Weiqi Wang

M.E. 530.646 Final Project

Final Project

Initiation

The program is capable of utilizing three position points from the white board to determine the normal unit vector of the board. Additionally, if the three points are each located on the left, top and right edge of the board, the program could also calculate the center position of the board. However, since the three points on the board provided do not fit the requirement, the center of the board was input manually. The calculation of the normal vector was accomplished by subtracting the any combination of two points to receive two vectors, and then determining the cross product of the two vectors. To normalize the vector, the result is divided by its norm. To convert the norm to orientation of the plane, a rotation matrix is constructed using the equation:

$$R = [n \times e_1 \quad e_1 \quad n]$$

Where n is the normal vector, e_1 is the unit vector in x direction, this ensures that the drawing's orientation is correct. The transformation from origin to the center of the board is then:

$$g = \begin{bmatrix} R & p \\ 0_{1\times3} & 1 \end{bmatrix}$$

With P being the position of the center of the board.

The program also allows the user to implement any image as the drawing. MATLAB converts the custom image to a black and white binary image based on the grayscale of the image and will find all the boundaries between the black and white colors, these values will be stored as lines in a cell array. Figure 1, 2 and 3 serve as the illustration of the changes of the images.

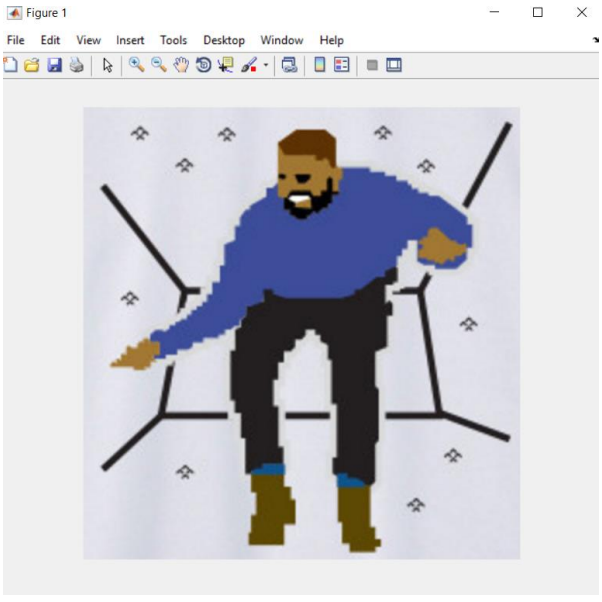

Figure 1. Original Image

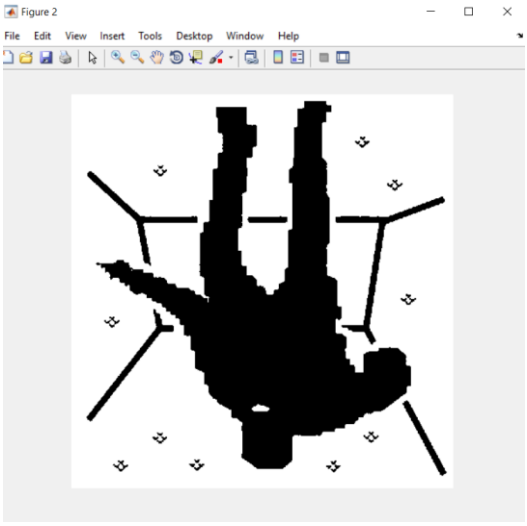

Figure 2. Black and white inverted image

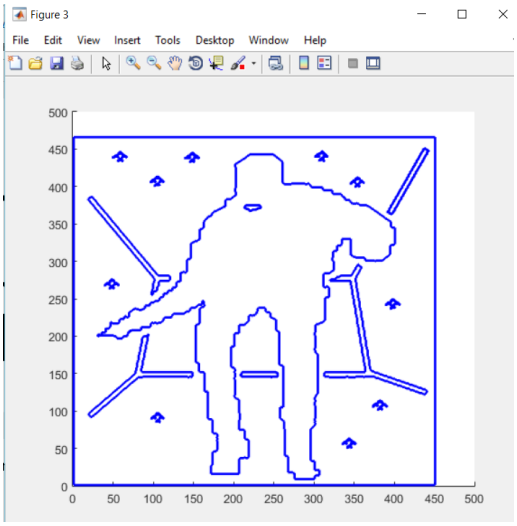

Figure 3. Final line conversion

Each element in the cell array is a nx2 matrix, where each row is a corresponding x and y set of values of a single point of the line. In rare cases with complicated images, the drawing will be decomposed into numerous lines, which could take significant amount of time to draw.

