EE366L/CE366L: Introduction to Robotics Lab

Dr. Basit Memon

April 20, 2022

Contents

Lab Instructions

1. Make sure to sanitize your hands as you enter the lab.
2. The credentials for signing in to the lab computers are:

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

3. Since we're working in groups, rotate among yourselves so that each member of the group has a chance to get hands-on experience.
4. Tasks marked with an [*] don't require access to the hardware and can be completed post-lab.
5. 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.
6. The weekly lab findings report will be 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

1

Getting familiar with MATLAB

"If you know your enemy and know yourself, you need not fear the result of a hundred battles."

– Sun Tzu, The Art of War

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`). We'll be utilizing this exact arm for the first part of this lab course. 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.**

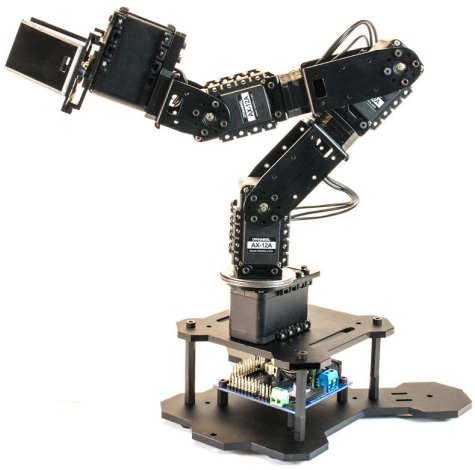

Figure 2.1: Phantom X Pincher Arm

Task 2.1: (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.

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

ii. 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.

iii. 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.

iv. How many degrees of freedom does this arm possess?

2.2 The Arbotix-M Controller

The Arbotix-M controller, depicted in Figure 2.3, lies at the heart of this arm. This controller receives higher level instructions from an external processing unit and generates instructions in specific format required for the servomotors 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.

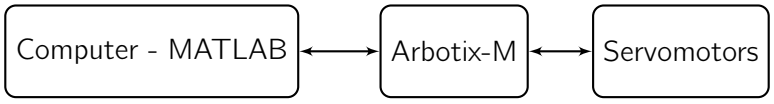

Figure 2.2: Data flow between MATLAB and servomotors

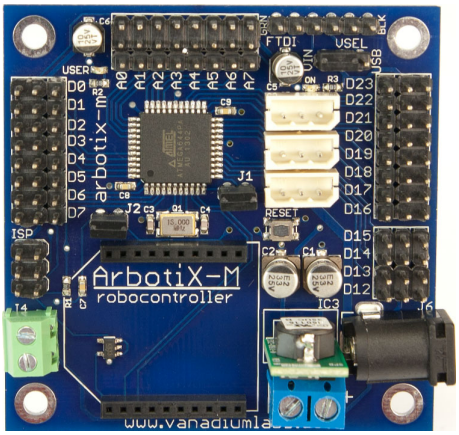

Figure 2.3: The Arbotix-M Controller

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 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.

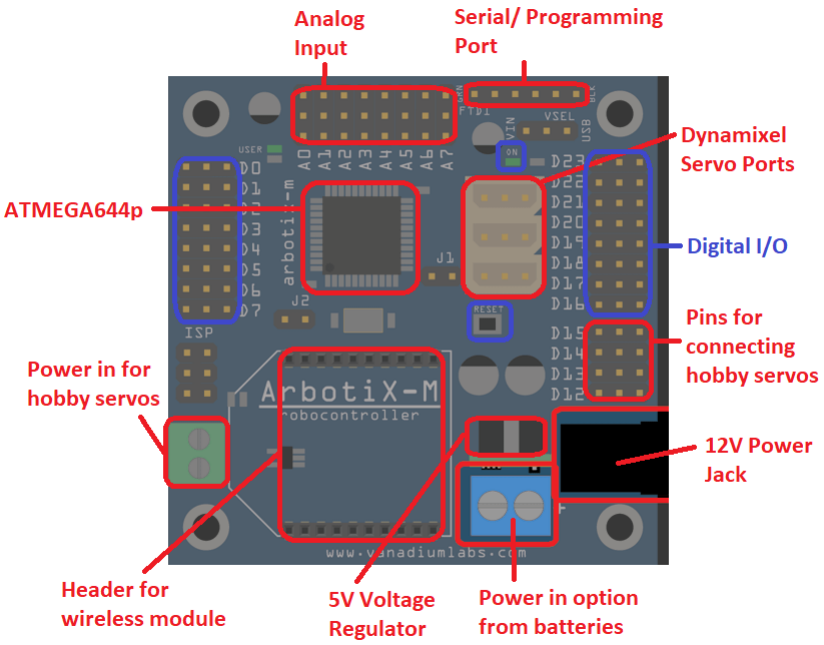

Figure 2.4: Getting to know Arbotix-M

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.

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.

