Lab 5: Robotic Pick and Place System

Michael Laks
mjlaks@wpi.edu

Teresa Saddler
tsaddler@wpi.edu

Marshall Trier
mtrier@wpi.edu

Benjamin Ward
blward@wpi.edu

***Abstract*—In this lab, the robotic arm was used to implement a pick and place machine which uses the webcam to identify object locations and then moves each object based on its color and size. To accomplish this objective, object localization using the webcam was implemented and trajectory generation was used to control the arm's location.**

I. Introduction

Through the development of computer vision integration, the RBE 3001 arm system can be applied as a robotic pick and place system. Given localization and identification from vision software, the arm end effector can be maneuvered in such a way that it can be used to pick, grasp, and place. Extension of this functionality includes sorting functionality - placing a variety of different objects in appropriate locations.

II. Methodology

A. System Design

The RBE 3001 arm system is a 3 degree-of-freedom system integrating three revolute joints powered by low-power servo motors in order to control end effector position. Each joint includes absolute encoders which can be used to determine absolute rotation of each joint.

High-level control and visualization software, which operates on an x86-64 desktop computer (Ubuntu 16.04), is paired with low-level STM32 (ARM) firmware which implements fast interrupt-driven control loops. These systems communicate using a custom 512-bit packet protocol over a standard USB HID interface.

1) High-level Control Software: The high-level control software, written in Matlab, performs the following general functions: sending and receiving status and command packets, forward, differential, and inverse kinematics, trajectory generation, manipulability analysis and singularity detection, logging, and physical visualization.

One possible program structure is detailed in Figure 1.

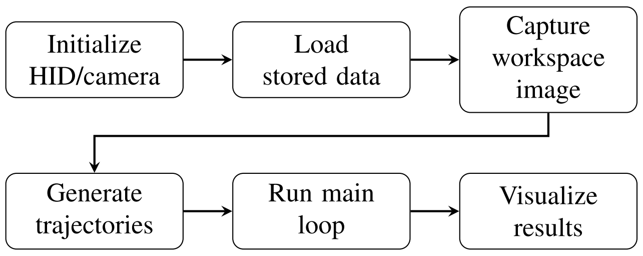

Fig. 1. One option for high-level program structure

The main loop of the high-level control and visualization software, as per Figure 2, is straightforward. After receiving a status packet, forward kinematics and differential kinematics are applied to determine current position and velocity in task space. The manipulability ellipsoid volume is calculated and thresholded to determine whether the system is nearing a singularity. A stick model of the arm system is displayed with visualizations of position, joint axes, instantaneous velocity, manipulability, etc. Finally, the setpoint is advanced if required by the trajectory in order to send the next point in the path being followed. Data is logged for later visualization and analysis.

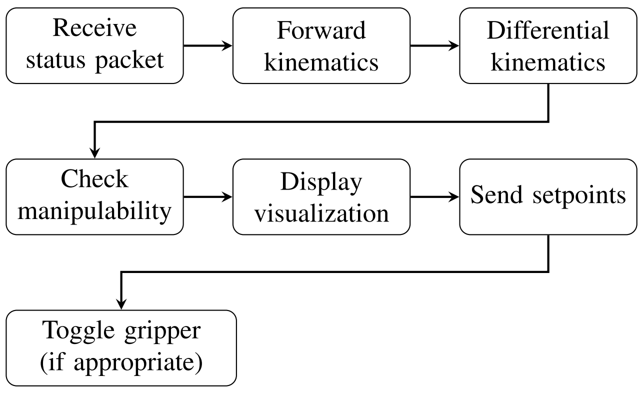

Fig. 2. Main loop event flow

A more sophisticated system, one which could respond to changing object positions in real time, might capture workspace images as part of the main program loop, and recalculate trajectories as needed.

2) Low-level Real-time Software: Low-level STM32 firmware, written in C/C++, performs real-time tasks including maintaining servo loops, reading positioning information from encoders, and communication with high-level control and visualization code. Abstract program flow is detailed in Figure 3.

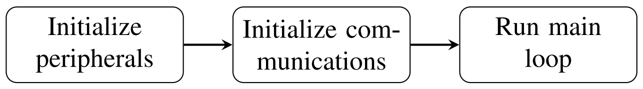

Fig. 3. Main loop event flow

The real-time main loop performs only a few functions, since most hardware communication is allocated to on-chip peripherals. Through the communications interface, packets are sent and recieved for: status, PID setpoints and coefficients, system calibration actions, and end effector gripper servo state.

