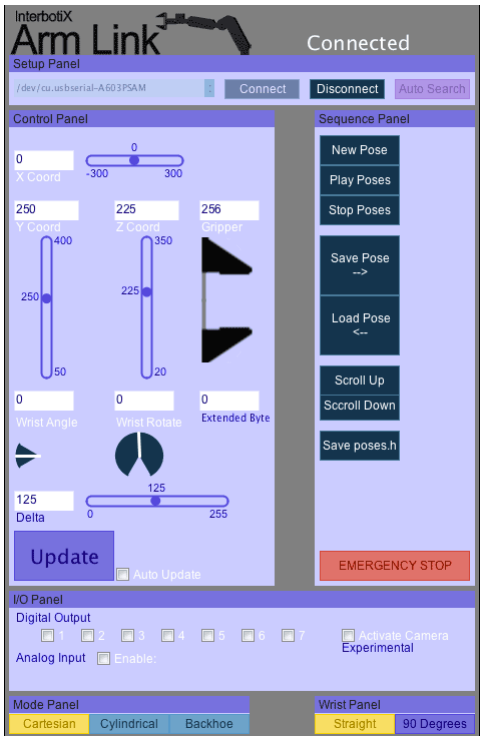


Figure 2.8: Arm Link Application Panel

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.

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.

For further details on the controls, refer to [3].

2.3.3 Common Problems

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

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

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

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

Task 2.2 Configurations Exploration (20 points)

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

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

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

(c) Grab one each of the provided objects. In this task, you'll place each of these objects at a fixed location in your workspace, move the arm using `ArmLink` to that location, pick the object, and place it at another location. During this activity, how is the real world environment being sensed and how is the arm motion being adjusted based on the received sensing data? Where is this processing happening?

(d) [*][a] The coordinates in the Cartesian or Cylindrical mode describe task space locations. Task space can be used to describe tasks to be carried out by the manipulator, e.g. grabbing a water bottle. Give an example of a task that can be described better in Cartesian coordinates, and a task, which is best expressed in cylindrical coordinates.

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

2.4 Establishing a coordinate system

The design of our autonomous pick and place pipeline will also be based on a vision-based feedback loop, similar to the one determined by you in the previous task. A camera will take the place of our eyes in this loop. This loop is formally termed as visual servoing[1]. There are two ways for visual servoing:

1. **Position-based visual servoing:** Coordinates of positions of interest are determined from the image and the robot motion controller moves the arm to desired positions.

2. **Image-based visual servoing:** This marks the positions of interest and robot positions in the image and the robot motion controller moves the arm so that robot is at the desired position inside the image.

In this project, we'll employ position-based visual servoing. To do so, we'll have to agree on a global coordinate system for specifying all positions. The numbers appearing in the Arm Link panels for X, Y, Z values appear to map the physically possible range along an axis onto a subset of integers, e.g. X, onto a subset of integers between 0 and 1023[2]. So, this must be based on an underlying coordinate system and we could use the same in our design.
inform resolution is 1:1 identify the unit where's the end point?

Task 2.3 Coordinate Axes (10 points)

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

(b) We'll set the origin of the x and y axes at the center of the shaft of the first motor, and the origin of the z axis at the level of the wood platform. If 1 unit in the ArmLink system corresponds to 1 unit in the real world, identify the units being utilized in the real world and the point of the arm whose position is being determined.

2.5 Pose Accuracy and Repeatability

Two parameters that typically characterize the performance of a robot are its pose accuracy and pose resolution. ISO 9283:1998 defines both these characteristics for industrial arms and the methods for measuring them. In this section, you'll gain familiarity with both these characteristics and determine them for the robot arm at hand.

[1] The term servo has a specified meaning in engineering literature, which will be defined later in the current chapter.

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

Pose Accuracy

It expresses the deviation between a command pose and the mean of the attained poses when approaching the command pose from the same direction. It is divided into positioning accuracy (AP_P) and orientation accuracy (AP_O).

The positioning accuracy is calculated as follows:

$$AP_P = \sqrt{(\bar{x} - x_c)^2 + (\bar{y} - y_c)^2 + (\bar{z} - z_c)^2}$$

where

$$\bar{x} = \frac{1}{n}\sum_{j=1}^{n} x_j$$
$$\bar{y} = \frac{1}{n}\sum_{j=1}^{n} y_j$$
$$\bar{z} = \frac{1}{n}\sum_{j=1}^{n} z_j,$$

(x_c, y_c, z_c) are coordinates of the command pose, (x_j, y_j, z_j) are the coordinates of the j-th attained pose, and n is the number of times the same command is given.

Pose Repeatability

It expresses the closeness of agreement between the attained poses after n repeat visits to the same command pose in the same direction. The value of position repeatability is the radius of sphere enclosing the set of locations the manipulator achieves for the same command with the same load and setup conditions.

The positioning repeatability RP_l can be calculated as:

$$RP_l = \bar{l} + 3S_l,$$

where

$$\bar{l} = \frac{1}{n}\sum_{j=1}^{n} l_j$$
$$l_j = \sqrt{(x_j - \bar{x})^2 + (y_j - \bar{y})^2 + (z_j - \bar{z})^2}$$
$$S_l = \sqrt{\frac{\sum_{j=1}^{n}(l_j - \bar{l})^2}{n-1}},$$

with $(\bar{x}, \bar{y}, \bar{z})$, (x_j, y_j, z_j), and n are as defined previously.

Figure 2.9 highlights four different repeatability and accuracy combinations in 2D. For position-based visual servoing (the one we're employing), 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

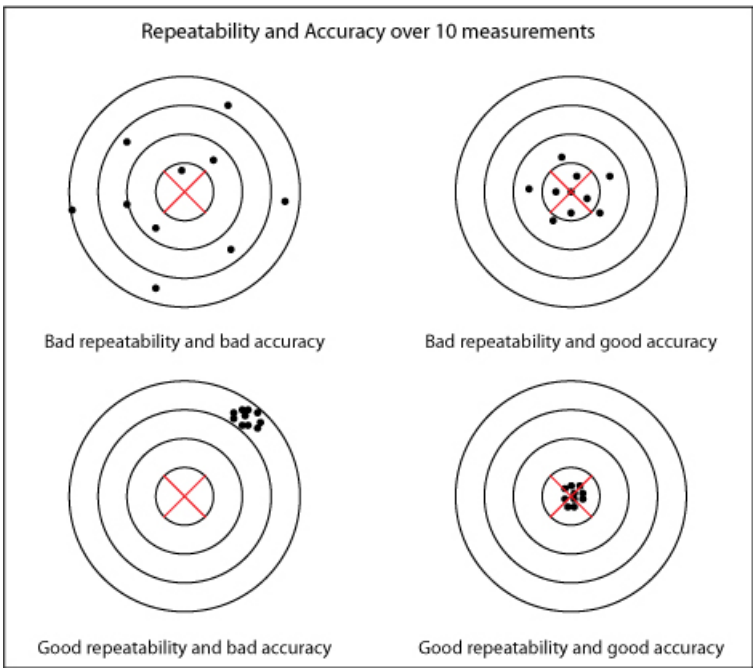


Figure 2.9: Red cross indicates commanded pose and the black dots achieved poses over multiple iterations of the same command.

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. Repeatability depends on the accuracy and resolution.

The standard also provides a procedure for measuring the two characteristics. Some key points from the procedure are:

- The wrist reference point is considered as the endpoint for positioning accuracy and repeatability.
- The trials are to be performed at 100% of rated velocity and 100% of the rated load.
- n should be 30.
- Te robot passes through five poses P_1 through P_5, which are located on the diagonals of the larges cube that the robot can achieve.

