Josh Roney
jpr87
EECS 373
PS7

For this assignment I was given some existing example code that moved an irb140 robot arm to a specific spot, pick up a block that was at the spot with its vacuum gripper, raise it up, and then drop it. I was tasked to modify this code to allow the arm to pick up the block at any coordinate that the block was at via data from a camera. To start, I had to transform the pixel coordinates of the camera into robot coordinates by multiplying the block's camera coordinate matrix (scaled into meters) by a 4x4 transformation matrix. After transforming the coordinates I then had the find_block node publish these coordinates to a topic that the example_block_grabber node was subscribed to. This node then used to data to command the arm to move and pick up the block and place it at my chosen coordinates of (0, 0.5) with an orientation of (0,0,0,1).

To accomplish the transformation I used what I learned from EECS 489 Robotics I to find a transformation matrix fitting for the camera coordinates. I knew that the 4x4 matrix was to be comprised of a 3x3 rotation matrix and a 3x1 translation matrix. The rotation matrix was a standard z-axis rotation matrix by an angle x in the form of

$$\begin{bmatrix} \cos(x) & -\sin(x) & 0 \\ \sin(x) & \cos(x) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Through experimentation I found the angle x to be about 0.205048 radians. The translation matrix was the position vector from the origin of the robot coordinates (0,0) to the origin of the camera's viewpoint in robot coordinates (0.543 , 0.321). The last row was the quaternion notation of [0 0 0 1]. I noticed in my work that the camera coordinate framed had a mirrored y-value to that of he robot coordinate frame so I multiplied the sine and cosine of the second row (which contributed to calculating the new y-value) by negative one to account for this. This created the matrix that allowed me to transform the camera coordinate position vector to robot coordinates of

$$\begin{bmatrix} \cos(0.205048) & -\sin(x0.205048\ ) & 0 & 0.543 \\ -\sin(0.205048) & -\cos(0.205048) & 0 & 0.321 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

In the find_block node I set my published outputs to be equal to result of this matrix multiplication of the above matrix by the quaternion representation of the camera coordinates. This was published for the example_block_grabber node to receive as input. My example_block_grabber node was modified from the initial example code to use the received x and y position data as inputs for the robot's destinations rather than a hard-coded position. The robot starts by moving out of the camera's view to allow for detection of the block. After moving it waits a short time to ensure it receives the block's location. It then descends to hover just above the block before descending yet again the make contact with the block and grip it. When the block has been gripped the arm ascends straight upwards with the block and then moves to my chosen coordinates where it descends down and places the block on the ground. It then repeats this process, updating the position of the block with every run-through.

While working on this assignment many strange things happened that I could not understand. There were several times in which the gripper simply wouldn't activate forcing me to kill all the processes and relaunch them until the gripper would communicate. I also was unable to understand how to control the orientation of the tool to rotate the block to my desired orientation.

Although I was unable to control the orientation I had an idea to display the yaw (rotation about the z-axis using the camera. When the camera converted the image to what is seen in OpenCV I would record the coordinates of the pixel in the minimum x column and the pixel in the minimum y row. This would give the coordinates of the two corners of one edge. Using these two coordinates I would calculate the angle that the edge was at using the equation (theta = atan((x2 – x1) / (y2 – y1)). Then I would analyze whether I needed to add 90 degrees (pi/2 radians) to the calculated angle depending on a few cases. Lastly I would output the final value using the ROS_INFO to see the angle of the block.