JEM 455
MATLAB Programming Assignment 2:
3D Scanner

Due: 3/7/2023 @ 11:59pm

Save all MATLAB files necessary to complete this project into a single folder. Compress that folder into a zip file and submit to Moodle by the due date. All necessary code files to run your script file should be included in the folder. In other words, I should be able to unzip the folder, open your script in MATLAB, and run your code without any errors. Any errors will result in points being counted off. I advise you to zip it, send it to another computer, and run it to make sure all your files are present. You do not need to include data files (.mat files) that I give you in your submission.

Background

The Quanser QArm is a 4 DOF (4 rotational joints) robot arm manipulator (Figure 1) made by the Canada-based company, Quanser. It has a pincher-style end effector used to pick up light objects, rotational displacement encoders in each joint, and an RGBD camera attached to the top of the end effector to detect objects around it. The objective of this programming assignment is to turn this robot arm into a 3D scanner using the encoder data and RGBD data. Essentially, the robot should be able to collect data while someone moves the end effector to scan its surroundings and your program should be able to process that raw data to create a 3D point cloud of the surroundings. After this programming assignment is finished, I will bring the robot into class and we can run your programs on data it collects real-time.

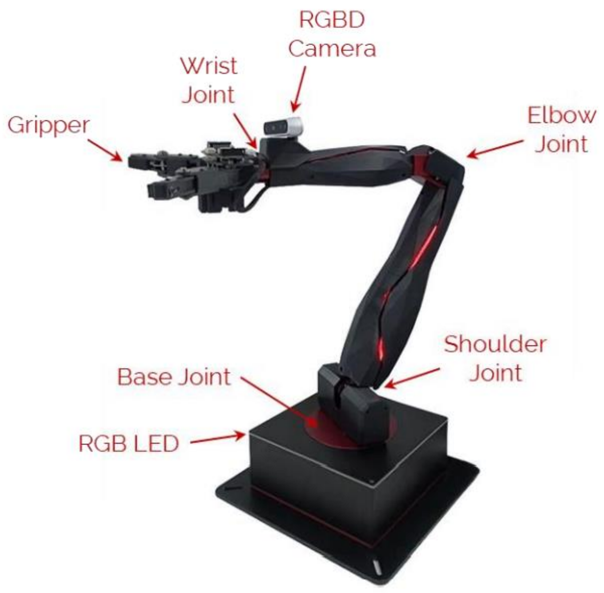


Figure 1: Quanser QArm

Specifics

Write a collection of MATLAB scripts and functions that can do the following.

- Load in the data collected by the robot
- Process the RGBD data to create a point cloud of objects with respect to the camera
- Process the encoder data and use forward kinematics to determine the pose of the end effector given joint positions
- Transform the local point cloud from the camera to a collection of global point clouds
- Display the 3D scan and a 3D trajectory of the robot arm's end-effector

Data

All data that you must process are in .mat files. They are named "Chair.mat" and "Chris.mat". These files contain 2 variables; `joint_pos` and `cam_d`.

`joint_pos`

The variable `joint_pos` is a $5 \times T$ matrix that contains the joint position of each joint in radians for T time stamps. From the first row to the last the data is the time stamp, joint position of the base, joint position of the shoulder, joint position of the elbow, and joint position of the wrist.

`cam_d`

The variable `cam_d` is a $480 \times 640 \times T$ matrix that contains the depth information of objects observed by the robot's RGBD camera in meters. Each "slice" represents an image taken in a single time stamp.

Functions

You must write the following MATLAB functions.

```
function [pc] = pointCloudFromImage(im, maxDist, minDist, pixelStep)
```

This function must do the following.

- Process a depth image of any size to create a 3D point cloud describing objects in reference to the RGBD camera
- Reject objects found outside of a range determined by the input parameters of the function
- Evaluates the pixels of the image using step sizes so that not every pixel has to be considered (see note below)

Inputs:

- `im` is a $R \times C$ greyscale image that holds the depth information of objects in the image. Each pixel holds the value of the distance of an object in meters.
- `maxDist` is a value that describes the maximum distance to be considered in creating the point cloud. Any object further than this threshold from the RGBD camera is rejected by the function and not included in the point cloud.