For more details, read pages 10-23 in the standard document uploaded on LMS.

Task 2.4 Accuracy and Repeatability (50 points)

Identify a method for physically measuring the position of the wrist reference point of a robot arm.

(a) Design an experiment for determining the accuracy and repeatability of the robot arm and provide details of this experiment.

(b) Conduct sufficient trials of this experiment and determine values for positioning

accuracy and repeatability.

(c) Does the accuracy of this robot vary with distance from the base? How will you determine it?

(d) [*] Research the positioning accuracy and repeatability of industrial arms and compare the values with the ones obtained in the previous part. How do they compare?

(e) [*] If the accuracy and/or repeatability of our robot arm is lower, how will it impact the performance of our pick and place pipeline?

2.6 Teaching Pendant

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

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

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

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

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

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

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

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

Task 2.5 Bonus (20 points)

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

2.7 References

1. Manufacturer provided arm resources: https://learn.trossenrobotics.com/
2. Arbotix-M Controller resources https://www.trossenrobotics.com/p/arbotix-robot-controller.aspx
3. ArmLink: https://learn.trossenrobotics.com/20-interbotix/robot-arms/137-interbotix-arm.html
4. Getting started with Arbotix-M https://learn.trossenrobotics.com/arbotix/7-arbotix-quick-st step3

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

– Elton John

3

CHAPTER

Sensing and actuation for the arm

The objectives of this lab are to:

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

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

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

Definition 3.0.1: Sensors

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

Definition 3.0.2: Actuators

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

3.1 Construction of Vision-based Pick and Place Robot

In the last lab, you carried out a manual pick and place operation, identifying the position of the gripper and the object through your eyes and then correspondingly adjusting the gripper

position using manual control through the provided interfacing software. However, the output of your project will be an autonomous vision-based pick and place robot, for which a number of questions will have to be answered:

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

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

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

3.1.1 Servomotors

A servomotor is a regular DC motor coupled with sensing, to measure the rotational position of the output shaft, and a controller, which uses that position feedback to precisely control the position of the motor. Advanced servos can also control the motor's angular velocity and acceleration. This feedback loop is illustrated in the block diagram in Figure 3.2.

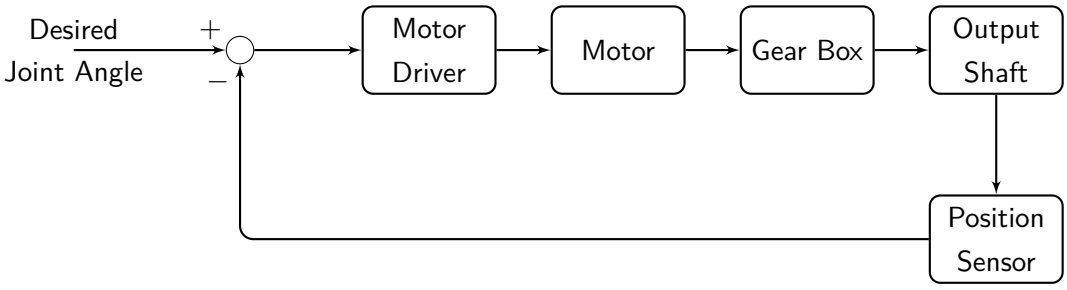


Figure 3.2: Block diagram of a servomotor

Our robot arm has five Dynamixel AX-12A servomotors (https://www.robotis.us/dynamixel-ax-12a/). The functional components highlighted in the previous block diagram are captured in Figure 3.3 (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/lv-vgnHDV68.

The controller on the Dynamixel AX-12A motors has a set instruction format for its read and write instructions used for reading and writing the values of its registers. The structure to

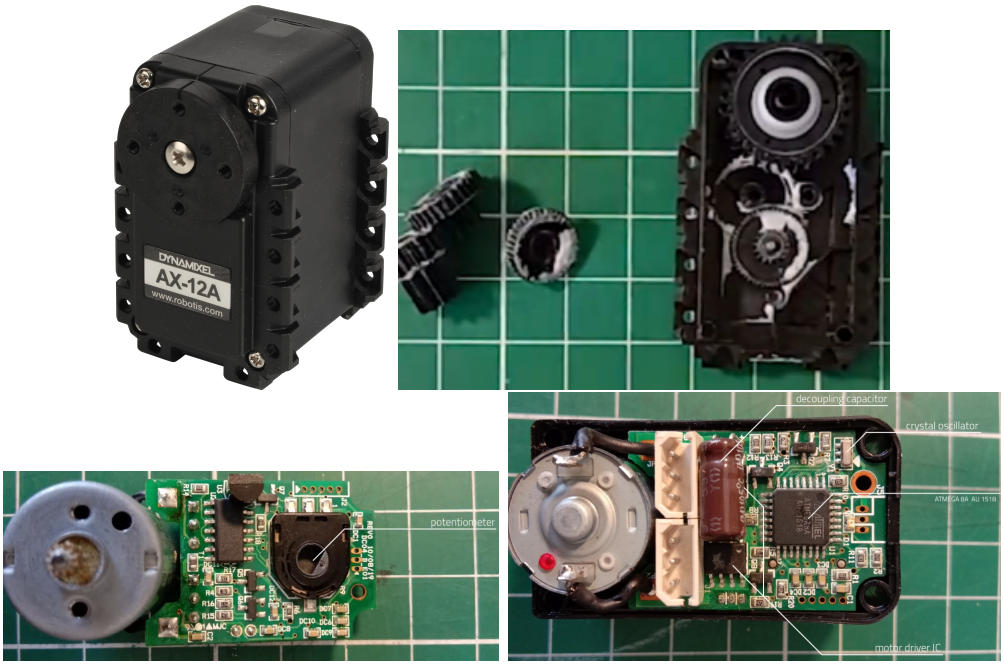


Figure 3.3: Top Left: AX-12A Motor, Top Right: Gear Box, Bottom Left: Potentiometer, Bottom Right: Microcontroller and Motor Driver

be followed in every instruction message packet is completely outlined in Dynamixel reference manual [1]. The Arbotix-M plays the role of an intermediary, allowing us to use its easier C libraries for setting and getting the position of each motor, while the libraries translate our instructions to the packet structure required by AX-12A motors.

Task 3.1 Understanding functioning of system (10 points)

(a) [*] Identify and list all the sensors and actuators in our complete robotic system illustrated in Figure 3.1.

(b) [*] Figure 3.3 suggests that a potentiometer is being used as the shaft position sensor. Research and describe how can a potentiometer be used for this purpose.

Task 3.2 Motor Specifications (10 points)

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

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

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

[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 3.1.1: 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.

3.2 Camera

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

3.2.1 Working with SR-305 Camera

You'll be using Intel RealSense SR305 camera, shown in Figure 3.4, 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. A number of ways are being used in practice today to calculate depth, e.g. stereo images, time

Figure 3.4: Intel RealSense SR305 Camera

of flight, etc. The SR305 works on the principle of coded light. In addition to a regular RGB camera, an SR305 has an IR (Infrared) camera and an IR projector, as seen in Figure 3.5. An emitter projects patterned light, typically a pattern of vertical bars, on to a scene, as illustrated in Figure 3.6. 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 [3].