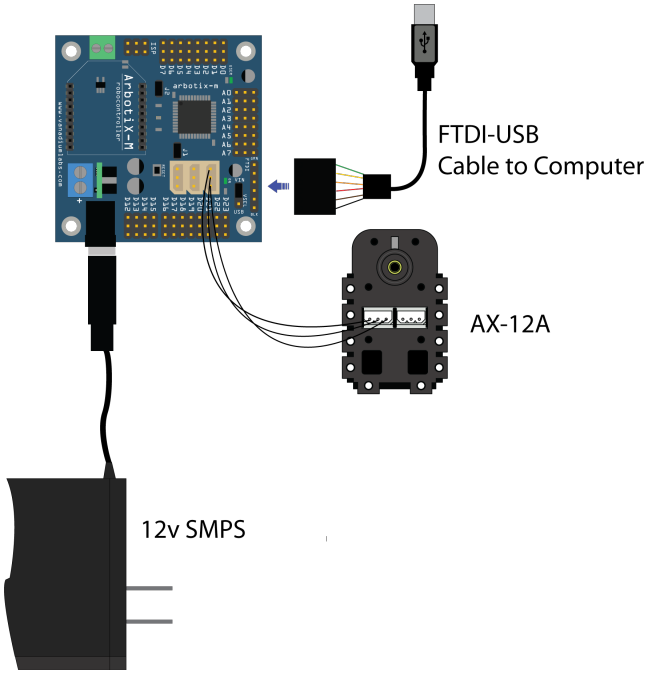

Figure 2.5: Setting up the Arbotix-M with power, servos, and programming

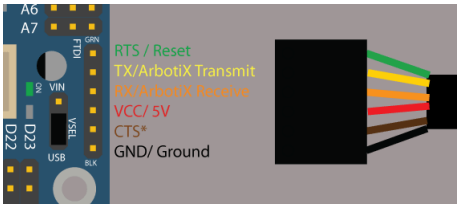

Figure 2.6: FTDI-USB Cable Connection

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.

