530.646 Final Lab Report

Group 15: Qiuyu Deng, Yeting (Hattie) He, David Samson

Main Project: Pick up and Place

General Program flow

To perform the pick up and place operation, our project makes use of a matlab GUI which controls the training process, selection of the control algorithm, and the block placing task. Running ur5_project.m, located in the main project folder, initializes the UR5 interface, the gripper, and the GUI.

First the user selects the control type using the dropdown in the GUI. The options are the required Inverse Kinematics, Resolved Rate Control, and Gradient Control, specified for the final project.

Next the user trains the starting and ending locations of the block. To train, the following steps are performed:

1. UR5 is jogged to the starting location with the gripper surrounding the block
2. "Record Initial Block Position" button is clicked in the GUI
    a. This causes the GUI to record the initial position of the block
3. UR5 is then jogged to the target location
    a. (identical to the start position relative to where the block would be, just centered over the desired location at the target)
4. "Record Target Block Position" button is clicked in the GUI to record the final position

Finally, to pick up and place the block, the user clicks the "Move Block" button, which runs the pick up and place algorithm described below.

Pick Up and Place Algorithm

The pick up and place process is outlined in the MoveBlock.m function in "ur5_final_project/Main Project/helpers". The function is called by the GUI, which passes in the initial and final positions, as well as the selected control type.

To pick up and place the block, the function does the following steps:

1. The function constructs a handle to the correct control function based on the specified type

    a. Additionally, arguments to be used by the controller such as move durations, gains, and thresholds as specified in an variable called args.
        i. Note that for the gradient controller, args needed to be different for each call, in order to be able to run in a reasonable amount of time, so the gradient controller does not make use of this args variable
2. The robot moves to our custom home position
3. The gripper opens
4. The robot moves to 10 cm above the starting location
5. The robot moves to the starting location
6. The gripper closes
7. The robot moves to 10 cm above the final location
8. The robot moves to the final location
9. The gripper opens
10. The robot moves to 10 cm above the final location
11. The robot moves to the home position

For each time the robot moves, the handle to the correct controller is called with the 4x4 transform, called *gst_target* in the code, for the desired location as well as the arguments created for that transform. *gst_target* is a homogeneous transformation matrix of the end effector frame, with respect to the baselink of the ur5. As the joint angles of start and target position are recorded by the program, f_ur5FwdKin.m is used, to transform a 6x1 matrix indicating joint angles into a 4x4 homogeneous matrix, that *gst_target* will be obtained in this way.

Note that again, the call to the function handle has extra arguments after args which are only used by the gradient controller (This should probably be refactored in the future, but it works for now).

Additionally, between each gripper command, and movement command, the *waitfobuttonpress* function is called, to ensure that the current task is completed before the robot moves onto the next one.

Inverse Kinematics Method

1. The function file named IK_control.m is called in the file named MoveBlock.m
2. Due to the default offset existing in ur5, another homogeneous transformation, *g_T_tool_K*, needs to be applied on the gst_target to correct the direction of of the end effector frame.
3. For each time when *gst_target* passes into this function, *ur5InvKin* function will be used to transform the 4*4 homogeneous matrix in to a 6*8 matrix which indicates the 8 sets of values of joint angles that the ur5 can perform to get the desired position.

4. The best set of joint angles will be selected via *get_best_theta* function to make ur5 go through shortest path.
5. Finally the ur5 will move to the selected joint angles.