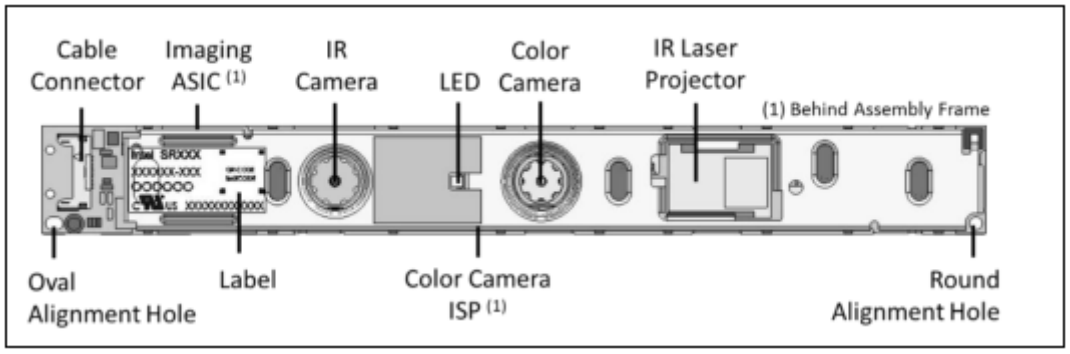


Figure 3.5: Components of an SR-305 Camera

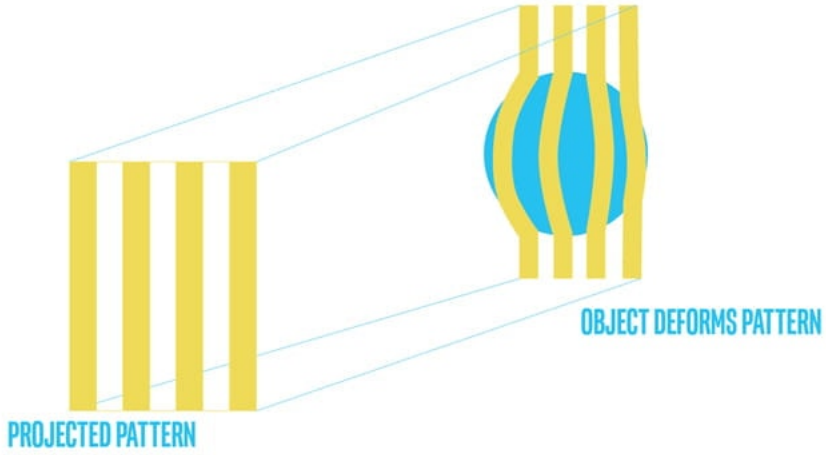


Figure 3.6: Principle of coded light

Task 3.3 Getting to know the camera (0 points)

Download `Intel RealSense Viewer` tool from Canvas to verify that your camera is working and to explore the various parameters. If you enable both the RGB and depth streams, you shall see live videos for both where the depth stream represents different depths in different colors. Hover over any pixel in the depth image and you shall see the depth value in meters at the bottom. Explore the different processing filters available for each stream. Details of filters are provided at [6].

3.2.2 Image manipulation in MATLAB

The images viewed in `Intel RealSense Viewer` in the previous section will be imported in MATLAB so that objects of interest can be identified and their position can be determined using machine vision algorithms. The following task requires you to gain familiarity with representing and manipulating images, and common image processing operations in MATLAB.

Task 3.4 Image Manipulation in MATLAB (60 points)

Complete at least modules 1-4 of the 'Image Processing OnRamp' (https://matlabacademy.mathworks.com/details/image-processing-onramp/imageprocessing) and provide your 'Progress Report' as submission for this task.

3.2.3 Obtaining SR305 images in MATLAB

Intel provides an SDK ([4]) for its RealSense line of cameras. The SDK includes a MATLAB wrapper too and we'll be making use of it in our project. Install the SDK by using the provided installer. Make sure to check the `MATLAB Developer Package` during installation. The package will be installed to `C:\Program Files (x86)\Intel RealSense SDK 2.0\matlab\+realsense\`.

Task 3.5 Extract color and depth images (10 points)

The MATLAB function `depth_example()`, located in the folder where the SDK is installed, provides the code to extract a depth image. Verify that the code works and make sense of the provided code.

Modify this code so that it also extracts a color frame and displays it. You can do so by using the function `get_color_frame()`. Don't forget to reference the appropriate class instance and format the received frame before displaying.

Task 3.6 Reflection (10 points)

Based on your experiences with the robot arm in the current and previous session, write a brief note outlining your takeaways and any unanswered questions about the vision-based pick and place robotic system that we intend to develop as our first project.

3.3 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 3.1 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 3.1: 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.

3.3.1 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 3.7, 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

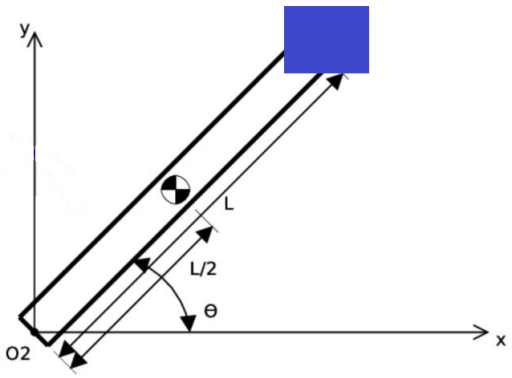


Figure 3.7: One link arm

M_L. Then the torque exerted by the load at any position of the arm is given by

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

where the first term is due to the weight of the load and the second term because of the weight of the link itself. It is assumed that the center of mass of the link is at its mid-point. A motor installed at joint $O2$ will have to exert the same amount of torque to maintain the load at its current position. Note that we're doing this calculation at static equilibrium, i.e. to achieve zero net generalized forces on the arm, when it is not in motion. Since we're currently only interested in the maximum torque required from the motor, we can simply consider the worst-case position 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 3.8. The lengths and masses 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

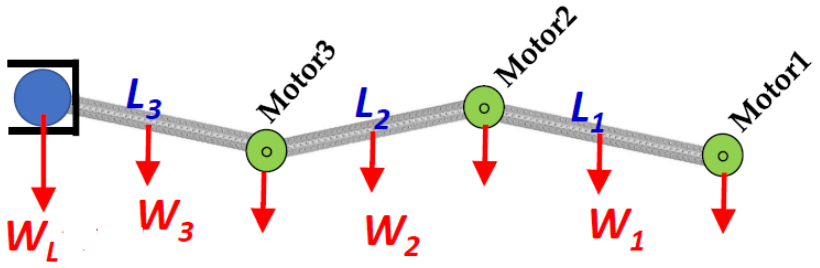


Figure 3.8: Multi-link robot arm

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.

3.3.2 Gripping Force

In this section, we'll review the physics behind the calculation for the gripping force needed to grasp an object. These are again calculations for the condition of static equilibrium. Assume that we have a parallel gripper, as shown in Figure 3.9. Let F be the force being exerted by the

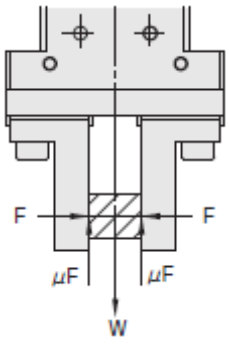


Figure 3.9: Force required to grip an object

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

to the weight of the object, i.e.

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

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

Task 3.7 Bonus (10 points)

According to the manufacturer's provided heuristic [2], the load on any motor should not exceed $1/5$ of the stall torque. Keeping this heuristic, motor specifications, mass measurements in Figure 3.10, and physical principles outlined in Section 3.3.1, verify the wrist lift strength specified by the manufacturer. The wrist lift strength is the load that the wrist joint can support.