Since the matrix indexing and pixel indexing are inverted in the MATLAB. The image data is flipped in the calculation process, and rescaled to fit in a 15x15cm plane. Figure 4 is another demonstration of the program at work.

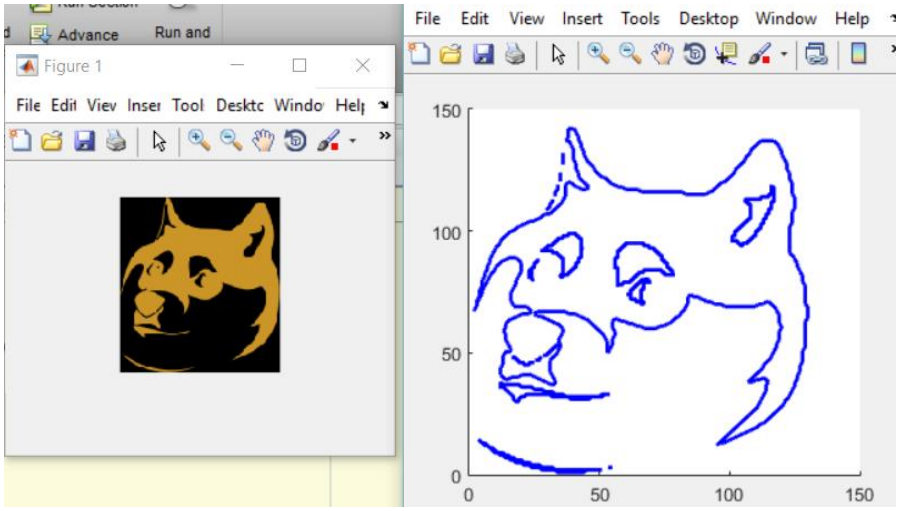

Figure 4. Another image conversion

Inverse Kinematics

The code for the simulation is named: Simulation_Inverse.m

The code for the real robot is named: Robot_Inverse.m

At the beginning of the drawing process, the current drawing point will first be allocated on the original x-y plane, as if the drawing board is placed at the origin. Multiplying this point's position with the transformation matrix will return the position of the point on the board, using this point and the rotation matrix of the board in the previous equation will produce the desired transformation matrix for the robot. In simulation, a small sphere is also placed at the end location to illustrate the drawing as seen in figure 5.

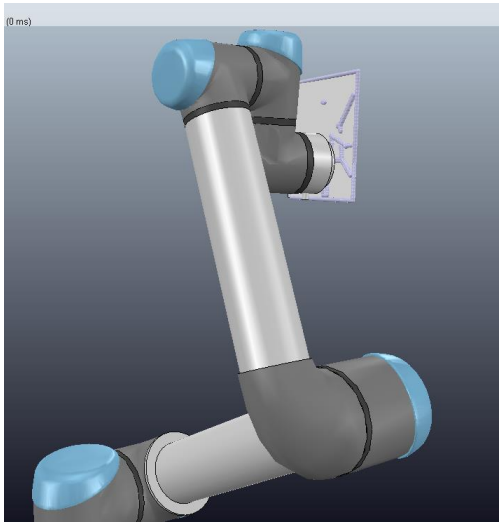

Figure 5. Simulation Drawing

Applying the previously written inverse kinematics on the transformation matrix yields a column of six angles, which is sent to the real life robot or the simulation. In addition, in order to prevent the robot from producing lines that the user does not wish for, in the program to command the real robot, the transformation matrix for the first and last point of each line has an offset of 2 cm in the y direction, allowing the robot to lift up the pen at the beginning and the end of each line. This modification was not used in the simulation, as it tends to slow down the simulation as the number of spheres increases. The program also allows a delay of 1 second in the said actions, since the end-effect must travel a significantly longer distance. However, in the line drawing process, no delay or error checks are implemented. The final image in simulation is shown in figure 6.

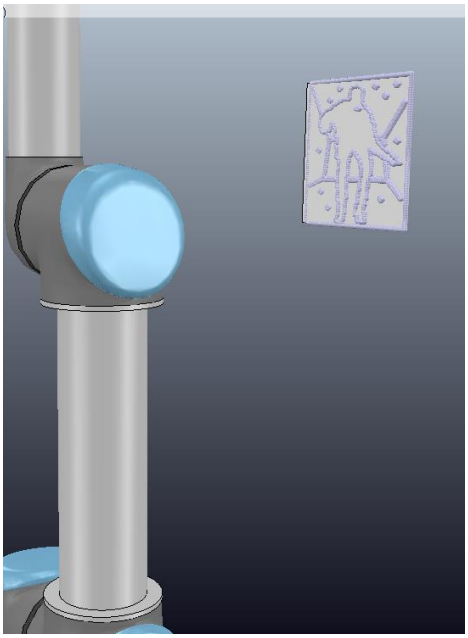

Figure 6. Simulation final drawing

A video of the real robot drawing can be viewed using the following link:

https://www.youtube.com/watch?v=p_dvInNqHiQ&feature=youtu.be

Differential Kinematics

The codes for the simulation is named: Simulation_Differential.m

The code for the real robot is named: Robot_Differential.m

This method allows the implementation of proportional control in manipulating the robot. At the beginning and the end of each line, identical to the inverse kinematics, the program produces the transformation with the pen offset and uses inverse kinematics to determine the joint angles. The differential kinematics method is only applied in the line drawing process. The method features applying the following equation in the calculation for the transformation matrix:

Weiqi Wang

M.E. 530.646 Final Project

$$q_* = q + k * J_s^{-1}(g_*^{-1}g)^\vee$$

Where q is the column of current joint angles, q_* stands for the desired joints angles after the adjustment; g is the current transformation matrix of the end effector retrieved from simulation of the robot, g_* is desired transformation, or in most cases, the transformation matrix for the next point, k is the proportional control factor, which is usually 1. The inverse of the spatial Jacobian matrix was also used in this equation, written as J_s^{-1}, and the un-hat function $^\vee$ allow a transformation from a 4x4 matrix to a 6x1 matrix as seen in equation 4:

$$\begin{bmatrix} \widehat{w} & v \\ 0 & 1 \end{bmatrix}^\vee = \begin{bmatrix} v \\ w \end{bmatrix}$$

Where v and w are each a 3x1 matrix.

It should be noted that although the use of spatial Jacobian matrix works for this specific task, it is in fact an erroneous approach. Since the orientation of the board does not change as the robot is drawing, the velocity of the end point with respect to the world frame origin is

$$\dot{p} = v_s + w_s p$$

With $w_s = 0$, meaning the point velocity is just the v_s. This allows the spatial Jacobian to work in this specific task.

However, in case the board's orientation does vary, the issue could be fixed using the body Jacobian, using the equation:

$$q_* = q + k * J_b^{-1}((g^{-1}g_* - I)^\vee$$

Where the I is a 4x4 identity matrix and J_b is the body Jacobian matrix.

Since the orientation of the white board is fixed in the project, the spatial Jacobian is implemented in the program for the real robot. However, in the simulation, the second method was adopted and both method have proved to work.

As seen in figure 7 and 8, the simulation was successful. A video of the real robot is also filmed and can be accessed using the link:
https://www.youtube.com/watch?v=5_y_SJazQT8&feature=youtu.be

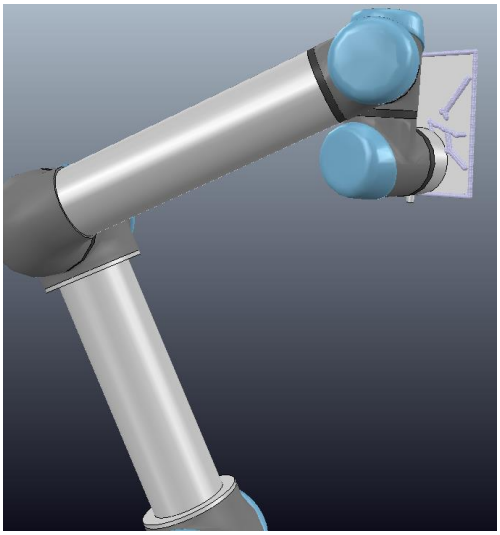

Figure 7. Differential Kinematics Simulation

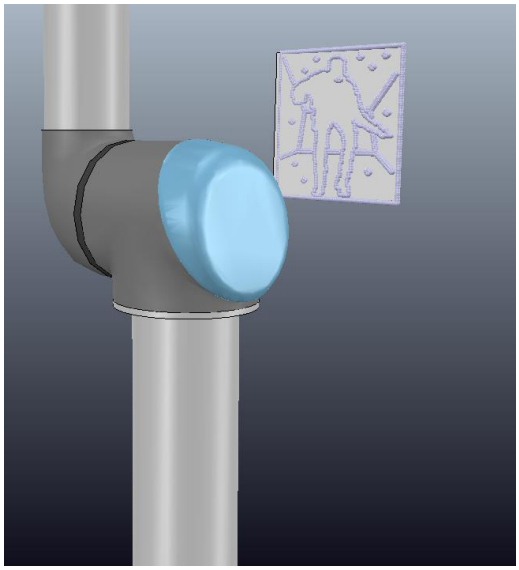

Figure 8. Differential Kinematics Completed Drawing