Gradient Control Method

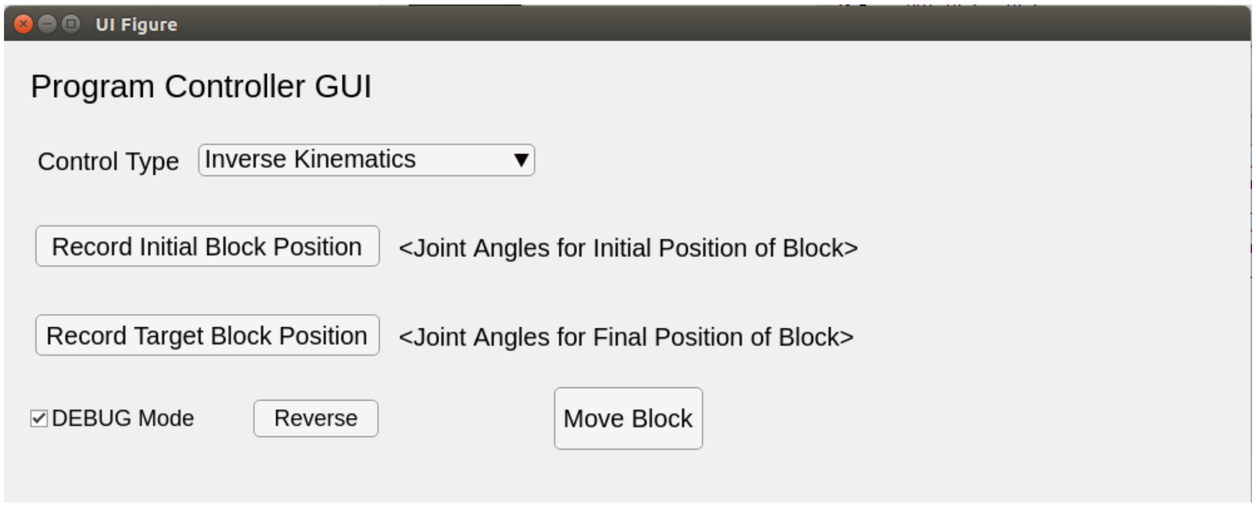
1. gst_target, K, threshold and scale are the inputs for *Grad_control* function call. K is the gain of the system which will provide power to ur5 movement and make ur5 get closer to the target position gradually. Threshold determines the minimum value of the normalized twist, and if the actual threshold is less than the desired threshold, ur5 will stop moving. That happens when ur5 gets close enough to the desired position. Scale helps to modify the second argument of *ur5.move_joints* function, which represents time interval.
2. There is a loop contained in this function. Every time when ur5 moves to position, the joint angle values will be recorded and updated and then goes into the loop again.
3. Body jacobian for end effector of ur5 is obtained by *f_ur5BodyJacobain* function.
4. Determinants of body jacobian matrix for joint angles will be checked for singularity. If singularity exists, the whole program will stop.
5. Xi is a 6*1 matrix that represents the twist of the movement and can be gotten from *f_getXi* function. Basically, the normalized twist will be used to check whether the actual threshold value satisfies the desired ones.
6. New joint angles, *q_next,* will be calculated from the equation *q_next = q_current - K * step * transpose(J) * xi;*
7. The scale of time interval has a relationship with the distance of joint angles., which can change the speed of ur5 when undergoes movements.
8. Ur5 will move to new position as the value of *q_next* is obtained.
9. The time interval, the gain and the threshold required for a new movement in this gradient method is really picky at different stages, so there should be some modification when setting the those input values.

Resolved Rate Control Method

1. *DK_control* function file only contains one input, which is gst_target.
2. The algorithm used for this method is almost the same as the one used for gradien control method, except the fact that the transpose of Jacobian will be changed into inverse of Jacobian, and the gain value of the system for whole process remains the same.

GUI Driver

The GUI controlling the block placing task contains the following functionality


UI Figure
Program Controller GUI
Control Type
Inverse Kinematics
Record Initial Block Position
<Joint Angles for Initial Position of Block>
Record Target Block Position
<Joint Angles for Final Position of Block>
☑ DEBUG Mode
Reverse
Move Block

GUI element:

- Control Type Dropdown
    - Allows the user to specify the control algorithm to use.
    - Options are Inverse Kinematics, Resolved Rate Control, and Gradient Control
- Record Initial Block Position Button
    - Saves a copy of the UR5 current transform from base_link to T
    - Displays as text the current UR5 joint angles
        - This is mostly just for feedback to the user. The joint angles aren't actually used.
- Record Final Block Position Button
    - Exact same as the initial position button
- DEBUG Mode check box
    - Sets the global variable DEBUG to true if checked or false if unchecked
    - DEBUG mode prints out extra output during the move block process
- Reverse button
    - Swap the initial and target positions that were saved
    - Useful for having the UR5 move a block back to start after placing it. No need to retrain
- Move Block button
    - Calls the MoveBlock function with the initial and final transforms for the block positions
    - Pressing this button disables all graphical elements until the move block sequence has completed
    - After the sequence completes, all graphical elements are re-enabled, for use in another block moving task

The GUI saves all previously recorded data, so training is only necessary when the GUI is initially opened, or the position of the block changes from what was trained.