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

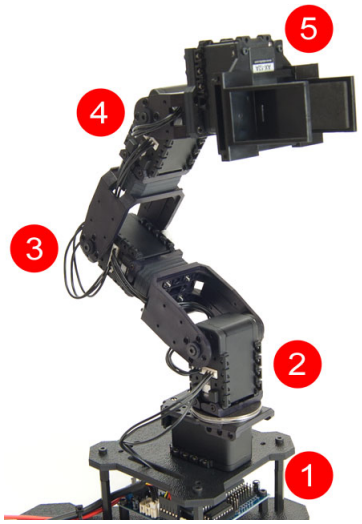

Figure 2.7: IDs of the servos in arm

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)
   ```

   (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

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. Feel free to play around with the controls as you read through this section. **However, 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.**

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

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

3. **Backhoe:** In this mode, you can directly control the position of the base, shoulder, and elbow joints.

In addition to this,

- The `Straight` and `90 Degrees` wrist option place the wrist in the horizontal or vertical configuration.

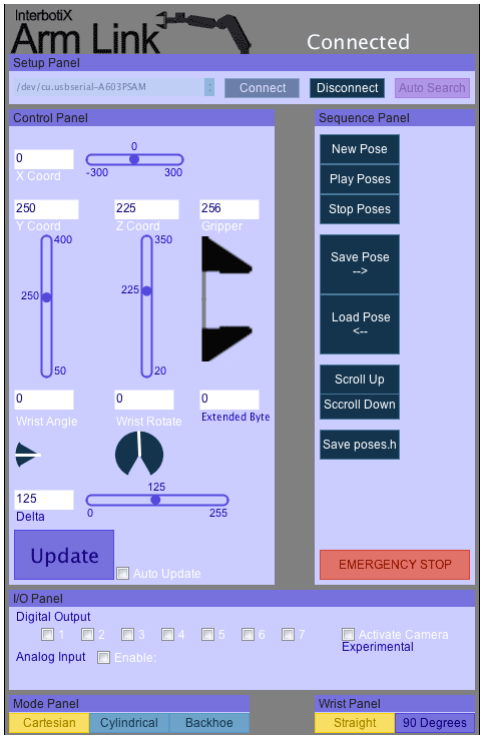

Figure 2.8: Arm Link Application Panel

- 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: (25 points)

The arm manufacturer's website states that the vertical reach of this arm is 35 cm and its horizontal reach is 31 cm. Move the robot to a configuration that allows the end effector to reach its maximum height.

(i) Note down the configuration in the form of joint-link diagram. In this diagram, links can be represented by line segments and revolute joints by circles.

(ii) Using a ruler or any other measuring instrument, find the vertical reach, and comment on how distance was measured and its comparison to the provided specification. **Note that the endpoint of line segment for measuring distances is always chosen to be at the mid-point across all three axes in the space between the fingers of grasper.**

(iii) Move the robot to a configuration with maximal horizontal reach now, document the configuration, and measure the reach.

(iv) 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. What will be the dexterous workspace of this robot manipulator, if any? Think of an example of an unachievable wrist configuration.

(v) [*][a] The coordinates in the Cartesian or Cylindrical mode describe task space locations. Task space is 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 Resolution

The next task is about an important parameter of any robot's performance, called resolution.

Resolution

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

Task 2.3: (25 points)

You may be wondering about the physical interpretation of the numbers being inputted in the Arm Link panels. These numbers appear to be a dimensionless representation that map the physically possible range for any variable, e.g. X, onto a subset of integers between 0 and 1023[a]. 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.

We'll determine the lower and upper limits and resolution of the arm in the X, Y, and Z directions. For each direction,

1. move the robot to the reachable point closest to the base in that direction;
    - The base is the black structure on which the first motor is mounted.
    - You'll have to keep the coordinates of the two directions constant in the panel, other than the one you're interested in.
    - For the direction of interest, try to set its coordinate to its lowest possible value in the panel. Be careful and move it slowly, keeping an eye on possible collisions. In case of an impending collision, adjust the other two coordinates.
    - You may not be able to achieve the limit of an axis from every configuration in the workspace.
2. note down the coordinate corresponding to this location from the panel; you can also find the arm limits at `https://learn.trossenrobotics.com/arbotix/arbotix-communication-controllers/31-arm-control#limits`;
3. measure the physical distance from the origin to the end-effector point, identified in the previous task; 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.
4. move the robot to the reachable point farthest from the base in that direction and repeat steps 2 and 3;
5. the resolution of the arm in this direction is computed as (the total physical distance traveled) divided by the (difference in coordinates plus one). This is the minimum distance that our manipulator can move in this direction when we increment the coordinate by one in the software panel.

You're to provide limits of the arm both in Arm Link coordinates and physical units, and the working for the resolution, for each of the three directions - X, Y, and Z. Also, provide any comments that you may have on this process - encountered problems, questioned assumptions, significance, etc.

[a]The number range is 0-1023, because the arm servos use a 10 bit location for motor angle.

2.5 Accuracy

Accuracy

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

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

Task 2.4: (30 points)

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

1. Randomly select a few points (x, y, z) in the workspace of the robot manipulator. Note that these will be in physical units, e.g., mm, cm, etc.
2. Based on your response to Task 3, find the software coordinates corresponding to each point and direct the robot to move to each of these points.
3. Physically measure and find the actual location of the end-effector, and note it down.
4. Compute the absolute error between the specified location and actual achieved location, and find the mean absolute error over all samples, which represents accuracy of the robot. Don't forget the units.
5. Does the accuracy of this robot vary with distance from the base? Divide the workspace into concentric circles, radiating outwards from the base. Choose a few points in each region and compute the accuracy separately for each region.

You are to provide your data in tabular form for global accuracy calculation in 4. For 5, provide your methodology for dividing the manipulator workspace, tabulated data for each region, an image of the regions and respective accuracy circles, and comments on the obtained accuracy values.

2.6 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-link-software.html
4. Getting started with Arbotix-M https://learn.trossenrobotics.com/arbotix/7-arbotix-quick-start-guide#step3

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

– Richard P. Feynman

3

Manipulate with the arm

This lab is a continuation of the previous lab's activities on getting familiar with the capabilities and limitations of the Phantom X Pincher arm. You'll be introduced to one more performance specification, delve deeper into understanding the capabilities of the actuators that drive this arm, get a sense of the manipulation abilities of the arm and attached gripper, and explain the arm's operational abilities with simple physics.

3.1 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 3.1.

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 3.2. 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 format for receiving instructions, and the structure to be followed in every 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.

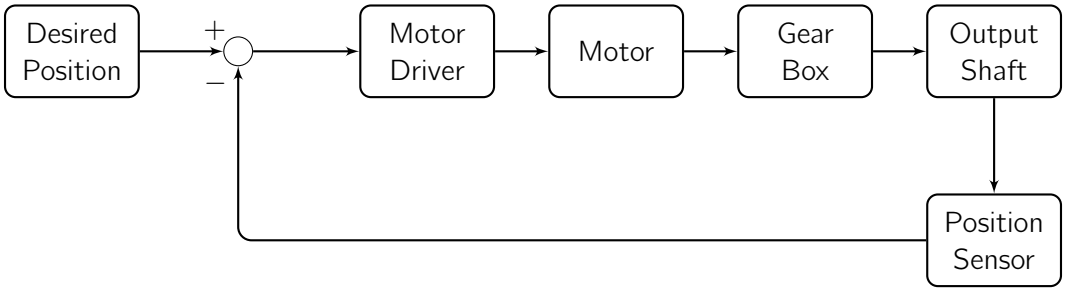

Figure 3.1: Block diagram of a servomotor

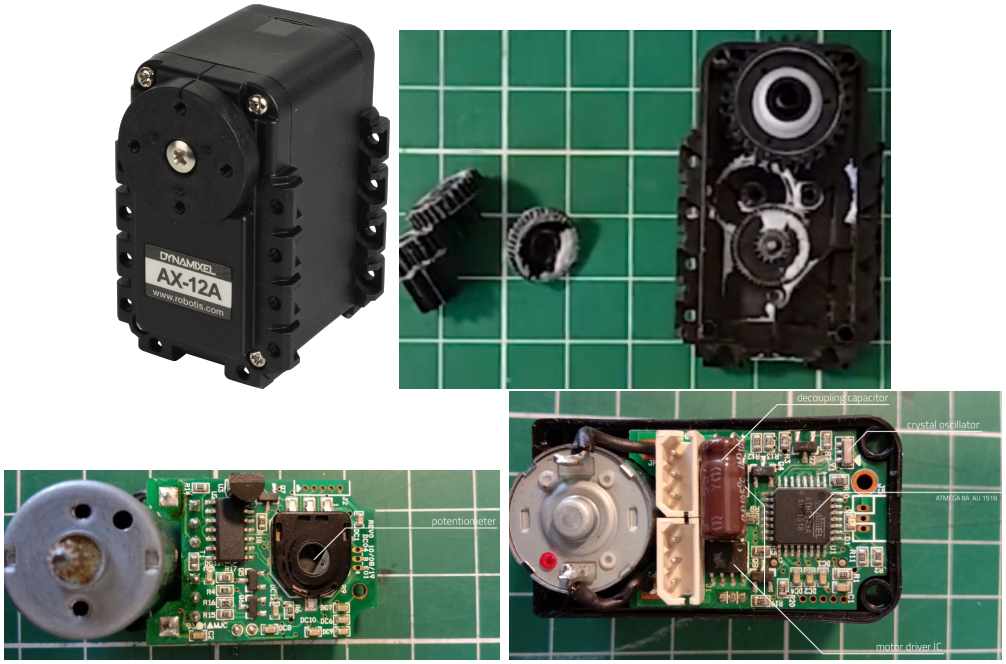

Figure 3.2: Top Left: AX-12A Motor, Top Right: Gear Box, Bottom Left: Potentiometer, Bottom Right: Microcontroller and Motor Driver

Task 3.1: (15 points)

This task is about familiarizing ourselves with the Dynamixel reference manual (`https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/`), some relevant specifications included in it, and reflect on our understanding of these arms.

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

(b) [*] Comment on the relationship between this angular resolution of motor and the Cartesian resolution determined in the last session.

(c) [*] The only sensors on this arm are inside servomotors measuring the angular position of each joint. Recall that ArmLink software allowed you to specify coordinates in the Cartesian task space. Think and comment on what must be happening at the back-end of this software.

[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.

3.2 Repeatability

Repeatability

This specification represents the ability of a manipulator to return repeatedly to the same location.

This measure is associated with the motion taught to the robot. 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. In this case, all the factors affecting the accuracy of a robot are inherently taken into account and the only factor affecting the repeatability is its resolution.

Repeatability is also specified in the form of radius of a sphere enclosing the set of locations to which the manipulator returns, when sent from the same origin by the same program with the same load and setup conditions.

Task 3.2: (25 points)

To determine the repeatability of this arm, you'll teach a simple motion to the robot, execute it multiple times, and observe the effect. 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.

1. 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.
2. 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
```