TODO: Large flowchart detailing all inter-system communications

B. Lab Procedure

In order to implement the object localization, we began by setting up the webcam attached to the robotic arm. After installing Matlab's package for USB webcams, we were able to ensure that the webcam was available using the `webcamlist` command. Once this was confirmed, we were able to create a webcam variable using the `webcam()` command and view the video feed using the `preview()` command.

We then calibrated the camera using the Matlab Camera Calibrator app. In this app, we took a series of images of a 3D printed object with a checkerboard containing 12mm x 12mm black and white squares in different positions and orientations throughout the workspace of the robot. The program then automatically establishes a consistent reference frame convention on the checkerboards, while discarding images it is unable to process. We then ensured that the reference frames were consistent between images, and discarded any in which they weren't, and then proceeded to allow the program to calibrate the camera, setting a threshold of the reprojection error such that we still had a sufficient amount of images but also a very low error. Finally, using this calibration, we were able to export the Camera Parameters and save the variable to be loaded whenever the camera is used.

Next, we performed a camera-robot registration, in order to enable objects within the view of the camera to be localized in the reference frame of the robot. To do this, we placed the checkerboard in the bottom-right of the robotic arm's workspace (positive x and positive y in the reference frame of the robot), and ran the `getCamToCheckerboard` function which returns the transformation matrix between the reference frames of the camera and the checkerboard, $T_{cam}^{checker}$ where the reference frame of the checkerboard. We also calculated the transformation matrix between the reference frames of the base of the robot and the checkerboard, $T_{robot}^{checker}$ manually based on the linear and rotational change in the reference frames. We were then able to multiply T_{cam}^{check} by the inverse of $T_{robot}^{checker}$ in order to calculate the transformation matrix between the camera and the base of the robot, T_{cam}^{robot} in order to convert coordinates of objects in the camera view from camera-space to task-space. We validated the calculated transformation matrices by taking images using the webcam and identifying points in the image with known task-space coordinates and using the transformation matrices to convert the camera-space coordinates to task-space coordinates and checking for error between the known and calculated task-space coordinates.

The next step was to enable identifying of actual objects in the task space automatically and show visually on the image the location of the object using colored circles corresponding with the object's color. To do this, we had to create masks to find objects of different colors, specifically the blue, green and yellow of the balls and the black of the bases. For the blue, green and yellow of the balls, we created individual masks using Matlab's Color Thresholder app, which removed everything from the image other than the ball of the correct color. Each of these masks were then exported to a Matlab function which when given an image, output a binary mask of the image showing only the objects of the correct color. For the black mask, we used Matlab's Image Segmenter to isolate the black bases of the balls, and then create another function that when given an image, outputs a binary mask of the image showing only the bases. Using these masks, we then used the `regionProps` function to find the centroids and radii of each of these objects in camera-space. Using the centroids and radii, we were then able to plot circles on the image showing the locations of the balls and the bases, using the appropriate colors.

Once objects could be localized, we were then able to implement actual retrieval of objects in the workspace. First, the gripper needed to be made functional. Code was then created to allow the gripper to be commanded to use. A new server was created in Matlab and the C++ software to control the gripper, so that it could open and close the gripper using a command in Matlab. Next, the object localization was applied in the partially-implemented `findObjs.m` script we were provided with, and the coordinates of the centroids of the objects were converted into task-space using T_{cam}^{robot}. Using these object locations, we were then able to apply the trajectory-planning principles used in previous labs to create a trajectory between the arm's current location and the location of the object. This object is then be grasped by the gripper using the `set_gripper` command, and then based on the color of the object, the object is moved to a new location. This script is designed to continually move objects until there are none left in the workspace.

C. Calculations

1) Forward Kinematics: The forward kinematics can be calculated using the DH parameters shown in Table I, which were determined by the reference frames established in Figure 4.

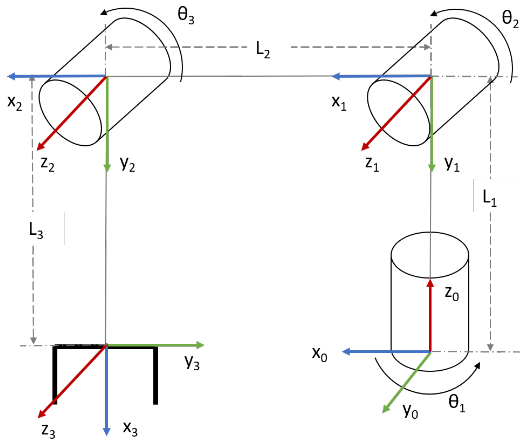

Fig. 4. Reference Frames of Joints in Home Position

TABLE I
DH PARAMETERS OF ROBOTIC ARM

θ	d	a	α
θ_1	L_1	0	$-\frac{\pi}{2}$
θ_2	0	L_2	0
$\theta_3+\frac{\pi}{2}$	0	L_3	0

Using these DH parameters, the transformation matrices between the different reference frames can be calculated as follows:

$$T_0^1=\begin{bmatrix} c_1 & -s_1c_{-\frac{\pi}{2}} & s_1s_{-\frac{\pi}{2}} & 0c_1 \\ s_1 & c_1c_{-\frac{\pi}{2}} & -c_1s_{-\frac{\pi}{2}} & 0s_1 \\ 0 & s_{-\frac{\pi}{2}} & c_{-\frac{\pi}{2}} & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0^1=\begin{bmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$T_1^2=\begin{bmatrix} c_2 & -s_2c_0 & s_2s_0 & L_2c_2 \\ s_2 & c_2c_0 & -c_2s_0 & L_2s_2 \\ 0 & s_0 & c_0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1^2=\begin{bmatrix} c_2 & -s_2 & 0 & L_2c_2 \\ s_2 & c_2 & 0 & L_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$T_2^3=\begin{bmatrix} c_{3\frac{\pi}{2}} & -s_{3\frac{\pi}{2}}c_0 & s_{3\frac{\pi}{2}}s_0 & L_3c_{3\frac{\pi}{2}} \\ s_{3\frac{\pi}{2}} & c_{3\frac{\pi}{2}}c_0 & -c_{3\frac{\pi}{2}}s_0 & L_3s_{3\frac{\pi}{2}} \\ 0 & s_0 & c_0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^3=\begin{bmatrix} -s_3 & -c_3 & 0 & -L_3s_3 \\ c_3 & -s_3 & 0 & L_3c_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The transformation matrix between the base and the effector of the robotic arm can be calculated by multiplying the individual transformation matrices of the joints in Equations 1, 2 and 3.

$$T_0^3=T_0^1T_1^2T_2^3=T_0^2T_2^3$$

$$\begin{aligned} T_0^2 &= T_0^1T_1^2 \\ &= \begin{bmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_2 & -s_2 & 0 & L_2c_2 \\ s_2 & c_2 & 0 & L_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c_1c_2 & -c_1s_2 & -s_1 & L_2c_1c_2 \\ s_1c_2 & -s_1s_2 & c_1 & L_2s_1c_2 \\ -s_2 & -c_2 & 0 & -L_2s_2+L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{4}$$

$$\begin{aligned} T_0^3 &= T_0^2T_2^3 \\ &= \begin{bmatrix} c_1c_2 & -c_1s_2 & -s_1 & L_2c_1c_2 \\ s_1c_2 & -s_1s_2 & c_1 & L_2s_1c_2 \\ -s_2 & -c_2 & 0 & -L_2s_2+L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -s_3 & -c_3 & 0 & -L_3s_3 \\ c_3 & -s_3 & 0 & L_3c_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -c_1c_2s_3-c_1s_2c_3 & -c_1c_2c_3+c_1s_2s_3 & -s_1 & -L_3c_1c_2s_3-L_3c_1s_2c_3+L_2c_1c_2 \\ -s_1c_2s_3-s_1s_2c_3 & -s_1c_2c_3+s_1s_2s_3 & c_1 & -L_3s_1c_2s_3-L_3s_1s_2c_3+L_2s_1c_2 \\ s_2s_3-c_2c_3 & s_2c_3+c_2s_3 & 0 & L_3s_2s_3-L_3c_2c_3-L_2s_2+L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{5}$$

Using Equation 5, the position of the effector in task-space can be calculated using the first three entries in the last column, and plugging in the joint angles of each of the three joints on the arm. The result of this operation will be a column vector with the x- y- and z- coordinates of the effector in task-space.

2) Inverse Kinematics: The inverse kinematics of the base joint can be solved for geometrically using the 3D representation of the arm in joint space shown in Figure 5.