Extra Project: DrawBot

Description

For the extra portion of the project, we decided to make the UR5 draw pictures on a whiteboard.

General Program Flow

Similar to the move block program flow, the DrawBot functionality is managed through a matlab GUI. Running ur5_project_extra.m, located in the extra project folder, initializes the UR5 interface, and the GUI.
First the user must attach the drawing tool to the robot. See below for more details about the drawing tool.

Next, the user must train the robot where the origin of the board is, so that it knows where to draw the picture. To train, the following steps are performed:

1. UR5 is jogged to the point where the drawing tip is touching the origin of the board (upper left corner), and the tool frame is oriented completely vertically (i.e. the marker is perpendicular to the board).
2. “Record Origin” button is pressed in the GUI
3. UR5 is jogged to a point along the x-axis of the board (downward from the origin), again with the drawing tip touching the board, and the drawing tool oriented perpendicular to the board
4. “Record e1” button is pressed in the GUI
5. UR5 is jogged to a point along the y-axis of the board (to the right of the origin), again with the tip touching the board and the tool oriented vertically
6. “Record e2” button is pressed in the GUI

Next the user must select an image to draw. The drawbot can draw any jpg or bmp image. To select an image, the user clicks the “Select Image” button in the GUI, which opens a file selection dialog window. The user can then select the image they want to draw

After the user selects an image, the can click the “Preview Image” button in the GUI, which will show a preview of what the robot will draw. Additionally, the number in the “Number of Levels” text box can be adjusted to change how detailed of a recreation of the image will be drawn (larger number means more details).

Finally, the user clicks the “Draw Image” button in the GUI which runs the draw image algorithm described below.

Drawing an Image Algorithm

The image drawing algorithm is outlined in the draw_contour.m file in “ur5_Final_Project/Extra Project/helpers”. The function is called which passes in the constructed origin frame of the board, as well as the (upright) orientation of the drawing tool, and a contour matrix containing the contours used to draw the image. Additionally there is a scale parameter which is used to ensure that the image drawn fits within the drawing area.

To draw an image, the function does the following steps:

1. The robot is moved to slightly above the first point to be drawn
2. Matlab calls the waitforbuttonpress function to ensure that the user is ready to begin
3. While the contour matrix is not empty, the following loop sequence is run
    a. The next contour is extracted from the contour matrix
    b. The contour is converted from 2D space into 3D world space
    c. The robot lifts the drawing tip up slightly off the board
    d. The robot moves to above the first point in the contour
    e. The robot drops the drawing tip onto the board
    f. The robot traces out the contour using the Inverse Kinematic controller
    g. The contour is removed from the contour matrix
4. The loop is repeated until there are no more contours to draw, at which point the robot moves to the home position, indicating that the drawing is complete

Constructing the Board Origin Frame

Relative to some image to be drawn, the origin frame used for this portion of the project is in the upper left corner, with the x-axis pointing downward, and the y-axis pointing across to the right (and z-axis pointing up out of the board). So in order to tell the robot correctly where to draw, we must put the coordinate frame in world space in the upper left corner of our drawing area with x pointing down and y pointing to the right relative to whatever we are drawing on.

To actually create the coordinate frame, the user places the drawing tip at three points: the origin, a point along the positive x direction, and a point along the positive y direction. When recording the points, the user must make sure to maintain the orientation of the ur5 tool frame, as technically the tool frame is what is being recorded, not the tip of the drawing implement. The program then takes translational coordinates from the x, y, and origin frames, and uses it to construct the board origin coordinate frame.

Given an origin and points along the x and y axis in 3D space, the origin frame is constructed as follows:

1. X_hat is set as the x point minus the origin, normalized
2. Y_hat is set as the y point minus the origin, minus the projection onto X_hat, normalized
3. Z_hat is set as the cross product of X_hat and Y_hat