- `minDist` is a value that describes the minimum distance to be considered in creating the point cloud. Any object closer than this threshold from the RGBD camera is rejected by the function and not included in the point cloud.
- `pixelStep` is an integer that describes how many steps to take between pixels when reading their values to create the point cloud. For example, if the pixel step is 3, the function would read values `im(1, 1), im(1, 4), im(1, 7),` etc. Essentially, this tells you how many pixels to skip when going through the image. This must be applied to both the columns and the rows of the image.

Outputs

- `pc` is an $N \times 3$ matrix that contains the local 3D location of the points detected by the camera in meters.

Note: Figure 2 illustrates the local coordinate frame of the camera. Each pixel from the center pixel can be thought of as an angle displaced around the y_c axis (for pixels in the x direction) and the x_c axis (for pixels in the y direction). Figure 3 illustrates this concept. In the camera we are using, the resolution of pixels to angle is $\lambda_{px} = 0.0017\ rad/px$.

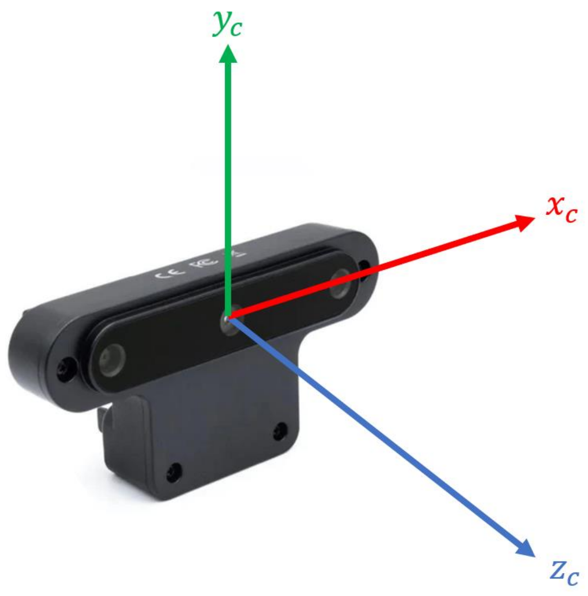


Figure 2: Local coordinate frame of the RGBD camera.