3. Center the servos. This is option 8 in the menu. Enter this number in the serial monitor. Using this as our initial configuration will ensure the same setup each time.
4. Relax the servos, which will allow us to move the arm by hand. This is option 0 in the menu.
5. 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.
6. Center the servos again, resetting to the initial pose, and then play the sequence of poses (option 4)

(contd.)

7. Physically verify that the end-effector has reached the desired location. Measure any straight-line difference, using callipers, and note it down.
8. Repeat the previous two steps a number of times to generate enough samples.
9. For each sample, compute the absolute error between the achieved and intended location. Obtain the mean absolute error over all samples. This mean will be our measure of repeatability.

For your submission, briefly describe the taught motion, provide a video of the motion, tabulated raw data, repeatability calculations, and your comments on the results.

3.3 Grasping Poses

This is a fun section where you'll experiment with the grasping capabilities of this manipulator.

Task 3.3: (20 points)

Set up `ArmLink`, which you'll be utilizing for this task.

(a) Grab one each of the provided cube, triangular, and box-shaped objects. In this task, you'll place each of these objects at a fixed location in your workspace, specify the coordinates of that location in ArmLink to move the arm to that location, and grasp that object. **You'll not use the sliders in this task.**

- Describe the process of converting your measured coordinates to ArmLink coordinates, based on your findings from previous tasks.
- Find a mapping between the physical gap between two fingers and ArmLink coordinates of gripper.
- Employ your mapping to determine the gripper coordinates for each object.
- Determine the best wrist orientation to grab each object.
- Report and comment on your findings.
- Note that an object can be placed on the board in any orientation. For each object, is there an orientation which precludes you from grabbing it with this arm?

(b) Try to grab other objects with this arm.

Task 3.4: (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.

3.4 How much can the arm lift?

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.

3.4.1 Torque requirements for lifting

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

Let's start by considering a one-link arm, shown in Figure 3.3, which can rotate about the point $O2$. Assume that the link is of length L, has mass m_1, and the arm is carrying a load of mass M_L. Then the torque exerted by the load at any position of the arm is given by

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

where the first term is due to the weight of the load and the second term because of the weight of the link itself. It is assumed that the center of mass of the link is at its mid-point.

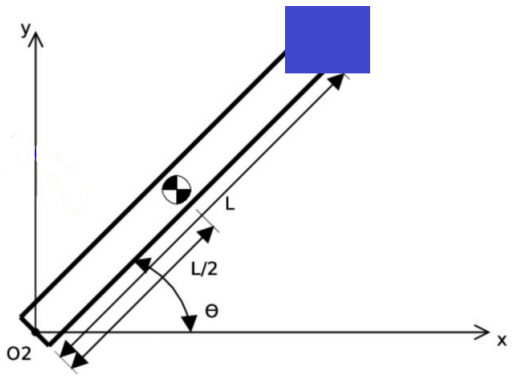

Figure 3.3: One link arm

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 3.4. The lengths and masses

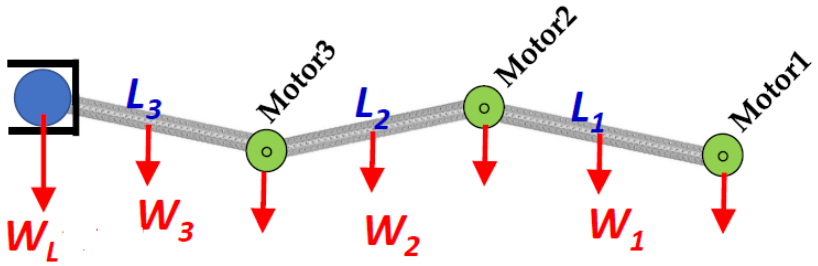

Figure 3.4: Multi-link robot arm

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

on each of the motors can be computed as:

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

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

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

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

3.4.2 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 3.5. Let F be the force being exerted by

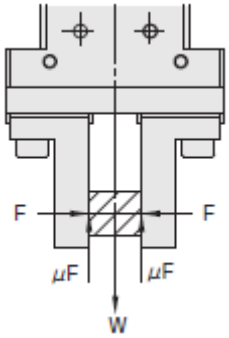

Figure 3.5: Force required to grip an object

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

be equal to the weight of the object, i.e.

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

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

Task 3.5: (10 points)

(a) 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 3.6, and physical principles outlined in Section 3.4.1, verify the wrist lift strength specified by the manufacturer. The wrist lift strength is the load that the wrist joint can support.

(b) **(Bonus)** Verify the arm strength specifications provided by the manufacturer.

Figure 3.6: Masses of different arm components

Task 3.6: (10 points)

Based on your experiences with this arm in the current and previous session, write a brief note outlining your takeaways and any questions that may have risen about this arm and robotics, in general.

3.5 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)

"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 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.