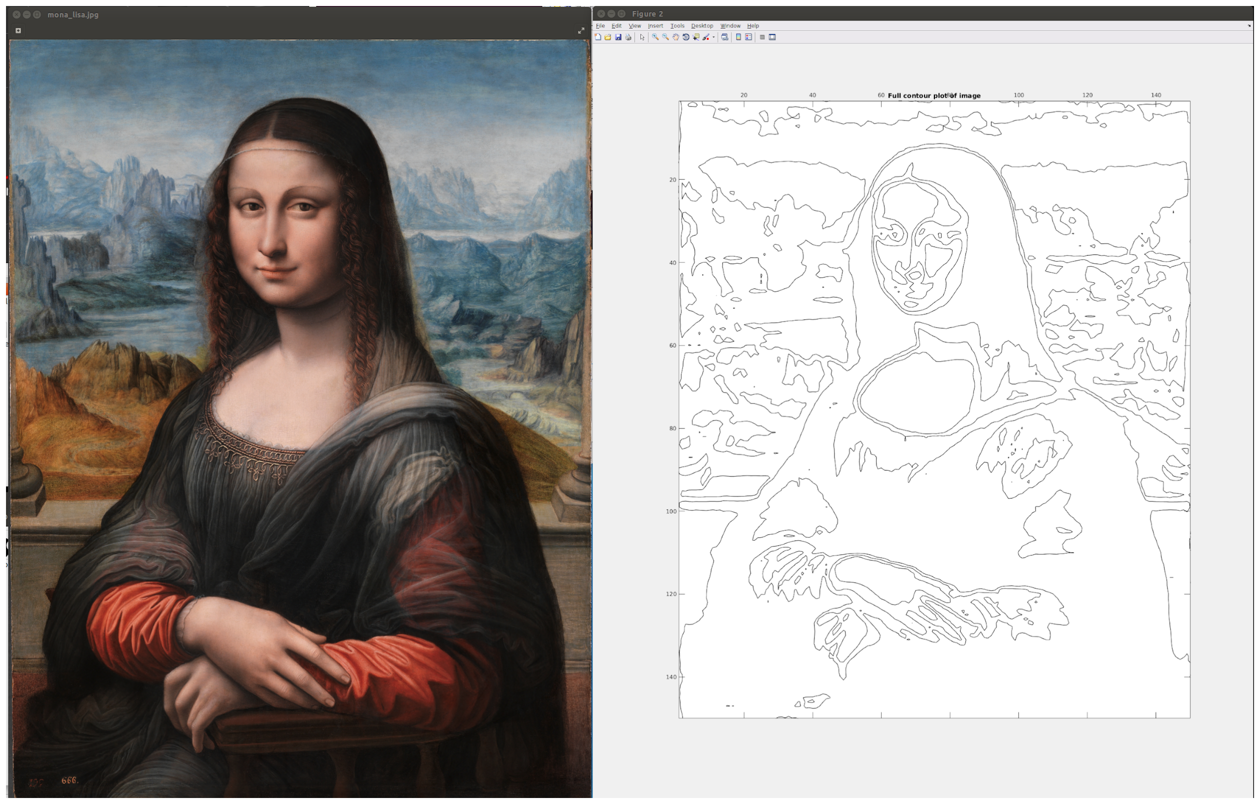
4. The rotation of the origin frame is constructed as [X_hat Y_hat Z_hat]
5. The 4x4 transform is constructed as [rotation origin; 0 0 0 1]

Creating Image Contours

The DrawBot can draw any image that matlab can read. This is accomplished using matlab's contour function, on the image. The process uses the following steps:

1. The image is read into matlab using imread()
2. If the image is a color image, it is converted to grayscale
3. The contours are extracted using contour(image, levels)
    a. Levels is used to specify how many level set contours to divide the image into

The way that this works is by finding the level sets of constant brightness within an image. While not a perfect recreation of the image, it is an easy way to convert an image into something that the robot can draw.


mona_lisa.jpg
Figure 2
File Edit View Insert Tools Desktop Window Help
Full contour plot of image
20
40
60
100
120
140
20
40
60
80
100
120
140

Example of the Mona Lisa converted to contours using the contour() function

Converting 2D contours into 3D worldspace

Converting a contour from 2D space to 3D space is relatively straightforward. The contour matrix provides each contour as a stream of x and y coordinates for points in 2D space. So a contour looks something like this:

$$\text{Contour} = \begin{bmatrix} x(1) & x(2) & x(3) & \dots \\ y(1) & y(2) & y(3) & \dots \end{bmatrix}$$

I.e. a contour is a series of column vectors for each point in the contor. The first step in converting the contours into world space is to represent them in homogeneous coordinates:

$$\text{Local\_Path} = \begin{bmatrix} x(1) & x(2) & x(3) & \dots \\ y(1) & y(2) & y(3) & \dots \\ 0 & 0 & 0 & \dots \\ 1 & 1 & 1 & \dots \end{bmatrix}$$