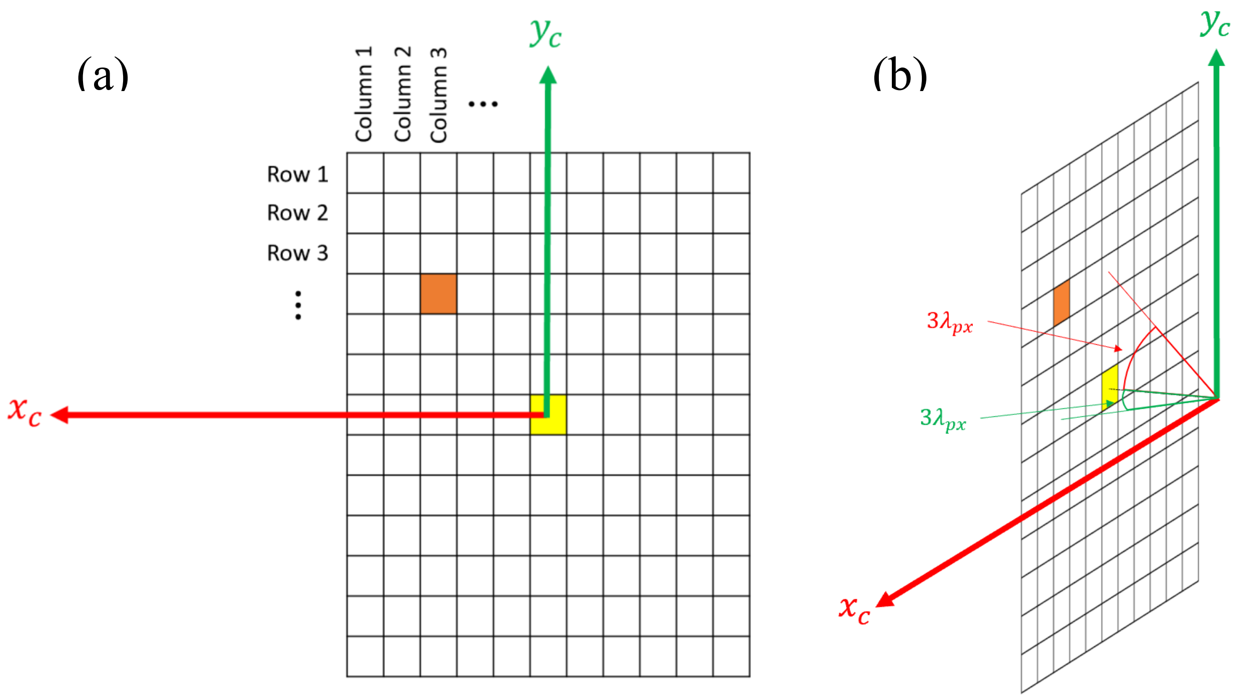


Figure 3: (a) View of a digital image from the perspective of the camera. Yellow pixel indicates the center pixel of the image and the orange pixel indicates a pixel of interest that contains an object. (b) The same figure from a side view showing that the angle of our object is 3 times a pixel angle resolution around both the x and y axes because the object was found 3 pixels away in the y and x direction respectively.

```
function [xi] = qarmPose(q)
```

This function must do the following.

- Process the encoder data from each joint and translate that into a collection of homogenous transformation matrices describing the pose of each joint's coordinate frame with respect to the global coordinate frame.
- Be able to indicate whether or not the encoder data is reading an invalid joint displacement for each joint. If an invalid joint displacement is detected, the function returns NaN into the output. Reference the table below to indicate whether or not a joint is outside of a valid displacement. We need this capability because we will use this function in a later project.

Inputs:

- `q` is a 4×1 vector that contains the configuration space of the robot. Each element of this variable is an angular displacement in radians of each joint in the order of base, shoulder, elbow, and wrist.

Outputs:

- `xi` is a $4 \times 4 \times 6$ matrix containing the homogenous transformation matrix of the global coordinate frame, each joint's coordinate frame, and the end effector coordinate frame all with respect to the global coordinate frame.

Note: To create this function you must perform the DH convention on the QArm robot. Use the diagram in figure 4 to do this. Use the values $L_1 = 0.14m, L_2 = 0.35m, L_3 = 0.05m, L_4 = 0.25m,$ and $L_5 =$

$0.065m$. Note that the coordinate frames of each joint have been created for you and that the global coordinate frame is not shown although it mimics frame 0.

Joint and gripper ranges	Base:	-2.9671 rad (-170°) to +2.9671 rad (170°)
	Shoulder:	-1.4835 rad (-85°) to +1.4835 rad (+85°)
	Elbow:	-1.6581 rad (-95°) to +1.3090 rad (+75°)
	Wrist:	-2.7925 rad (-160°) to +2.7925 rad (+160°)
	Gripper:	0 (0%) to 1 (100%)

Table 1: Extreme limits of each joint in radians and degrees.