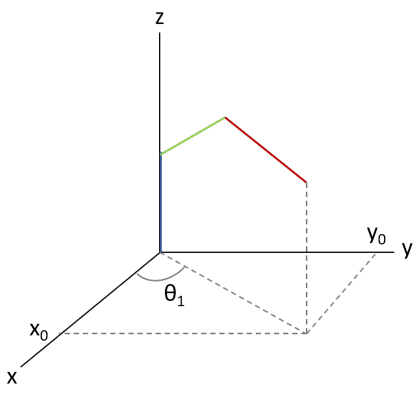

Fig. 5. Geometric Inverse Kinematics Base Joint Angle

The relationship between the angle of joint 1 and the position of the end effector in task space can be represented by:

$$\theta_1 = atan2(y_0, x_0) \tag{6}$$

The inverse kinematics of joints 2 and 3 can be solved geometrically using a 2D representation of the arm in joint space, using a reference frame placed on the second joint. In this case, two cases should be considered: both when the end effector is below the second joint, and when it is above the second joint, as shown in Figure 6.

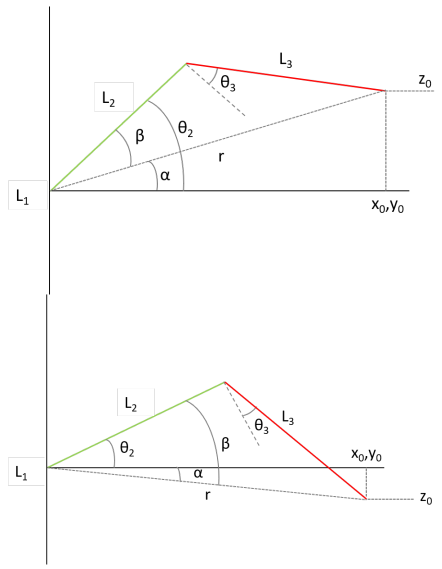

Fig. 6. Geometric Inverse Kinematics Joints 2 and 3

Using these graphical representation, θ_2 can be taken as:

$$\theta_2 = -(\alpha + \beta)$$

where:

$$\alpha = atan2\left((z_0 - L_1), \sqrt{x_0^2 + y_0^2}\right)$$

$$r^2 = x_0^2 + (z_0 - L_1)^2 + y_0^2$$

$$\cos\beta = \frac{r^2 + L_2^2 - L_3^2}{2rL_2}$$

$$\beta = \arccos\left(\frac{r^2 + L_2^2 - L_3^2}{2rL_2}\right)$$

Thus, θ_2 is defined as:

$$\theta_2 = -\left(atan2\left((z_0 - L_1), \sqrt{x_0^2 + y_0^2}\right) + \arccos\frac{r^2 + L_2^2 - L_3^2}{2rL_2}\right) \tag{7}$$

θ_3 can be found in a similar manner.

$$\cos\left(-\theta_3 + \frac{\pi}{2}\right) = \frac{L_2^2 + L_3^2 - r^2}{2L_2L_3}$$

$$\theta_3 = -\arccos\left(\frac{L_2^2 + L_3^2 - r^2}{2L_2L_3}\right) + \frac{\pi}{2} \tag{8}$$

In conjunction with the task-space coordinates of the end effector of the robotic arm, Equations 6, 7, and 8 can then be used to find the corresponding joint-space coordinates.

3) Jacobian: In general, the Jacobian is defined as:

$$J(q) = \begin{bmatrix} J_p \\ J_o \end{bmatrix}$$

Here, J_p is:

$$J_p = \begin{bmatrix} \frac{\delta P_e}{\delta\theta_1} & \frac{\delta P_e}{\delta\theta_2} & \frac{\delta P_e}{\delta\theta_3} \end{bmatrix},$$
$$where P_e = T_0^3\,(1:3,\ end)\, and\ T_0^3 = T_0^1 T_1^2 T_2^3$$

Using the equation of the transformation matrix between the base joint and the end effector, Equation 5, the matrix P_e can be calculated as follows:

$$P_e = T_0^3(1:3,\ end) = \begin{bmatrix} -L_3c_1c_2s_3 - L_3c_1s_2c_3 + L_2c_1c_2 \\ -L_3s_1c_2s_3 - L_3s_1s_2c_3 + L_2s_1c_2 \\ L_3s_2s_3 - L_3c_2c_3 - L_2s_2 + L_1 \end{bmatrix}$$

The partial derivatives of P_e can then be calculated, with respect to the generalized joint variables, θ_1, θ_2, and θ_3.