Task 4.1: DH Frame Assignment (10 points)

Choose a suitable image of the Phantom X Pincher or draw a graphical representation of it, using the convention from the book, and assign DH frames to it. Make sure to clearly indicate the z and x axes of each frame; drawing the y axis is optional. Place the origin of the end-effector frame at any point on the body of the gripper, for convenience of measurements in upcoming tasks.

Task 4.2: DH Parameters (10 points)

Determine the DH parameters based on your frame assignments. You can determine the values of the parameters either by measuring them physically or applying the available measure function to the manufacturer provided 3D model (`https://www.trossenrobotics.com/p/PhantomX-Pincher-Robot-Arm.aspx`.

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

$$A_i = \begin{bmatrix} c\theta_i & -s\theta_i\, c\alpha_i & s\theta_i\, s\alpha_i & a_i\, c\theta_i \\ s\theta_i & c\theta_i\, c\alpha_i & -c\theta_i\, s\alpha_i & a_i\, s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Task 4.3: Homogeneous Transformations (15 points)

In this task, you'll use MATLAB's symbolic math toolbox to determine the intermediate homogeneous transformations ${}^0T_1$, ${}^1T_2$, ${}^2T_3$, ${}^3T_4$, and ${}^0T_4$.

(a) Write a MATLAB script to document 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.

(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.

4.2 Positioning the arm

Now that you have the expressions for the position of the end-effector in terms of the joint angles, let's test them out.

4.2.1 Building the functions on MATLAB

Task 4.4: Forward Kinematic Mapping (10 points)

Write 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. 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 model for Phantom X Pincher.

Task 4.5: (10 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 should display the position and orientation of end-effector with respect to the base frame.

Select 4-5 random configurations for the manipulator and determine the end-effector position and orientation from the provided script 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.

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

4.2.2 Setting up communication between MATLAB and arm

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

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

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

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

4.2.3 Library of Arbotix Functions

Almost all capabilities provided by the manufacturer in Dynamixel AX-12A motors has been captured in the MATLAB library. Following is a collection of method 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 take 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.

4.2.4 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 4.6: (20 points)

Write 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 `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, depending on your choice in Task 4.
- 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.
- You have the 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.
- The definition of DH joint angle may not be the same as the one adopted by the motor software for its joint angle. In order to provide the correct input to the motors, you need to find out any possible shift between 0° of DH joint angles and 0° of the motor angles of the corresponding joints, whether the positive rotation directions in DH frames and corresponding motors are aligned, and compensate for any differences in your function's code. This can be done by comparing the home configuration of the robot according to DH frame assignment and the motors' inherent calibration.
  - Draw the zero configuration of the robot and provide it with your submission. This is the configuration in which all the joint variables, according to your DH frame assignment, are zero.
  - Compare this zero configuration to the home configuration of the arm, i.e. when all motors receive instruction to set their position to 0°. Complete and submit Table 4.1, based on your observations.

Motor ID	DH Joint Angle (θ_i)	Servo Angle ψ_i
1	0°	
2	0°	
3	0°	
4	0°	

Table 4.1: Linear mapping between servo angles and DH angles

Task 4.7: (10 points)

Select 5 random configurations for the manipulator and determine the end-effector position and orientation from your own `findPincher` function and by physically measuring the position. Tabulate the errors in Euclidean distance between the position computed by your software and the achieved position by the arm for all samples. Compute the mean error. If there is any error, what are the possible source(s) of error?

4.3 Identifying reachable workspace

With the help of our forward kinematic mapping, we can run a Monte Carlo experiment to identify the reachable workspace of our manipulator. By randomly sampling our joint space and then computing the corresponding end-effector position using forward kinematics, we can get a reasonable representation of the reachable workspace if the random sampling is dense.

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 $\theta_i^{\min}$ and upper bound $\theta_i^{\max}$ using the expression:

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

Task 4.8: Identifying reachable workspace (15 points)

Use the outlined idea for generating random joint angles and your forward kinematics function to plot the reachable workspace of our Phantom X Pincher. You can use the MATLAB function `plot3` to plot your result. What is its projection in the x-y plane of the base frame? What is the maximum horizontal reach according to your identified workspace?

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-parameters.html
6. https://petercorke.com/toolboxes/robotics-toolbox/
7. https://petercorke.com/robotics/dynamixel-ax12a-servos/

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

– Richard P. Feynman

5 Velocity Kinematics and Singularities

In the previous lab, you constructed the forward kinematic mapping 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 determine Jacobian of this manipulator, which relates the joint velocities to end-effector velocities. Having determined the Jacobian, you'll identify the kinematic singularities of this manipulator and observe the behavior of the manipulator near singularities.

5.1 Determination of the manipulator Jacobian

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 5.1: Manipulator Jacobian (10 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 transform you'll obtain will be a 4×4 matrix function of t. To extract a particular entry of this matrix, you'll have to first evaluate it at a value of t and save it in an intermediate variable. For example, if `B` is a matrix symbolic function and you want to find matrix entry $(1, 2)$, then use:

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

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

5.2 Singularity Analysis

In this part, you'll identify the singularities of this manipulator. 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.

Task 5.2: Rank of Jacobian (10 points)

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

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

Determining the conditions at which the Jacobian drops rank is not an easy task. We only have to look at our recently found Jacobian 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 5.3: Singular Configurations (15 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) List and interpret the singular configurations.

Task 5.4: Exploring singularities (15 points)

The previous expression corresponds to three possible configurations:

1. Position A: the arm is fully stretched.
2. Position B: the end-effector is right above the base, without the arm being stretched.
3. Position C: the end-effector is above the base and the arm is stretched.

Move the manipulator to each of these positions. You can keep $\theta_4 = 0°$. Change each of the joint angle by small amount. At each position, in which direction is the arm unable to move?

5.3 References

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

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

– Richard P. Feynman

6 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.