Figure 3.10: Masses of different arm components

3.4 References

1. Dynamixel Reference Manual (https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/)
2. AX-12A motor load heuristic (http://en.robotis.com/model/board.php?bo_table=robotis_faq_en&wr_id=33&sca=GENERAL)
3. Zabatani, Aviad, et al. "Intel realsense sr300 coded light depth camera." IEEE transactions on pattern analysis and machine intelligence 42.10 (2019): 2333-2345.
4. https://github.com/IntelRealSense/librealsense
5. Intel RealSense Depth Camera SR300 Series Product Family Datasheet
6. https://github.com/IntelRealSense/librealsense/blob/master/doc/post-processing-filter.md

7. https://github.com/IntelRealSense/librealsense/wiki/Projection-in-RealSense-SDK-2.0

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

Kinematics refers to the geometric and time-based properties of the motion of an object without considering the forces and moments that cause the motion. In this lab, you will study this relationship in the context of position and orientation of end-effector with respect to joint angles. This aspect of kinematics is called forward position kinematics. You'll follow the DH convention to assign frames to our manipulator, determine DH parameters, and the homogeneous transformation from the base frame to the end-effector. Having established the forward kinematics mapping, you'll verify its accuracy physically and use it to determine the workspace of the manipulator. Your forward kinematics function will be carried forward with you throughout the future labs. You'll use MATLAB to write you kinematic function as well as to control the arm.

4.1 Determination of forward kinematic mapping

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

Definition 4.1.1: Standard DH Convention

1. Links numbered from 0 to n, and joints 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-hand frame.
6. The end-effector frame, n, can be arbitrarily chosen.

Task 4.1 DH Frame Assignment (15 points)

Using standard DH convention, assign DH frames to the robot arm in Figure 4.1. Make sure to clearly indicate the z and x axes, and the origin of each frame; drawing the y axis is optional. Place the origin of the end-effector frame at the center of gripper motor horn, for convenience of measurements in upcoming tasks.

Definition 4.1.2: DH Parameters

The four DH parameters are:

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

Figure 4.1: DH Frame assignment

Task 4.2 DH Parameters (15 points)

Annotate Figure 4.1 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

Definition 4.1.3: Homogeneous Transformation using DH parameters

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

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

Task 4.3 Homogeneous Transformations (5+5+3 points)

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

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

- You can create a symbolic variable in MATLAB using `syms` function, e.g. `syms('theta_1')` will create a symbolic variable θ_1 in MATLAB. In case of a live script, the variable will also be displayed in Greek alphabet.
- The MATLAB functions `cos` and `sin` expect arguments in radians, while `cosd` and `sind` in degrees.

- Using the standard naming convention for the variables storing homogeneous transformations may result in convenience later, e.g. `T01` or `T_01`.

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

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

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

4.1.1 Building the functions on MATLAB

Task 4.4 FK Function (10 points)

Provide a MATLAB `function [x,y,z,R] = findPincher(jointAngles)` or `function [x,y,z,R,theta,phi] = findPincher(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 function definitions depends on whether you want to adopt the strategy for describing orientation outlined in the previous remark. You can also decide whether your function will accept arguments in degrees or radians, and whether you want to . You can find help on how to create MATLAB functions at [2] and [3].

Using the DH parameters, we can build a model of our manipulator in MATLAB as well. MATLAB treats robots in exactly the same way as we have done in class, i.e. a chain of joints and rigid bodies [4]. The provided MATLAB script file `pincherModel.m` follows the example in [5] to build a model for Phantom X Pincher.

Task 4.5 Verification of Forward Kinematic Mapping (5 points)

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

configuration graphically.

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

4.2 Identifying reachable workspace

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

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

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

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

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

Task 4.6 DH and Servo Joint Angles alignment (10 points)

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.

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

Task 4.7 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 `plot3` may be of help for this task.

4.3 Positioning the arm

qWe'll now control the robot from MATLAB and physically verify that our forward kinematics computations are correct. For this, we'll make use of the `Arbotix` library written by Peter Corke as part of his Robotics toolbox [6]. Follow the steps below to get the arm set up to communicate with MATLAB.

4.3.1 Setting up communication between MATLAB and arm

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

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

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

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

4.3.2 Library of Arbotix Functions

Almost all capabilities provided by the manufacturer in Dynamixel AX-12A motors have been captured in this MATLAB library. Following is a collection of methods available in the library and 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 [7]. 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.

4.3.3 Controlling the arm from MATLAB

You will now write a helper function to control the physical arm's joint angles. This function has no relation to the previous forward kinematics functions, and will be used in upcoming tasks and labs.

Task 4.8 Communicating with motors (12 points)

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

- The `jointAngles` vector contains joint angles, according to DH frame assignment,

in order from the base to the wrist. The angles should either be in radians or angles.

- Remember that the library method `arb.setpos` expects angles in radians.
- The angle limits for the motors are $[-150^\circ, 150^\circ]$, and your function should output an error and stop execution, if a provided joint angle is outside this limit.
- You will have to map the received angles θ_i to its corresponding value in $[-\pi, \pi]$ to compare against the allowed joint limits and pass them on to the motors. One way to find the equivalent value of θ in the interval $[-\pi, \pi]$ is `angle = mod(theta+`π`,2`π`)-`π.
- In order to provide the correct input to the motors, you need to find out the appropriate mapping function, $\psi_i = g(\theta_i)$, based on Table 4.2.
- 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 4.9 Verification of positioning (10 points)

1. Select 5 random joint configurations for the manipulator;
2. Determine the end-effector position and orientation from `findPincher` function;
3. If it appears that the arm will self-collide in the resulting image for your configuration, obtained from `findPincher`, or the obtained end-effector position is too close to the baseboard (less than 95mm), then change chosen joint angles. This is a safety measure to prevent any collisions due to errors in your mappings.
4. Use your `setPosition` function to move the end-effector to each of the five chosen configurations.
5. For each configuration, physically measure and note the actual end-effector position.
6. Tabulate the errors in Euclidean distance between the position computed by your software and the achieved position by the arm for all the samples. Compute the mean error.
7. If there is any error, what are the possible source(s) of error?

4.4 References

1. Spong, Mark W., Seth Hutchinson, and Mathukumalli Vidyasagar. Robot modeling and control. John Wiley & Sons, 2020.

2. https://www.mathworks.com/help/matlab/matlab_prog/create-functions-in-files.html

3. https://www.mathworks.com/help/matlab/ref/function.html

4. https://www.mathworks.com/help/robotics/ug/rigid-body-tree-robot-model.html

5. https://www.mathworks.com/help/robotics/ug/build-manipulator-robot-using-kinematic-dh.html

6. https://petercorke.com/toolboxes/robotics-toolbox/

7. https://petercorke.com/robotics/dynamixel-ax12a-servos/

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

– Chebyshev to Sylvester, 1873

5

CHAPTER

Inverse Kinematics

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.

5.1 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 5.1 Inverse Kinematics Solutions (40 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 end-effector is along the length of the last link, ϕ is the angle it makes with the x-axis of frame 1, i.e. $\phi = \theta_2 + \theta_3 + \theta_4$. When the gripper is parallel to the base board, then $\phi = 0°$[a].

[a] See the remarks below for further explanation

In the problems discussed in class, the manipulators have a spherical wrist allowing for a tidy decoupling of the inverse position and inverse orientation problems. Unfortunately, we have less than 6 DoF and no spherical wrist here. We will solve for the joint variables in the order that may not be immediately obvious, but is a consequence of the construction of the robot. At each step, you can either use the algebraic or geometric method to find expressions for that joint variable. For the algebraic method, you can make use of your Forward Kinematics expressions. A sketch for deriving the IK expressions is provided in Section 5.3.

Task 5.2 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.

5.2 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 mate possible strategies for choosing an IK solution, given (x, y, z, ϕ) and the current joint angles.