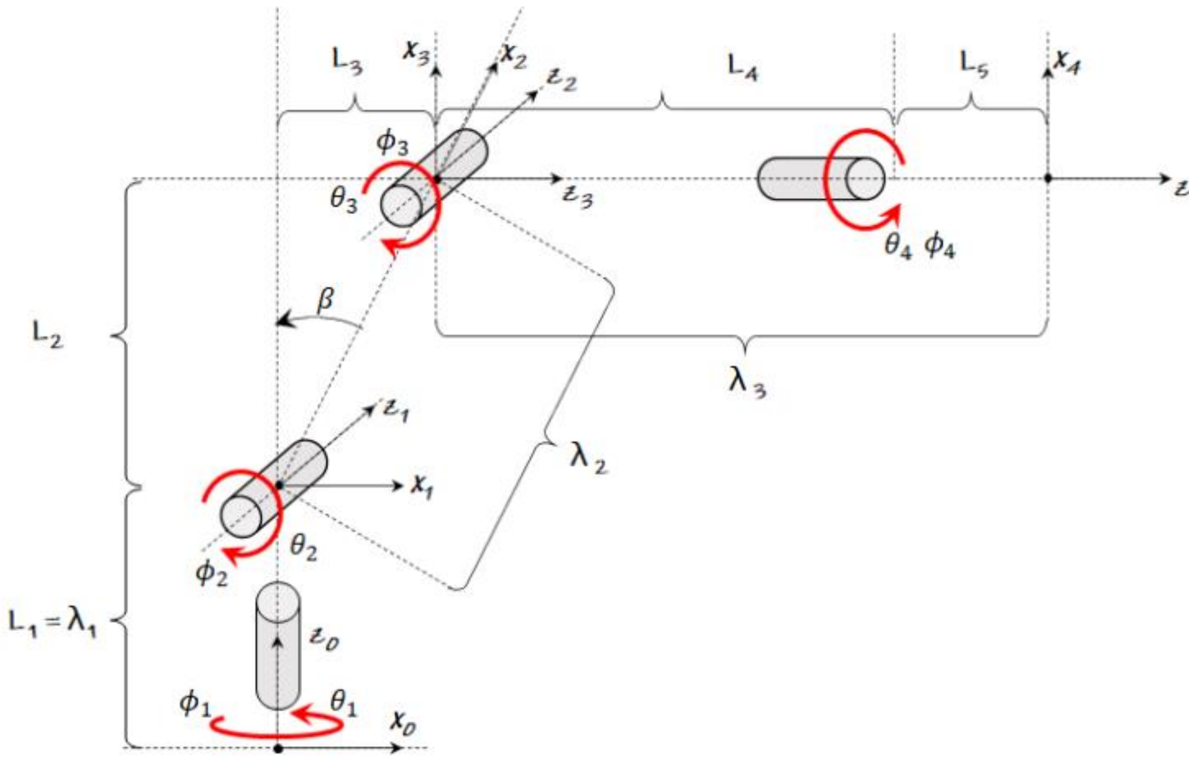


Figure 4: Diagram of the QArm in all-zero configuration with coordinate frames assigned according to DH convention.

```
function [pc_t] = qarmCam2Global(q, im)
```

This function must do the following.

- Process the encoder data in conjunction with the depth image to create a point cloud of global points for a single time stamp.

Inputs:

- `q` is a 4×1 vector that contains the configuration space of the robot. Each element of this variable is an angular displacement in radians of each joint in the order of base, shoulder, elbow, and wrist.

- `im` is a $R \times C$ greyscale image that holds the depth information of objects in the image. Each pixel holds the value of the distance of an object in meters.

Outputs:

- `pc_t` is an $N \times 3$ matrix that contains the point cloud with respect to the global coordinate frame for a single time stamp.

Note: It is important to know that the camera is positioned at $(x, y, z, yaw, pitch, roll) = \left(0.05m, 0m, -0.095m, -\frac{\pi}{2}, 0, 0\right)$ according to the ZYX 3-angle representation with respect to the end effector.

```
function animateTrajectory(locationTL, n, pauseTime)
```

This function must do the following.

- Process the location data of an end effector and plot every nth location on a scatter plot every `pauseTime` seconds.
- Plot the points as red dots (`.`).

Inputs:

- `locationTL` is a $3 \times T$ matrix that contains the x, y, and z location of the end effector for T time stamps.
- `n` is an integer indicating how many time stamps to skip between plotting a point. This is to avoid bogging down your processor and graphics card with thousands of points it's trying to plot.
- `pauseTime` is a decimal value indicating how long of a pause should be between plotting points.

Outputs: None

Note: Some functions you should research from MATLAB are `pause` and `scatter3`.

Script

Write a script that can load in any .mat file that contains run data and creates a figure with an animation of the robot arm's trajectory and a second figure of all global points collected by the robot's camera.

Grading Rubric

Script compiles without intervention	8 pts
pointCloudFromImage function	20 pts
Correctly maps image to point cloud	*14 pts*
Ignores values outside of range	*3 pts*
Skips pixels according to pixelStep	*3 pts*
qarmPose function	20 pts
Correctly calculates the end-effector pose	*15 pts*
Rejects values outside of mechanical limits	*5 pts*
qarmCam2Global function	20 pts
Correctly maps values to global	*15 pts*
Calls the qarmPose function	*1 pts*