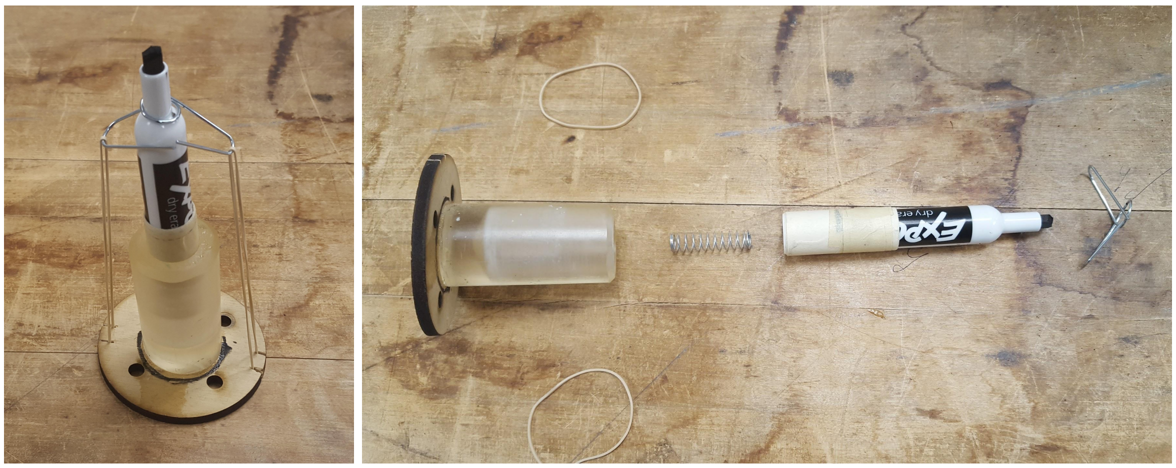
Lastly to place the contours on the board, as defined by the board origin transform (tf_board):

$$\text{World\_Path} = \text{tf\_board} * \text{Local\_Path}$$

Drawing Hardware

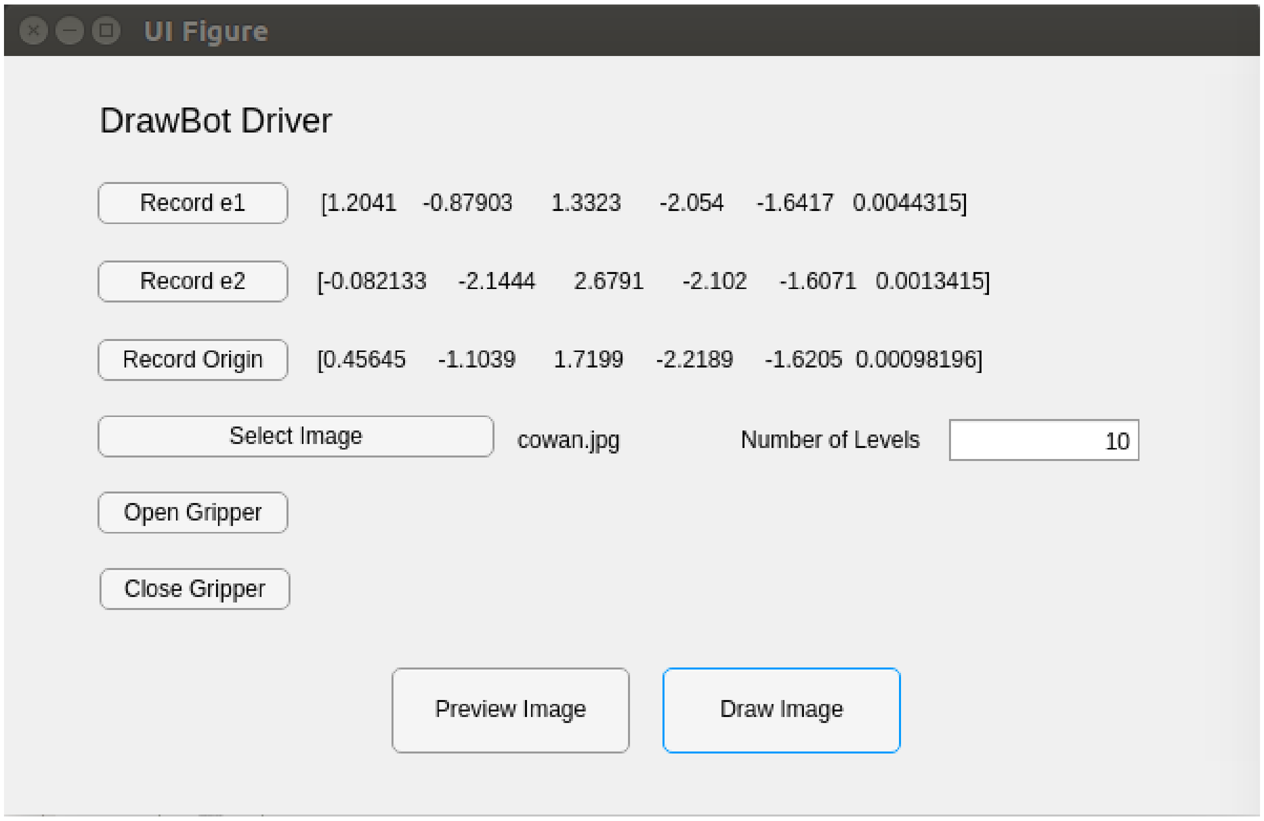
To allow for a tolerance in the robot drawing on the board, a custom drawing tool was fabricated. The tool allows the robot to draw on the board with up to roughly ± a centimeter of error in vertical direction. The allows for the board origin training to be less precise, as well as allowing for the board to not be perfectly flat.