Task 5.3 Optimal Solution (30 points)

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

To complete this task, you may have to write helper function `checkJointLimits` to make sure that a solution is realizable. Recall that the motor limits are in the interval $[-150°, 150°]$, but you may want to constrain the angles further, similar to armLink, to avoid self-collisions. You may also need a helper function `getCurrentPose` to get the current configuration of the robot. This can make use of the MATLAB Arbotix methods. In minimizing the objective function, remember that angles wrap around, i.e. $\psi + 2n\pi = \psi$, when determining the angular difference. You can use `mod(angle+`π`,2`π`)-`π to rewrite an angle in the interval $[-150°, 150°]$ before computing the angular difference.

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

Task 5.4 IK Exploration (20 points)

(a) Select five points (x, y, z, ϕ) in the workspace of the robot and execute the optimal solution for each point, as determined by your `findOptimalSolution` function. Measure and note down the achieved point in each case.

(b) Determine the accuracy of your system and compare it to the value obtained in the previous lab. Comment on any possible statistically significant differences.

(c) Is there any (x, y, z, ϕ) in the workspace for which all possible solutions are realizable? Justify.

5.3 Sketch for deriving IK expressions

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

5.4 References

1. Spong, Mark W., Seth Hutchinson, and Mathukumalli Vidyasagar. Robot modeling and control. John Wiley & Sons, 2020.

CHAPTER 6

A complete motion control system

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

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

6.1 Our complete motion control system

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

Task 6.1 FSM (30 points)

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

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

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

Task 6.2 FSM Implementation (50 points)

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

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

6.2 Determination of the manipulator Jacobian

In the previous labs, you constructed the forward and inverse kinematic mappings of the manipulator, which relates the end-effector pose to the joint variables. The speed of the manipulator's joints or end-effector was 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 also determine Jacobian of this manipulator, which relates the joint velocities to end-effector velocities. Having determined the Jacobian, you can identify the kinematic singularities of this manipulator and observe the behavior of the manipulator near singularities.

Recall that the Jacobian of the 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 studied in the class, where each column corresponds to the respective

joint of the manipulator. However, another way to obtain J_v is by realizing that

$$^0v_4 = J_v\,\dot{q},$$

i.e. $^0v_4$ and correspondingly J_v can be obtained by differentiating the position of the origin of the end-effector frame, $^0p_4$, expressed in the base frame, with respect to time.

$$^0p_4 = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

The expressions for $^0p_4$ can be obtained from the homogeneous transformation $^0T_4$. We would still have to use the method outlined in the class to obtain J_ω.

Task 6.3 Manipulator Jacobian (20 points)

Use the DH parameters and homogeneous transformation, $^0T_4$, obtained in the previous lab to find the Jacobian for the manipulator in the lab.

- For convenience, a MATLAB function `createA(theta,d,a,alpha)` is available on canvas to easily create homogeneous transformations in symbolic form.
- 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 transformation 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 $\frac{\partial f}{\partial x}$.
- Chain rule will frequently yield simplified expressions.

6.3 References

1. Spong, Mark W., Seth Hutchinson, and Mathukumalli Vidyasagar. Robot modeling and control. John Wiley & Sons, 2020.

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

– David Marr

CHAPTER 7

Adding visual sensing to our system

The motion control system that you have built by the end of the last lab allows you to pick an object from a perfectly defined location and place it again at a perfectly defined position. This system assumes that the object to be picked is at a known location. In this lab, you'll let go of that assumption by adding visual sensing to your system. This will be used to generate the pick and (possibly) place coordinates for the motion control system, designed earlier. The installed vision system can also be used to implement a complete visual feedback-based closed loop, but we'll operate our system in the open loop configuration of Figure 3.1 for now.

Computer vision algorithms are enabling robots to tackle extremely complex environments. Our case here is comparatively easier, i.e. detecting a known object (cube) in a relatively uncluttered environment. We'll adopt the geometric approach to computer vision for determining the position of the object, as opposed to the other major approach these days, i.e. data-driven 'deep perception' approach. We'll continue to use the RGB-D camera, Intel RealSense SR-305, explored in Chapter 3.

7.1 Background

7.1.1 Where will the camera be mounted?

Our setup will involve mounting the SR-305 overhead on the provided assembly at such a height that the robot arm and the objects in its workspace will lie in the field of view of the camera. How do we determine this height? You'll determine it experimentally to obtain the required field of view in the Intel Viewer, but it can also be determined based on the camera parameters and geometric modeling, which you'll carry out in the next lab. Our perception pipeline will have to:

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

This is also illustrated in Figure 7.1.

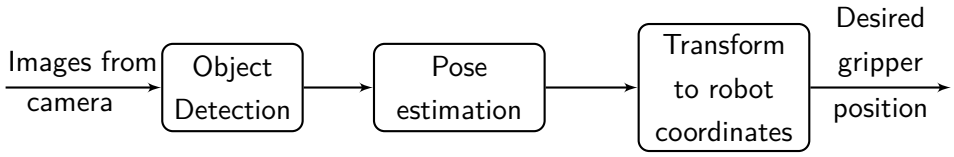


Figure 7.1: Block diagram of our vision pipeline

7.1.2 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 be focusing on depth images and point clouds.

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 7.2. Each pixel value in the RGB image is typically 3-dimensional and provides information about the luminance of visible light along the pixel direction. There are different representation formats (color spaces) for expressing this luminance information[1]. 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 7.3. Note that this distance is the normal distance from the

[1] https://www.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html

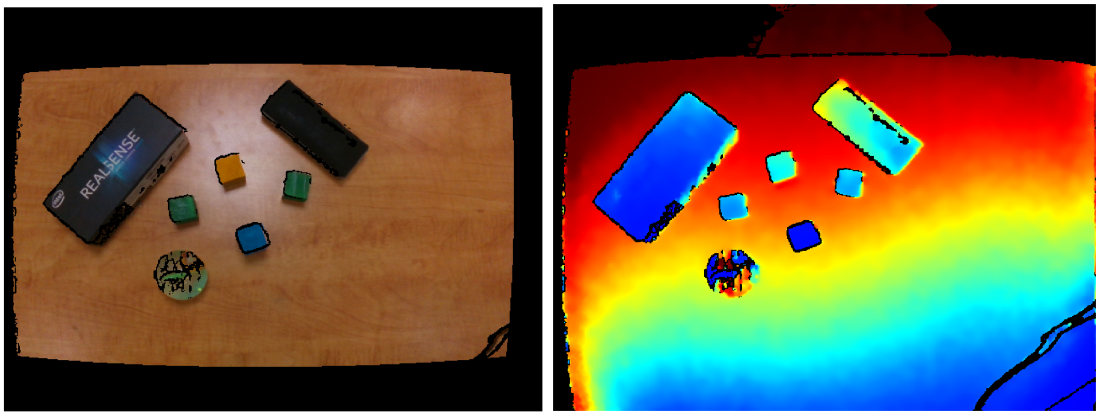


Figure 7.2: (Left) Image from RGB Camera. (Right) Depth Image

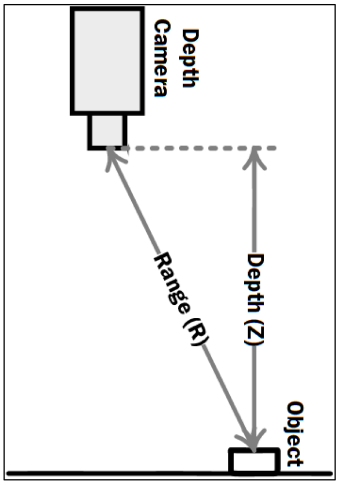


Figure 7.3: Depth value in the depth image

camera plane to that object. The different colors in the depth image in Figure 7.2 correspond to different values of depth, and are obtained by mapping the depth readings in the image to a colormap.