$$\frac{\delta P_e}{\delta\theta_1} = \begin{bmatrix} L_3s_1c_2s_3 + L_3s_1s_2c_3 - L_2s_1c_2 \\ -L_3c_1c_2s_3 - L_3c_1s_2c_3 + L_2c_1c_2 \\ 0 \end{bmatrix}$$

$$\frac{\delta P_e}{\delta\theta_2} = \begin{bmatrix} L_3c_1s_2s_3 - L_3c_1c_2c_3 - L_2c_1s_2 \\ L_3s_1s_2s_3 - L_3s_1c_2c_3 - L_2s_1s_2 \\ L_3c_2s_3 + L_3s_2c_3 - L_2c_2 \end{bmatrix}$$

$$\frac{\delta P_e}{\delta\theta_3} = \begin{bmatrix} -L_3c_1c_2c_3 + L_3c_1s_2s_3 \\ -L_3s_1c_2c_3 + L_3s_1s_2s_3 \\ L_3s_2c_3 + L_3c_2s_3 \end{bmatrix}$$

Using the partial derivatives, the position portion of the Jacobian matrix, J_p is then found to be:

$$J_p = \begin{bmatrix} L_3s_1c_2s_3 + L_3s_1s_2c_3 - L_2s_1c_2 & L_3c_1s_2s_3 - L_3c_1c_2c_3 - L_2c_1s_2 & -L_3c_1c_2c_3 + L_3c_1s_2s_3 \\ -L_3c_1c_2s_3 - L_3c_1s_2c_3 + L_2c_1c_2 & L_3s_1s_2s_3 - L_3s_1c_2c_3 - L_2s_1s_2 & -L_3s_1c_2c_3 + L_3s_1s_2s_3 \\ 0 & L_3c_2s_3 + L_3s_2c_3 - L_2c_2 & L_3s_2c_3 + L_3c_2s_3 \end{bmatrix}$$

Then, the orientation-portion of the Jacobian matrix, which is defined as:

$$J_o = \begin{bmatrix} T_0^0(1:3,\ end-1) & T_0^1(1:3,\ end-1) & T_0^2(1:3,\ end-1) \end{bmatrix}$$

is found to be:

$$J_o = \begin{bmatrix} 0 & -s_1 & -s_1 \\ 0 & c_1 & c_1 \\ 1 & 0 & 0 \end{bmatrix}$$

Using J_o and J_p, the full Jacobian matrix is then found to be:

$$J(q) = \begin{bmatrix} L_3s_1c_2s_3 + L_3s_1s_2c_3 - L_2s_1c_2 & L_3c_1s_2s_3 - L_3c_1c_2c_3 - L_2c_1s_2 & -L_3c_1c_2c_3 + L_3c_1s_2s_3 \\ -L_3c_1c_2s_3 - L_3c_1s_2c_3 + L_2c_1c_2 & L_3s_1s_2s_3 - L_3s_1c_2c_3 - L_2s_1s_2 & -L_3s_1c_2c_3 + L_3s_1s_2s_3 \\ 0 & L_3c_2s_3 + L_3s_2c_3 - L_2c_2 & L_3s_2c_3 + L_3c_2s_3 \\ 0 & -s_1 & -s_1 \\ 0 & c_1 & c_1 \\ 1 & 0 & 0 \end{bmatrix} \tag{9}$$

4) Force/Torque Relationship: Using the Jacobian, a simplistic relation between generalized joint forces and task-space forces can be obtained according to the relationship defined in Equation 10

TODO: Graphic showing orientation of forces

$$\begin{aligned} \tau &= J(q)_p^T F_{tip} \\ &= \begin{bmatrix} L_3s_1c_2s_3 + L_3s_1s_2c_3 - L_2s_1c_2 & L_3c_1s_2s_3 - L_3c_1c_2c_3 - L_2c_1s_2 & -L_3c_1c_2c_3 + L_3c_1s_2s_3 \\ -L_3c_1c_2s_3 - L_3c_1s_2c_3 + L_2c_1c_2 & L_3s_1s_2s_3 - L_3s_1c_2c_3 - L_2s_1s_2 & -L_3s_1c_2c_3 + L_3s_1s_2s_3 \\ 0 & L_3c_2s_3 + L_3s_2c_3 - L_2c_2 & L_3s_2c_3 + L_3c_2s_3 \end{bmatrix} \begin{bmatrix} 0 \\ F_g \\ 0 \end{bmatrix} \end{aligned} \tag{10}$$

Where F_g is defined as the force of gravity applied at the tip

III. Results

IV. Discussion

V. Conclusion

VI. Authorship

Section	Authors	Notes
Abstract	Teresa Saddler	
Introduction	Benjamin Ward	
Methodology	Teresa Saddler Benjamin Ward	Teresa: calculations, lab procedure Benjamin: flowcharts, system design, calculations
Results	N/A	N/A
Discussion	N/A	N/A
Conclusions	N/A	N/A