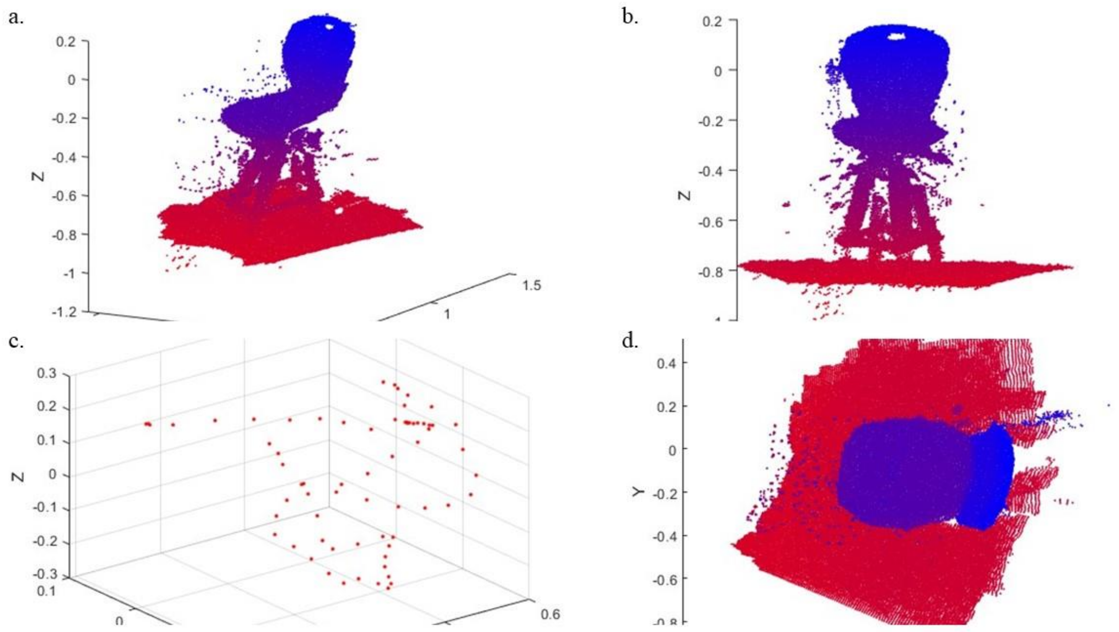
Uses camera coordinate frame	*4 pt*
animateTrajectory	10 pts
Plots all values	*3 pts*
Plots values timed out by pauseTime	*2 pts*
Skips values according to n	*3 pts*
Plots using red dots	*2 pts*
Script	22 pts
Loads the .mat file	*2 pts*
Plots 3D scan	*10 pts*
Plots 3D trajectory	*10 pts*

Test Cases

Below is a list of the test runs and the results you should receive. For these examples I used the following parameters.

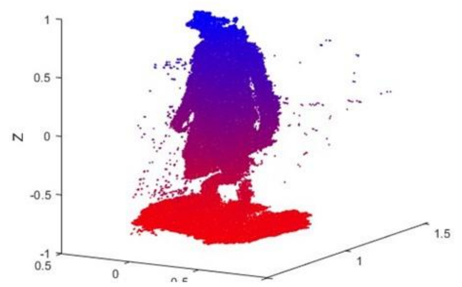
- `n`=1: Number of points to skip in `animateTrajectory`
- `pauseTime`=0.1: Pause time between plotting points in `animateTrajectory`
- `maxDist`=1.0: Maximum distance to consider in `pointCloudFromImage`
- `minDist`=0.2: Minimum distance to consider in `pointCloudFromImage`
- `pixelStep`=5: Number of pixels to skip in `pointCloudFromImage`

Chair.mat: a) Ortho view. b) Front view. c) End-effector trajectory. d) Top view.

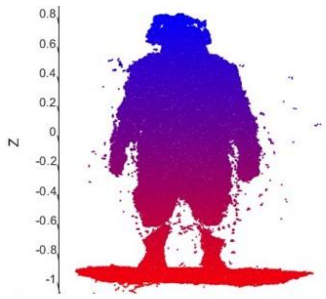


Chris.mat: a) Ortho view. b) Front view. c) End-effector trajectory. d) Top view.

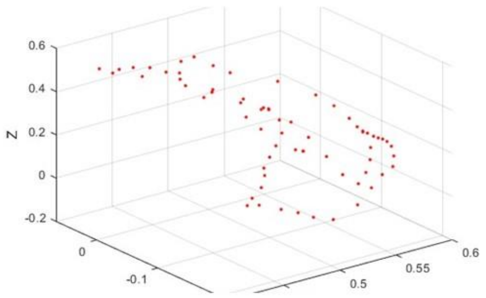
a.



b.



c.



d.