Design Specification

as a partial requirement for

the course on

Unified Robotics III: Manipulation

Final Project Design

Submitted By:

Zoraver Kang

Clayton Dembski

Cory Brolliar

Submitted To:

Professor Gregory Fischer

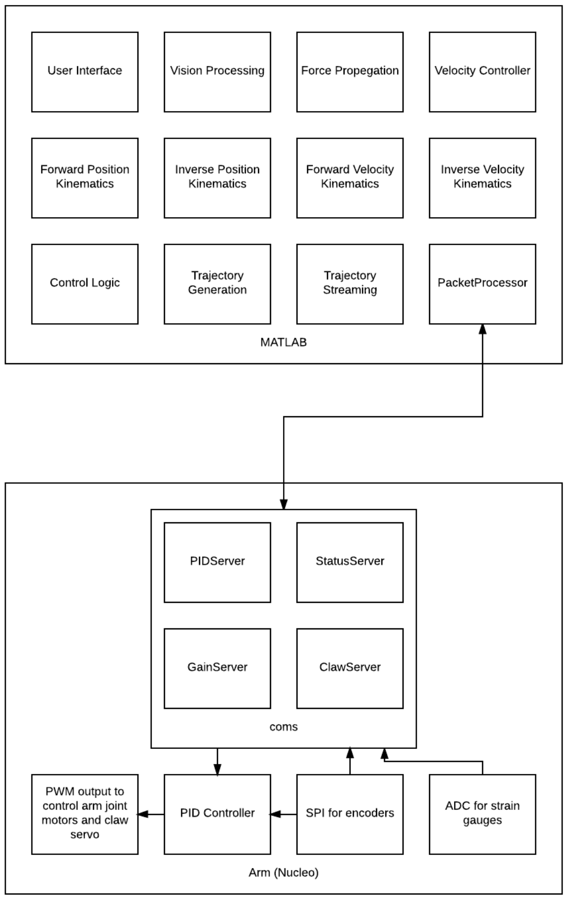
1 Introduction

This document describes our approach to the RBE3001 final project at a high level. It includes a system block diagram showing the delegation of tasks between MATLAB and the arm's embedded controller, flow charts of the software running in MATLAB and on the arm's embedded controller, and in depth documentation of the communication protocol used to facilitate communication between MATLAB and the arm. In addition, it documents the forward and inverse position and velocity kinematics of the system, as well as how a force applied to the tip of the arm can be determined by measuring the torque at each joint.

2 System Block Diagram

The overall block diagram for our system is shown in Figure 1 below.

Figure 1: System Block Diagram



3 Position Kinematics Using Transformations

```
function [ x, y, z ] = ForwardPositionKinematics(q0, q1, q2)


theta1 = 0;

theta2 = pi/4;

theta3 = 0;

a0 = 8;

a1 = 6.5354;

a2 = 7.874;


A = [cos(theta1), 0, sin(theta1),  0;

     sin(theta1), 0, cos(theta1),  0;

     0,          1,            0,  a0;

     0,          0,            0,  1];


B = [cos(theta2), -sin(theta2), 0, a1*cos(theta2);

     sin(theta2), cos(theta2), 0, a1*sin(theta2);

     0,           0,           1,              0;

     0,           0,           0,              1];


C = [cos(-(pi/2) - theta3), -sin(-(pi/2) - theta3), 0, a2*cos(-(pi/2) - theta3);

     sin(-(pi/2) - theta3), cos(-(pi/2) - theta3), 0, a2*sin(-(pi/2) - theta3);
     0          ,          0,          1,                  0;
     0          ,          0,          0,                  1];

transform = A*B*C;


x = transform(1,4);
y = transform(2,4);
z = transform(3,4);


end
```

4 Velocity Kinematics

4.1 Forward Velocity

```
function V = forwardvk( q0, q1, q2, q0_dot, q1_dot, q2_dot )

L1 = 8;

L2 = 6.5;

L3 = 8.125;


A = [-L2 * sin(q0) * cos(q1) - L3 * sin(q0) * sin(q1 - q2), -L2 *
     cos(q0) * sin(q1) + L3 * cos(q0) * cos(q1 - q2), -L3 *
     cos(q0) * cos(q1 - q2);
    L2 * cos(q0) * cos(q1) + L3 * cos(q0) * sin(q1 - q2), -L2 *
    sin(q0) * sin(q1) + L3 * sin(q0) * cos(q1 - q2), -L3 * sin(q0) *
    cos(q1 - q2);
    0, L2 * cos(q1) + L3 * sin(q1 - q2), -L3 * sin(q1 - q2)];


q_dot = [q0_dot;
        q1_dot;
        q2_dot];


V = A * q_dot;

end
```