Noise in camera

We can notice from Figure 7.2 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, before an object detection strategy is applied, as shown in modified pipeline of Figure 7.4. Our focus in this lab will be on the two blocks of 'Object Detection' and 'Post-processing'. As you may have gathered that the two blocks naturally depend on each other, so you'll identify strategies for both steps that collectively results in robust performance.

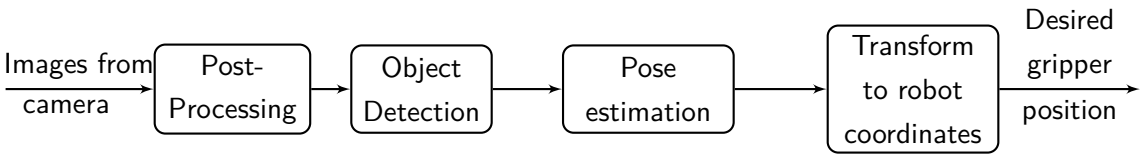


Figure 7.4: Complete flow of our vision pipeline

The MATLAB code `depth_example`, partly developed in Chapter 3, obtains these two images in MATLAB and applies desired post-processing to make them suitable for object detection. You'll have to decide the parameters of post-processing steps. All of this is done using the Intel provided SDK[2] (See [2] for more details). One of the post-processing steps is to align the two images so that their origins coincide and they're of the same size. This is why you see the black border around color image in Figure 7.2 as the native resolution of the RGB camera is higher than the IR camera, so it has to be downsampled.

Task 7.1 Explore depth images (0 points)

Set up the camera and the blocks, and use the provided MATLAB file `depth_example` to obtain a color and a depth image of your setting. Make sense of the depth readings obtained in the depth image. Study the provided code.

Point Clouds

If the information from the depth image is combined with information about the camera's internal parameters, e.g. focal length (we'll study this in class), then it is possible to transform this 2D depth image representation into a collection of 3D points from the viewpoint of the camera. This collection is called a 'point cloud'. Note that each of these points has a pose, i.e. each point has all three coordinates (x, y, z) with respect to the center of the camera lens, as seen in Figure 7.5. It is easy to convert this representation to the robot frame too, but more on that in the next lab. MATLAB has a whole set of functions available for point cloud processing that can be explored at `https://www.mathworks.com/help/vision/point-cloud-processing.html`.

Task 7.2 Explore point cloud (0 points)

The MATLAB file `pointcloud_example.m` provides code to generate a point cloud. It contains the same post-processing steps as the previous code. Run the code and explore the obtained point cloud. Study the provided code.

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

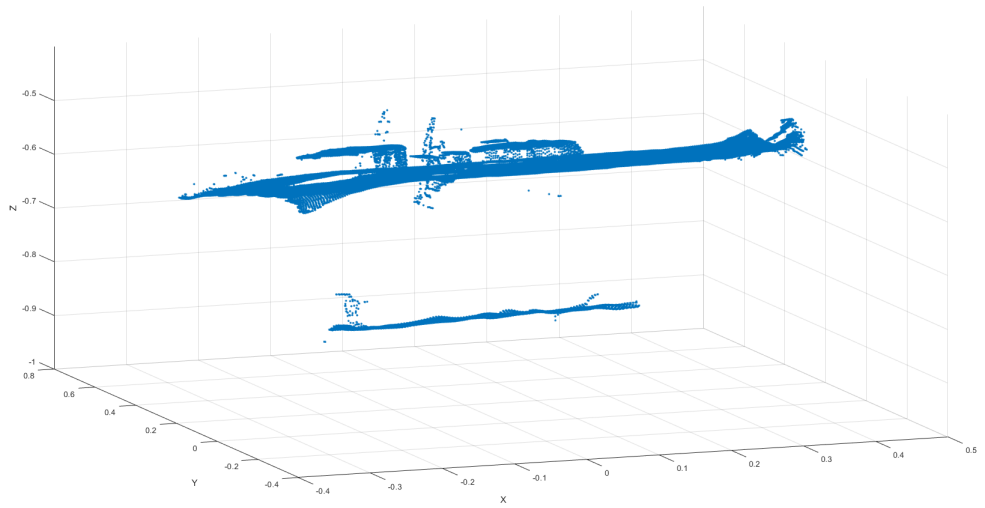


Figure 7.5: Point cloud of the same scene

7.2 Some segmentation techniques

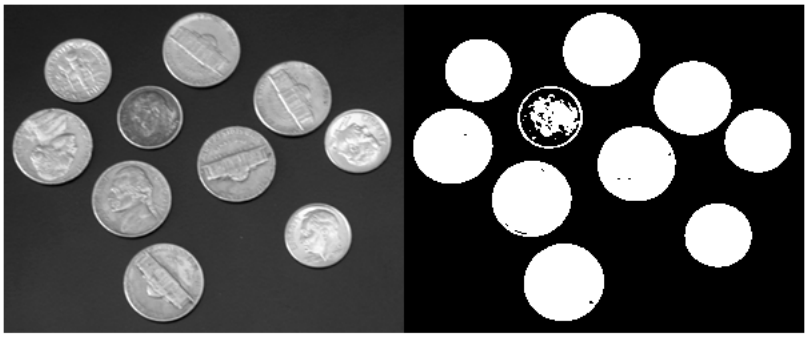
Images contain a large vast amount of data that tends to overwhelm what is significant to us in the image. *Segmentation* is the process of collecting together pixels into summary representations that emphasize some important, interesting, or distinctive properties, e.g. pixels of the same color. There 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.

7.2.1 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 7.6 provides an instance of this method in use based on the MATLAB example `https://www.mathworks.com/help/images/ref/imbinarize.html`. The 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.

7.2.2 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.`



, and also illustrated in Figure 7.7.

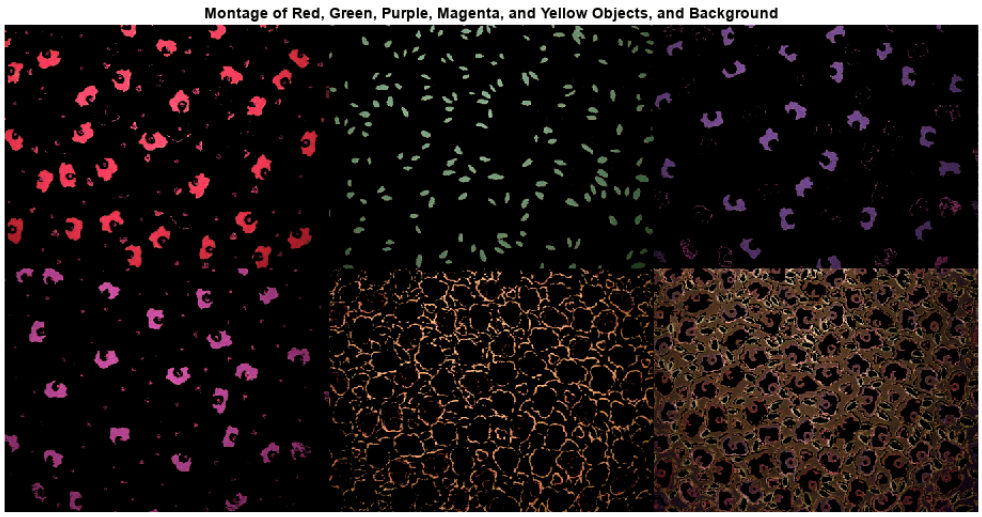


Figure 7.7: Segmentation based on color in L*a*b color space

7.2.3 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 [4]. 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.