6.1 Finding all IK solutions

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

One way is to specify the direction of one of the columns in the rotation matrix, e.g. if we want the gripper jaws pointing straight down and the end-frame is assigned such that $\hat{x}$ is

pointed along the length of the last link, then we need to set the first column of rotation matrix as $(0, 0, -1)^T$. Another way is 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.

6.1.1 Solution Derivation

Task 6.1: Inverse Kinematics Solutions (10 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. 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. The expressions provided in these steps are with respect to our frame assignments and your expressions may be different.

1. Write down the expressions for x, y, and z of the end-effector from your forward kinematics mapping.
2. Solve for θ_1, which is dependent on desired position only.
$$\theta_1 = \arctan 2(y, x)$$
3. Are there any other solutions for θ_1?
4. Find the coordinates of the wrist center (r', s'), i.e. the origin of frame 3.
$$(\bar{r}, \bar{s}) = (r - a_4 \cos\phi, s - a_4 \sin\phi),$$
where $r = \sqrt{x^2 + y^2}$ and $s = z - d_1$.
5. Solve for θ_3 and θ_2, which are dependent on the wrist center.
$$\cos\theta_3 = \frac{\bar{r}^2 + \bar{s}^2 - a_2^2 - a_3^2}{2a_2a_3}$$
$$\theta_2 = \arctan 2(\bar{s}, \bar{r}) - \arctan 2(a_3 \sin\theta_3, a_2 + a_3 \cos\theta_3)$$

6. Are there any other solutions for (θ_2, θ_3)?
7. Solve for θ_4, which is dependent on the desired orientation and joint angles θ_2 and θ_3

6.2 Implementation of IK solutions

Task 6.2: Inverse Kinematics MATLAB function (10 points)

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

Task 6.3: IK Exploration (10 points)

Is there any (x, y, z, ϕ) for which all possible solutions are realizable? Attach images of the manipulator achieving all solutions.

Task 6.4: Determining accuracy (10 points)

Determine the accuracy of the manipulator with your code and compare it with the accuracy determined earlier using ArmLink. For this, you'll again have to identify random points in the workspace, find corresponding joint angles, command the arm to realize one of possible solutions, and compute Euclidean error between achieved and desired positions. The mean error over all these points is a good metric for accuracy.

6.3 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 6.5: Optimal Solution (10 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 functions `checkJointLimits` and `getCurrentPose` to make sure that a solution is realizable and get the current configuration of the robot respectively. The latter can make use of the MATLAB Arbotix methods. In minimizing the objective function, remember that angles wrap around, i.e. $\psi + 2n\pi = \psi$, when determining the angular difference. You can use `mod(angle+π,2π)-π` to rewrite an angle in the interval $[-150°, 150°]$ before computing the angular difference.

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

6.4 References

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

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

– Richard P. Feynman

7

A complete motion control system

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

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

7.1 Smooth Motion

You would have noticed that the MATLAB function `setpos` doesn't allow you to control the speed of motion and results in jerky motion. In this section, you'll build a function to execute smooth motion between given joint angles q_0 and q_1. This is done by constructing a smooth trajectory between these joint angles and then making the motor controllers follow those trajectories.

Task 7.1: Trajectory Generation (10 points)

Write MATLAB function `coefficients = findCoefficients(initPos,finalPos,Tf)`, which accepts a scalar initial angle, θ_0, a scalar final angle, θ_f, and a time duration, Tf, and returns coefficients corresponding to either a cubic trajectory or a quintic trajectory between θ_0 and θ_f in time Tf. The velocities and accelerations (if needed) at the end points can be assumed to be zero. The units of the arguments can be chosen to be consistent with the design of your system.

While you have created a trajectory, you don't have control over the entire system. The motor controller to achieve a specified joint angle is already implemented and we don't know the exact values of the velocities and accelerations. So, you'll have to determine a suitable value of Tf experimentally.

Task 7.2: (20 points)

Write MATLAB function which accepts two joint vectors q_i and q_f, assumes the manipulator is currently in q_i configuration, and moves it smoothly to q_f. You'll have to experimentally determine a way to decide Tf based on the maximum entry of $|q_f - q_i|$. During this period Tf, you'll be repeatedly setting the position of all the joints based on the current time and determined trajectories. You'll have to experimentally determine the time, dt, between two successive commands to the motors. If dt is too small, you are frequently changing the goal position for the motor and consequently requiring it to constantly change its velocity and acceleration to compensate for the positional error. If dt is too large, the motor will keep on achieving its goal and come to rest every time before the next command arrives.

You may have noticed that the stated joint limits, $[-150°, 150°]$, don't account for collisions of links with other links.

Task 7.3: Correcting Joint Limits (5 points)

Carefully move each joint motor to experimentally determine the joint limits for the respective joint that prevent corresponding link from colliding with the neighboring link. Note the joint limits for each joint and update your previous limit checking codes with these values.

7.2 Our complete motion control system

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

understanding of the topic.

Task 7.4: FSM (20 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.
- System should verify the final placement location.
- Smooth motion and accurate placement is desirable.

Task 7.5: FSM Implementation (45 points)

Implement the system described by the previous FSM in MATLAB for Phantom X Pincher and the cube object. Submit a video of your best execution and identify and comment on points of improvement. Till this point, we haven't tried to avoid singularities. What would be your strategy for avoiding singularities?

7.3 References

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

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

– Richard P. Feynman

8

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. It is now time to add visual sensing to this system. This will be used to generate the pick and place coordinates for the motion control system, designed earlier. The installed vision system can also be used to add visual feedback to our pick and place operations.