4.2 Inverse Velocity

```
function q_dot = inversevk( x_dot, q0, q1, q2 )

L1 = 8;

L2 = 6.5;

L3 = 8.125;


A = [-L2 * sin(q0) * cos(q1) - L3 * sin(q0) * sin(q1 - q2), -L2 *
     cos(q0) * sin(q1) + L3 * cos(q0) * cos(q1 - q2), -L3 * cos(q0) *
     cos(q1 - q2);
    L2 * cos(q0) * cos(q1) + L3 * cos(q0) * sin(q1 - q2), -L2 *
    sin(q0) * sin(q1) + L3 * sin(q0) * cos(q1 - q2), -L3 * sin(q0) *
    cos(q1 - q2);
    0, L2 * cos(q1) + L3 * sin(q1 - q2), -L3 * sin(q1 - q2)];


q_dot = inv(A) * x_dot;

end
```

5 Inverse Position Kinematics

The inverse position kinematics of arm were determined through the use of geometry and trigonometry. By projecting the tip of the arm onto the xy plane, we easily found $q_0 = atan(p_y/p_x)$. In Matlab, we use the `atan2()` function to avoid the edge cases of the `atan()` function, so the above expression for q_0 becomes `q0 = atan2(x, y)`.

To determine the values of q_1 and q_2, we examined the arm in the plane shared by all three links, as shown in Figure 2.

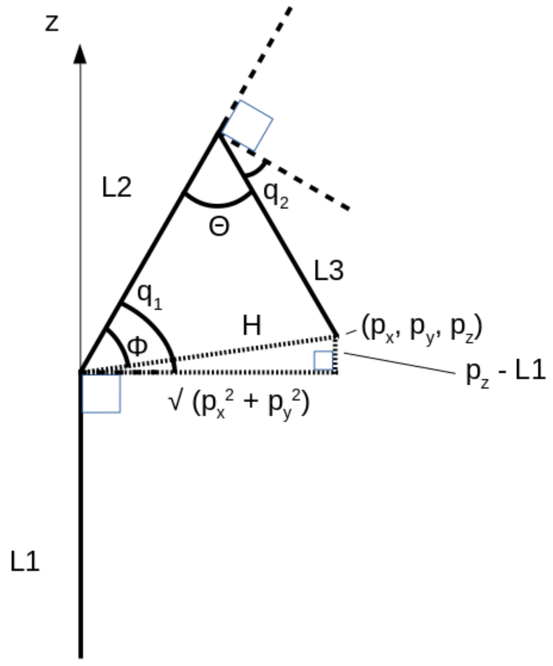


Figure 2: Arm profile in shared plane

The value of H can be easily determined using the Pythagorean theorem.

$$H = \sqrt{p_x^2 + p_y^2 + (p_z - L_1)^2}$$

Now that the value of H is known, Θ can be determined using the law of cosines.

$$H^2 = L_2^2 + L_3^2 - 2L_2L_3cos(\Theta)$$

$$2L_2L_3cos(\Theta) = L_2^2 + L_3^2 - H^2$$

$$cos(\Theta) = \frac{L_2^2 + L_3^2 - H^2}{2L_2L_3}$$

$$\Theta = acos\left(\frac{L_2^2 + L_3^2 - H^2}{2L_2L_3}\right)$$

Using the value of Θ, q_2 can be determined.

$$q_2 = \pi - \frac{\pi}{2} - \Theta = \frac{\pi}{2} - \Theta$$

The values of H and Θ can be used in conjunction with the law of sines to determine the value of Φ.

$$\frac{sin(\Phi)}{L_3} = \frac{sin(\Theta)}{H}$$

$$\Phi = asin\left(\frac{L_3}{H}sin(\Theta)\right)$$

The value of q_1 can be determined through the use of geometry and the value of Φ.