7.2.4 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.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.

7.2.5 Fitting geometric model

A possible segmentation at a global level is to discover geometric artifacts in the images or point clouds, e.g. lines, planes, etc. Algorithms in this domain use some parametric model of the geometric figure and then classify each pixel/point in the image to be either part of this geometric figure or not. This is done based on the distance of the point/pixel from the geometric artifact. Note that the algorithms will have to determine both the values of the parameters in the 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 MATLAB example https://www.mathworks.com/help/vision/ref/pcfitplane.html shows how planes can be detected in a point cloud using RANSAC, reproduced in Figure 7.8. The example https://www.mathworks.com/help/images/hough-transform.html shows how to detect lines in an image using Hough Transform.

7.2.6 Point cloud registration

Registration is the process of finding a transformation that takes one dataset to another, e.g. given a fixed point cloud describing a cube and an incoming new 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 position and orientation of cube in the new point cloud. These algorithms are based on ICP (Iterative Closest Point) algorithm. An example for this is https://www.mathworks.com/help/vision/ref/pcregistericp.html.

Task 7.3 Vision Pipeline (100 points)

Design and develop the complete vision pipeline for determining the positions of all the cubes of different color in the workspace of your robotic system. Assume that the workspace is relative uncluttered and each color has only one known cube size associated with it.

You'll have to formulate an object detection strategy, the corresponding post-processing parameters for your strategy, and then determine the position of identified cube (see note below). You are required to submit

1. a note describing your object detection strategy and the rationale behind it;

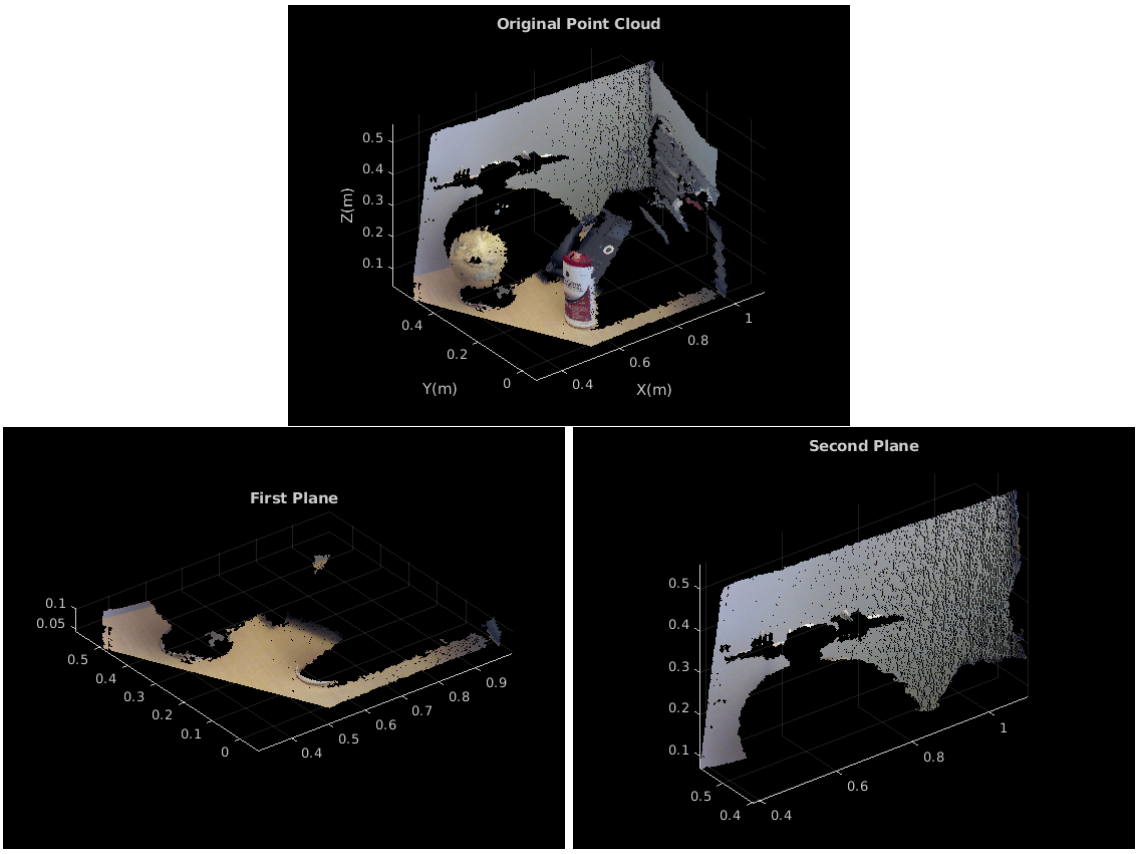


Figure 7.8: Finding planes in point cloud

2. a list of the post-processing steps and parameters, after experimenting and determining them from Intel RealSense Viewer;
3. aptly commented code for your vision pipeline;
4. images at various steps of a good test run;
5. statistics across multiple trials, including
   - number of objects of each color in the workspace to be detected
   - number of objects of each color correctly detected
   - accuracy rate for each color.

Determining the position of cube: Once the top face of a cube has been identified, i.e. we have obtained coordinates of all the points that form the top face, the position of the cube can be described by the center of top face, which is computed by finding centroid of all the points forming top face.

7.3 References

1. https://github.com/IntelRealSense/librealsense

2. https://github.com/IntelRealSense/librealsense/blob/master/doc/post-processing-filter
   md

3. https://github.com/IntelRealSense/librealsense/wiki/Projection-in-RealSense-SDK-2.
   0

4. Forsyth, D.A. and Ponce, J., 2002. Computer vision: a modern approach. prentice hall professional technical reference.

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

– David Marr

CHAPTER 8

Completing the Perception Pipeline

In this lab, you'll complete developing the perception pipeline, outlined in Figure 7.4, and integrate it with the motion control system, developed in Chapter 6. Recall that you had developed an image processing pipeline in Chapter 7 to obtain the coordinates of objects of interest, located in the camera's field of view, in the camera frame. In this lab, you'll transform those coordinates to the robot's base frame, using the perspective projective model of a camera.

8.1 Camera coordinates to World coordinates

We know that the coordinates of a point in the image plane are related to the corresponding world coordinates through the camera projection matrix, according to the perspective projective model of a camera. This camera projection matrix includes the intrinsic as well as extrinsic parameters of a camera. Since SR-305 has two cameras, we can correspondingly construct two projection matrices.

Task 8.1 Camera Intrinsic Parameters (10 points)

The MATLAB function `determineIntrinsics()`, provided on LMS, uses Intel's SDK to display the intrinsic parameters of the color camera. Modify the code to also display the intrinsic parameters of the IR camera. Elaborate each entry in light of [2].

The color camera and the IR camera have their own coordinate frames as the z-axis of a camera frame passes through the optical center of the respective camera, according to the standard perspective projective model. This is why you had to align images from the two cameras in the last lab to match the pixel coordinates, (u, v) to the corresponding depth, Z. The following lines in last lab's code served the purpose of aligning images from the two cameras:

```
align_to_color = realsense.align(realsense.stream.color);
fs = align_to_color.process(fs);
```

Task 8.2 Extrinsics used to align images (10 points)

The MATLAB function `determineExtrinsics()`, provided on LMS, will display the homogeneous transformation from the depth camera to color camera. Write it in the standard format of a homogeneous transformation.