8.1 Camera

You'll be using Intel RealSense SR305 camera, shown in Figure 8.1, in this project. 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 a color image. A number of ways are being used in practice today to calculate depth, e.g. stereo images, time 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 8.2. An emitter projects patterned light, typically a pattern of vertical bars, on to a scene, as illustrated in Figure 8.3. The bars illuminate certain pixels in the scene. Using the image plane coordinates of illuminated pixels, depth is determined by an intersection between the perspective projective model of camera and that of the emitter. 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

Figure 8.1: Intel RealSense SR305 Camera

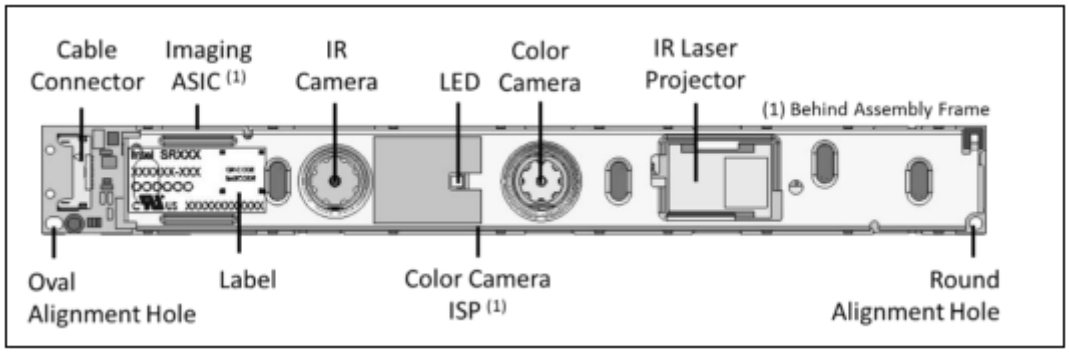

Figure 8.2: Components of an SR-305 Camera

SR-305 camera is provided in [1].

8.2 Working with SR-305 Camera

Task 8.1: 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 [4].

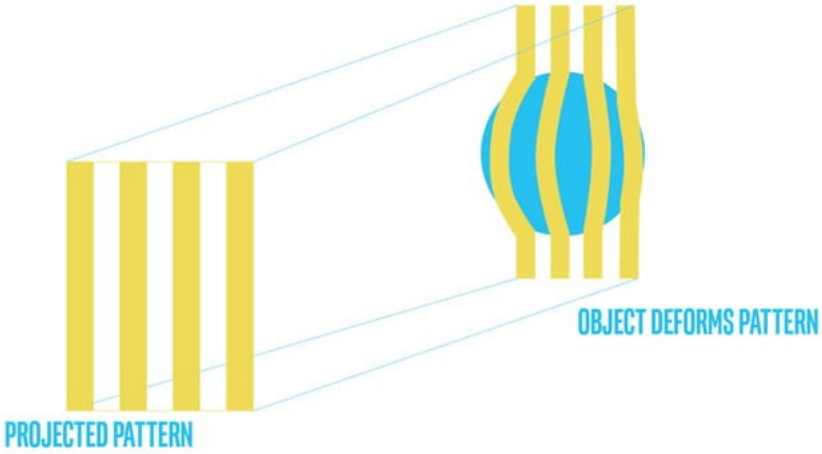

Figure 8.3: Principle of coded light

Task 8.2: Camera datasheet (20 points)

From the provided datasheet [3], determine the following:

(a) Resolution of color camera and IR camera;

(b) Frame rates of both cameras;

(c) Depth field of view;

(d) Depth start point and also explain it after experimenting with it.

Task 8.3: Field of View (20 points)

Using the value for the depth field of view, determine an appropriate height for the placement of camera so that the robot's workspace, determined earlier, is entirely in the depth FOV. This is important as the camera will be unable to determine depths of points outside FOV. Figure 8.4, while illustrating the diagonal FOV may still help you in the geometric formulation required for this task. 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.

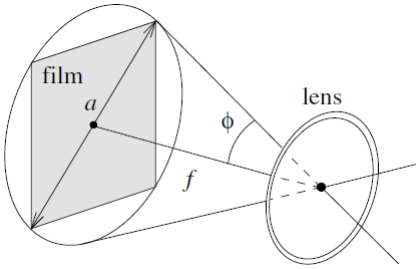

Figure 8.4: Diagonal FOV

8.3 Getting the images in MATLAB

Intel provides an SDK ([2]) 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 8.4: Extract color and depth images (20 points)

Your main task for today's lab is to extract frames of both depth and color sensors. The MATLAB function `depth_example()` in the directed 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 extracts a color frame too 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.

You may have noticed in the previous tasks that the depth information provided by the camera is reasonably accurate. In class we discussed the process for calibrating a camera to determine its projection matrix, but it is not required for this camera as it comes pre-calibrated from the manufacturer. We can still find out its intrinsic parameters using the provided functions.

Task 8.5: Camera Intrinsic Parameters (10 points)

The MATLAB function `determineIntrinsics()`, provided on LMS, will 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 acquired knowledge from class and [5].

As outlined in [5], the two cameras each have their own coordinate axes and the pixel coordinate in the color and depth images will be with respect to their own coordinate axes. If we intend to determine the real world coordinates of a point using the data from these two images, we'll have to be aware of the transformation between these two coordinate axes and align the images. Fortunately, the SDK provides a function to align the color and depth images.

Task 8.6: Align Images (10 points)

The MATLAB function `determineExtrinsics()`, provided on LMS, will display the transformation from the depth camera to color camera. As a result of this, we are required to align the depth and color images so that both have the same origin. But the two images are not of the same size, so in aligning one to the other one of the images will have to be either downsampled or upsampled. You can align the depth image to the color image by adding the following lines of code to your code from Task 8.4 before selecting depth or color frame:

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

How would you align the color image to the depth image? Which one will you prefer? Why?