The tool is constructed from a rod of plastic with a blind hole drilled into it slightly larger than a dry erase marker. A spring is placed in the bottom of the tube. The marker is wrapped in tape to fit exactly into the tube. The tube is lubricated, and the marker is placed into the tube. Lastly tension is provided against the spring by rubber bands, ensuring that the marker does not fall out of the tool during use. The tool is glued to a circle of plywood with holes that allow it to be fixed to the UR5 as an end effector tool


Drawing tool with a marker (left) and its components in an exploded view (right)

DrawBot GUI

The GUI controlling the block placing task contains the following functionality



[1.2041 -0.87903 1.3323 -2.054 -1.6417 0.0044315]

[-0.082133 -2.1444 2.6791 -2.102 -1.6071 0.0013415]

[0.45645 -1.1039 1.7199 -2.2189 -1.6205 0.00098196]

cowan.jpg

Number of Levels: 10

GUI Element:

- Record e1 button
    - Records the current location of the UR5, and saves it as the location of a point along the x-axis of the origin frame of the board
- Record e2 button
    - Same as record e1 button except records a point along the y-axis of the origin frame of the board
- Record Origin
    - Same as record e1 and record e2, except records the location of the origin of the board frame
- Select Image button
    - Opens a dialog box for to allow the user to select an image for the robot to draw
- Open Gripper / Close Gripper buttons
    - Sends a signal to open/close the gripper. Not actually used because the drawing tool is fixed directly to the UR5
- Number of Levels text field

    - Allows the user to specify how many levels of brightness to extract contours for from the image
    - Making this larger increases the detail in the drawing
- Preview image
    - Plots the contours of the image to be drawn
- Draw Image
    - Commands the robot to draw the picture on the whiteboard
    - The points trained for e1, e2 and origin are used to construct the origin frame which places where the image will be drawn in space
    - The picture is drawn according to the drawing and image algorithm specified above

Experimental Results

Main Portion

The block can be picked and placed successfully, which can be verified in the attached video. Inverse kinematics method can generate the most precise movement because the calculation is really straightforward. However, this method requires exploitation of the geometry of different robotic manipulators which might be time consuming to verify the method.
Resolved rate is a numerical method that makes the robot moves along an axis which the end effector can screw to reach the desired configuration. Visually, ur5 moves smoothly using this method.
Gradient rate control generates unpredictable motion because it is an optimization based method.
See the first portion of the video for these results:
https://www.youtube.com/watch?v=FeDsPanEYDo

Extra Portion

The DrawBot was successfully able to draw arbitrary images. The drawing tool allowed the marker to remain in contact with the board over the entire drawing space, and the training process was successful at specifying an origin to the board for the drawing space.