The point cloud, in the last lab, was constructed by using these intrinsic and extrinsic parameters to establish a correspondence between coordinates in the image plane and coordinates with respect to the IR camera frame. These coordinates will have to be transformed to the base frame of our robot, so that we can reference them in our motion control system. The coordinates can be easily transformed if the homogeneous transformation between the camera frame and robot base frame is available. Note that this transformation is entirely dependent on the physical placement and orientation of our SR-305 camera, and can be written from observation. Before you affix the camera and determine its placement physically, you'll explore camera's datasheet to understand two parameters - field of view and depth start point that are relevant to the present task.

Task 8.3 Camera datasheet (10 points)

From the provided datasheet [1], determine the values of the following and provide an explanation for each quantity:

(a) Resolution of color camera and IR camera;

(b) Frame rates of both cameras;

(c) Depth field of view;

(d) Depth start point.

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 8.1. 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 8.1, 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

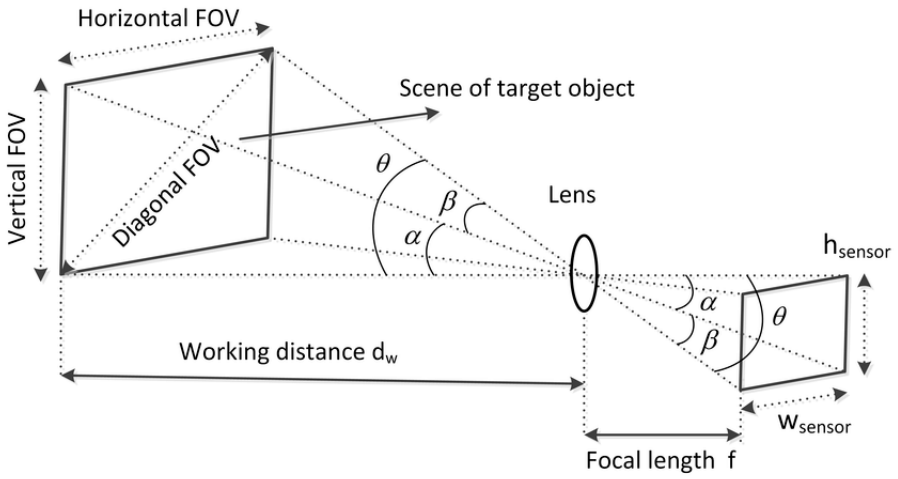


Figure 8.1: Field of view of a camera lens (Source: Ngo, T., Abdukhakimov, A., Kim, D. (2019). Long-Range Wireless Tethering Selfie Camera System Using Wireless Sensor Networks. IEEE Access.)

of the camera sensor and the focal length of the lens. It can be obtained as shown in Figure 8.2, where $\alpha/2 = \arctan(d/2f)$.

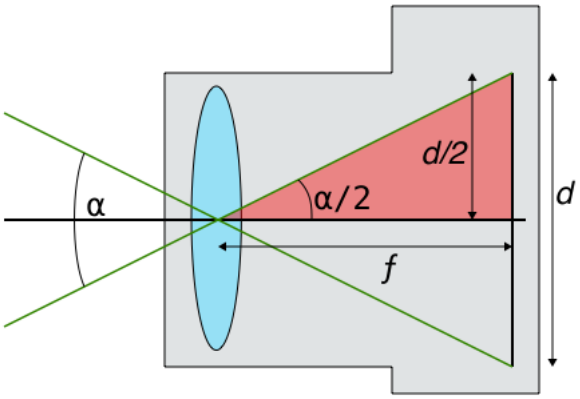


Figure 8.2: Focal length and sensor size determine fov

Task 8.4 Field of View (20 points)

Using the value for the depth field of view, compute an appropriate height (working distance) for the placement of camera so that the robot's workspace, determined earlier, is entirely in the depth FOV. Verify your computation by observing the complete workspace in the viewer. Use some way to mark the placement of your camera as we'll require that the camera be fixed during the operation of our robotic system.

Task 8.5 Real-world coordinates (20 points)

Determine the homogeneous transformation that transforms coordinates from the camera frame to robot base frame. Physically verify your computation and include your test results.

You can do this by finding the coordinates of an identifiable point physically and comparing them to the coordinates determined from the camera transformed to real-world coordinates. Note that you can directly determine the depth of pixel coordinates (u, v) by using the function `depth.get_distance(u,v)`, if `depth` is the extracted depth frame.

8.2 Vision-based pick and placer

Task 8.6 Picking and placing a cube (30 points)

Randomly place a single cube in your robot's workspace in an orientation such that it is possible for the arm to pick it up. In this task, your robotic system should correctly identify the location of the cube, pick it, and place it at a pre-specified free region in the workspace, using the motion control FSM from the previous lab. You'll have to connection your perception pipeline to motion control FSM. You're to submit a video of your successful test run and a note on the performance of your system with supporting data.

8.3 References

1. Intel RealSense Depth Camera SR300 Series Product Family Datasheet
2. https://github.com/IntelRealSense/librealsense/wiki/Projection-in-RealSense-SDK-2.0

CHAPTER 9

The Robot Games

This is our final session with the arm in the lab. The objective of this session is to see the fruits of your semester long efforts by making the robot execute complex tasks. Before we do that, we'll determine the singularities of this arm so that they can be avoided during the execution of these tasks.

9.1 Singularity Analysis

Recall that 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. We only have to look at Jacobian found in Chapter 6 to come to this realization. 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 9.1 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 at the same location as the origin of frame 3, i.e. $o_4 = o_3$. This will effectively set $a_4 = 0$ in our DH parameters and the Jacobian simplifies significantly.
- 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(t),'a_4',0)`.
- Looking at the Jacobian, you would realize that you can find the rank by finding the determinant of the top left 3×3 block of Jacobian matrix. The MATLAB function for determinant is `det`.
- Remember to make judicious use of `simplify` and `expand`.

(a) Show that the Jacobian loses rank when

$$a_2 a_3 \sin\theta_3 \left[a_2 \cos\theta_2 + a_3 \cos(\theta_2 + \theta_3)\right] = 0.$$

This is the same condition as the one obtained for RRR arm in your book.

(b) The previous expression corresponds to three possible configurations:

(i) Position A: the arm is fully stretched.

(ii) Position B: the end-effector is right above the base, without the arm being stretched.

(iii) Position C: the end-effector is above the base and the arm is stretched.

At each position, in which direction is the arm unable to move?

9.2 Complex Pick and Place Tasks

Task 9.2 Bin it! (50 points)

Randomly spread 10 cubes on one side of your robot's workspace. Out of these 10 cubes, five should be a mix of red and yellow, and five a mix of blue and green cubes. Designate spots in your robot's workspace for a red bin and a yellow bin. Assume that there are no other items in the workspace. Make your robot system place all the red cubes in the red bin and the yellow cubes in the yellow bin. You'll be scored by the RA after they have randomly placed cubes in the workspace. You'll get 10 points for every correctly placed cube, -10 points for each cube placed in the wrong bin, and -5 points each time a cube falls enroute. You're also to submit a video of a successful execution and MATLAB code.

Task 9.3 Hidden in the stacks (30 points)

Again, randomly spread 10 cubes in the same color distribution - five from the set of red and yellow, and five from blue and green. But now it is possible that the cubes are stacked to any level. Designate spots in your robot's workspace for a red bin and a yellow bin. Assume that there are no other items in the workspace. Make your robot system place all the red cubes in the red bin and the yellow cubes in the yellow bin. You'll be scored by the RA after they have randomly placed cubes in the workspace. You'll get 6 points for every correctly placed cube, -6 points for each cube placed in the wrong bin, and -3 points each time a cube falls enroute. You're also to submit a video of a successful execution and MATLAB code.