Task 8.7: Real-world coordinates (20 points)

Now that you have the two images aligned, determine the real-world coordinates of a point in the image using the perspective projective model. 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. Physically verify your computation. Remember that the obtained coordinates are with respect to the camera coordinate frame. Which camera? Don't forget the depth start point.

To complete our robotic system, we'll require our vision system to return world coordinates of point of interest in the base frame of the robot. Determine the homogeneous transformation that transforms coordinates from the camera frame to robot base frame.

8.4 References

1. Zabatani, Aviad, et al. "Intel realsense sr300 coded light depth camera." IEEE transactions on pattern analysis and machine intelligence 42.10 (2019): 2333-2345.

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

3. Intel RealSense Depth Camera SR300 Series Product Family Datasheet

4. https://github.com/IntelRealSense/librealsense/blob/master/doc/post-processing-filters.md

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

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

– Richard P. Feynman

9 Perception Pipeline

It is time to design the vision pipeline for your robotic system, which will be integrated with your motion control system. The tasks included in this lab will help you achieve the following objectives: (i) complete a pick and place system that locates all blocks in robot's workspace, picks them, and places them at a specified location; (ii) identify blocks of specific color in the workspace; (ii) identify and manipulate orientation of blocks to grasp them properly.

9.1 Vision-based pick and placer

Task 9.1: Obtaining a labeled image (20 points)

Acquire an image of the robot's workspace, from the RGB sensor of camera, and process it to obtain a labeled image of all the relevant objects in the scene. You can convert your image to grayscale and segment it using any method. Note that you'll have to strategize how to eliminate the irrelevant regions from your image. You can assume that the camera setup will not change and you can use any suitable MATLAB's image processing library functions. You are to submit your MATLAB code, a description of your strategy to remove unwanted objects, and a test case, i.e. an original image containing different objects and a colored labeled image using different colors to represent each object.

Your next task will be to identify the position of the cube. You'll have to certainly make use of the perspective projective model, but you'll have to address two issues: (i) differences in

depths because of variances in illumination; (ii) multiple faces in the image.

You would have observed in the viewer that the depth image displays different colors for the different regions of the board. This points to inaccuracies in the inherent depth determination process of the camera or variations due to ambient light conditions, or interference from other IR sources. In any case, we'll have to solve this problem to be able to determine real-world coordinates with reasonable accuracy. What can we do?

- We could take a depth image of our setup without any blocks, compare it to the actual height between camera and board, and obtain correction factor for Z at each pixel location. Averaging over a few snapshots would mitigate the effects of noise as well.
- We could make use of other information, related to the problem. We know the cubes to be of a fixed height, so we only need to determine the number of stacked cubes for our purposes, and not the exact depth.

In the setup discussed in class, camera was placed directly above a conveyor belt and only top faces of boxes were visible in resulting image. However, placement of camera in the lab may result in more than one faces of cubes to be visible in obtained images, depending on placement of cubes. What is the maximum number of faces that can be visible in the image? One possible solution to our problem is to make use of depth information to distinguish the different faces. Include depth information with every image point and use perspective projective movdel to obtain a cloud of points (X, Y, Z) for every region, i.e. 3D coordinates of all points making a region. Creating point clouds is a common practice in Robotics with other ranging sensors as well, e.g. LIDAR. MATLAB has built-in functions for point cloud processing as well. Look into `pointCloud`. Following information could be extracted from the point cloud.

- The 3D centroid of the cloud. The centroid will not be exactly at the center of cube.
- The minimum z value. Points of this z value will be part of top face.
- Limit of x and y.

There could be other solutions to this problem as well, for instance finding the lines in the image to identify cube. You're welcome to explore any solution of your liking.

Task 9.2: Picking and placing a cube (20 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 labs. You'll have to construct an algorithm for finding coordinates in the robot frame to be passed to motion pipeline. Remember to make adjustments for your end-effector frame assignment and include safety checks so that the arm never collides with base board. You're to outline your devised strategy for determining coordinates and submit MATLAB code corresponding to this task.

Task 9.3: Pile them up (20 points)

Randomly spread a number of cubes in your robot's workspace, including creating a stack of two cubes. In this task, your robotic system should pick all the cubes currently in its workspace and pile them at a pre-specified free region in the workspace. Your arm and end-effector should not collide with the cubes at any point. The cubes don't need to be placed in any specific arrangement in the pile, and you can drop a cube on the pile from a small enough height (doesn't make a large sound). You're to submit a video of this task, MATLAB code, and a flow chart of your code.

9.2 Color Segmentation

In this section, you're going to segment image based on color. Since we have not covered this in class, it requires you to study it on your own. You can use one of the two straightforward approaches: (i) transform your image 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 (`https://www.mathworks.com/help/images/color-based-segmentation-using-the-l-a-b-color-space.html`); (ii) automatically segment an image based on color using k-means clustering (`https://www.mathworks.com/help/images/color-based-segmentation-using-k-means-clustering.html`).

Task 9.4: Single out a color (20 points)

Randomly spread a number of cubes of varying colors in your robot's workspace. Implement a robotic system that sorts cubes by color and creates piles of cubes of different colors at specified free areas in the workspace. You're to submit a video and MATLAB code for this task.

9.3 Experimenting with shapes

Task 9.5: Blocks of a kind flock together (20 points)

Assume that the robot workspace is occupied by a random spread of the three objects of different geometries available in the lab. Implement a robotic system that collects objects of the same shape in a pile and creates three piles of objects in the workspace. You're to submit a video, MATLAB code, and a flow chart of this task.

Task 9.6: (Bonus) Going off the reservation (20 points)

Place a rectangular object in your robot's workspace in an orientation that makes it impossible for the robot to grasp it. Explore the possibility of nudging the block using the arm in just the right way so that it is in proper orientation for your robot to pick it.