The main issue with running the drawing program is the interface between matlab and the UR5. The contours are extracted as paths for the robot to traverse in world space, so the best case would be having the ability to directly command the joint velocities such that the robot would smoothly trace the contours. Using the matlab interface to stream points to the robot with our Inverse Kinematic controller, works well enough, but an image may contain tens of thousands of points in all of its contours. Occasionally the ur_interface would appear to drop a point (i.e. matlab would command the robot to move, and it wouldn't move, which can be seen as the robot pausing for 10 seconds in the results video). The controller would be able to recognize that the robot wasn't actually moving, and continue on to the next step, but if the last step had the robot in contact with the board, the robot would make a streak across the image when it moved on. For the most part, this was the main issue encountered, and ultimately the robot was still able to draw the images successfully.

See the second half of the video for examples of drawing pictures:
https://youtu.be/FeDsPanEYDo?t=2m13s

Workload Distributions

Qiuyu

- Wrote and tested Resolved Rate Control

Hattie

- Wrote and tested Inverse Kinematics Method
- Wrote and tested Gradient Control Method

David

- Created GUIs
- Organized project layout/interfaces
    - Wrote moveblock function, set to use the user specified control type
- Implemented the drawbot for the extra task
    - Wrote algorithms
    - Fabricated the marker holder

File Descriptions

Main Project Folders and Files

Main Project

- ur5_project.m: Driver program for the move block portion of the project. Running this starts the GUI and initializes the UR5
- example_poses.m: commands for putting the UR5 into several example poses
    - Mostly used for testing

Main Project/controllers

- IK_control.m: IK_control moves the UR5 to the destination with inverse kinematics

- DK_control.m: DK_control moves the UR5 to the destination with differential kinematics
- Grad_control.m: Grad_control moves the UR5 to the destination with gradient

Main Project/helpers

- DriverGUI.mlapp: app for GUI to run during the move block portion of the project
    - To look at the GUI code, open appdesigner from the matlab console, open this file with the app designer, and then switch to code view
- MoveBlock.m: Function for controlling the process of picking up and placing the block (as described above in the pick up and place algorithm)
- ctrl.m: enum containing the different control methods available
- get_above.m: returns the transformation that is above a given transform matrix
    - This is used to get the transform that is 10 cm above the block initial and target locations during the move block process
- get_best_theta.m: selects the best theta value from the result of the inverse kinematics, based on the current configuration of the UR5
- move_fastest.m: moves the ur5 to the desired position as quickly as possible
    - An optional duration parameter can be passed in, which ensures that the move command will take at least that long to run
- pos_practival.m: determines if a particular joint position is practical
    - Used to eliminate impractical solutions from the inverse kinematics
- set_debug.m: sets the value of the global DEBUG parameter
    - This is mainly a handle for the GUI to be able to set the global DEBUG value
- ur5_home.m: our custom home position without singularities for the ur5
- wati_for_goal.m: wait_for_goal pauses the program execution until the UR5 is at the goal
    - This can be passed either the desired joints the ur5 should go to, or a transform from base_link to T that the robot will reach

Main Project/Resolved Rate Controller

- For the project, we used Qiuyu's code from lab 3 for the resolved rate controller. Because we ran the project on David's computer, we had to change the filenames of the resolved rate code to make sure there were no name conflicts (each filename was prepended with "f_"). This folder contains all of the resolved rate controller functions from lab 3, that are used on this project.

Extra Project Folders and Files

Extra Project

- ur5_project_extra.m: Driver program for the DrawBot portion of the project. Running this starts the GUI and initializes the UR5
- Example_Board_Origin.m: commands for placing the UR5 at a sample of x, y, and origin points, for constructing the board origin frame
    - Mostly used for testing

Extra Project/helpers

- DrawBotDriverGUI.mlapp: app for GUI to run during the DrawBot portion of the project

    - To look at the GUI code, open appdesigner from the matlab console, open this file with the app designer, and then switch to code view
- draw_contours.m: draws the contours passed in onto the drawing area in 3D world space, as outlined in the drawing an image algorithm above
- first_in_contour.m: extracts the first point from a contour
    - Used to place the robot above the first point in a contour before drawing it
- lift_pen.m: commands the UR5 to lift the pen off the page, so that it does not make a mark when moving to the next contour image
    - Passing a negative height to this functions as placing the marker onto the board
- lift_transform.m: similar to lift_pen, however it simply returns a transformation matrix instead of actually commanding the UR5 to move

Extra Project/pictures

- This folder contains a series of sample pictures for the UR5 to draw