$$q_1 = \Phi + atan\left(\frac{p_z - L_1}{\sqrt{p_x^2 + p_y^2}}\right) = asin\left(\frac{L_3}{H} sin(\Theta)\right) + atan\left(\frac{p_z - L_1}{\sqrt{p_x^2 + p_y^2}}\right)$$

$$= asin\left(\frac{L_3}{\sqrt{(p_x^2 + p_y^2 + (p_z - L_1)^2)}} sin\left(acos\left(\frac{L_2^2 + L_3^2 - (p_x^2 + p_y^2 + (p_z - L_1)^2)}{2L_2L_3}\right)\right)\right)$$

$$+atan\left(\frac{p_z - L_1}{\sqrt{p_x^2 + p_y^2}}\right)$$

In summary, the inverse position kinematics of our 3-DOF arm are:

$$q_0 = atan\left(\frac{p_y}{p_x}\right)$$

$$q_1 = asin\left(\frac{L_3}{\sqrt{(p_x^2 + p_y^2 + (p_z - L_1)^2)}} sin\left(acos\left(\frac{L_2^2 + L_3^2 - (p_x^2 + p_y^2 + (p_z - L_1)^2)}{2L_2L_3}\right)\right)\right)$$

$$+atan\left(\frac{p_z - L_1}{\sqrt{p_x^2 + p_y^2}}\right)$$

$$q_2 = \frac{\pi}{2} - acos\left(\frac{L_2^2 + L_3^2 - (p_x^2 + p_y^2 + (p_z - L_1)^2)}{2L_2L_3}\right)$$

6 Force Propagation

We approached force propagation in two different ways: as a function of the inverse transpose of the 3 by 3 truncated Jacobian matrix, and as a function of a static free body diagram.

The jacobian is identical to matrix A in the Forward Velocity section. The torque at each joint of the arm is found as follows:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} J(3x6) \end{bmatrix}^T * \begin{bmatrix} f_x \\ f_y \\ f_z \\ n_x \\ n_y \\ n_z \end{bmatrix}$$

However, as we are looking at this force in a static system, the moments of the force are 0. Because of this, the corresponding half of the angular Jacobian is not needed, and the equation simplifies to:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} J(3x3) \end{bmatrix}^T * \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix}$$

This transpose Jacobian is invertible, as the number of rows match the number of columns. Because of this, methods of linear algebra can be used to find the force vector.

$$\left[\left[J(3x6)\right]^T\right]^{-1} * \begin{bmatrix}\tau_1\\ \tau_2\\ \tau_3\end{bmatrix} = \begin{bmatrix}f_x\\ f_y\\ f_z\end{bmatrix}$$

For a static system in a 2d plane, the diagram is as follows:

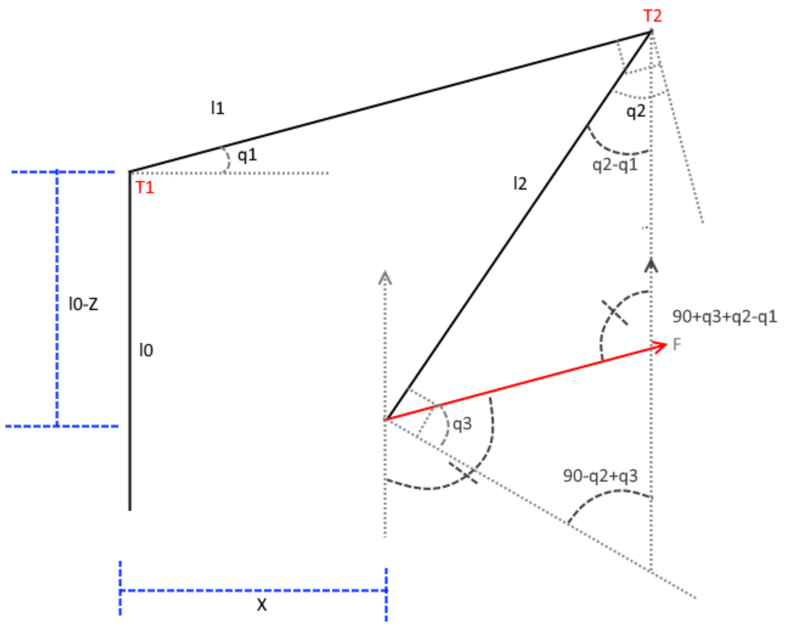


Figure 3: Arm profile with Force Vector

Two torque equations can be made, one for τ_1 and one for τ_2:

$$\tau_2 = f cos(\theta_3) * l_2$$

$$\tau 1 = (l_0 - z) f sin(90 + \theta_3 + \theta_2 - \theta_1) + x f cos(90 + \theta_3 + \theta_2 - \theta_1)$$

and the two unknowns can be solved. As θ_3 was embedded deep within sin and cos in τ_1, the mathematics to solve for it would be nontrivial. Due to this, the answer was solved through Matlab. The symbolic `solve()` function, and symbolic `simplify` function was used to solve for the magnitude and angle, displayed in corresponding order as the first, and second answer.

```
syms f theta3 l2 t1 l0 z theta2 theta1 x t2

eqns = [f*cos(theta3)*l2 == t1,(l0-z)*f*sin(90+theta3+theta2-theta1)+x*f*cos(90+theta3+theta2
-theta1)==t2];
s = solve(eqns,[f,theta3])
simplify(s.f)
simplify(s.theta3)
```

```
s = 

  struct with fields:

         f: [2×1 sym]
    theta3: [2×1 sym]


ans =
 
 ((cos(theta2 - theta1 + 90) + 1)*((l0^2*t1^2 - 2*sin(theta2 - theta1 + 90)*l0*l2*t1*t2 - 2*l
0*t1^2*z + l2^2*t2^2 - 2*cos(theta2 - theta1 + 90)*l2*t1*t2*x + 2*sin(theta2 - theta1 + 90)*l
2*t1*t2*z + t1^2*x^2 + t1^2*z^2)/cos(theta2/2 - theta1/2 + 45)^4)^(1/2))/(2*l2*z*cos(theta2 -
 theta1 + 90) - 2*l0*l2*cos(theta2 - theta1 + 90) + 2*l2*x*sin(theta2 - theta1 + 90))
  -((cos(theta2 - theta1 + 90) + 1)*((l0^2*t1^2 - 2*sin(theta2 - theta1 + 90)*l0*l2*t1*t2 - 2
*l0*t1^2*z + l2^2*t2^2 - 2*cos(theta2 - theta1 + 90)*l2*t1*t2*x + 2*sin(theta2 - theta1 + 90)
*l2*t1*t2*z + t1^2*x^2 + t1^2*z^2)/cos(theta2/2 - theta1/2 + 45)^4)^(1/2))/(2*(l2*z*cos(theta
2 - theta1 + 90) - l0*l2*cos(theta2 - theta1 + 90) + l2*x*sin(theta2 - theta1 + 90)))
 
 
ans =
 
 -2*atan((cos(theta2 - theta1 + 90)*((l0^2*t1^2 - 2*sin(theta2 - theta1 + 90)*l0*l2*t1*t2 - 2
*l0*t1^2*z + l2^2*t2^2 - 2*cos(theta2 - theta1 + 90)*l2*t1*t2*x + 2*sin(theta2 - theta1 + 90)
*l2*t1*t2*z + t1^2*x^2 + t1^2*z^2)/cos(theta2/2 - theta1/2 + 45)^4)^(1/2) + ((l0^2*t1^2 - 2*s
in(theta2 - theta1 + 90)*l0*l2*t1*t2 - 2*l0*t1^2*z + l2^2*t2^2 - 2*cos(theta2 - theta1 + 90)*
l2*t1*t2*x + 2*sin(theta2 - theta1 + 90)*l2*t1*t2*z + t1^2*x^2 + t1^2*z^2)/cos(theta2/2 - the
ta1/2 + 45)^4)^(1/2) + 2*l0*t1*cos(theta2 - theta1 + 90) - 2*t1*z*cos(theta2 - theta1 + 90) -
 2*t1*x*sin(theta2 - theta1 + 90))/(2*(l2*t2 - t1*x*cos(theta2 - theta1 + 90) - l0*t1*sin(the
ta2 - theta1 + 90) + t1*z*sin(theta2 - theta1 + 90))))
  2*atan((cos(theta2 - theta1 + 90)*((l0^2*t1^2 - 2*sin(theta2 - theta1 + 90)*l0*l2*t1*t2 - 2
*l0*t1^2*z + l2^2*t2^2 - 2*cos(theta2 - theta1 + 90)*l2*t1*t2*x + 2*sin(theta2 - theta1 + 90)
*l2*t1*t2*z + t1^2*x^2 + t1^2*z^2)/cos(theta2/2 - theta1/2 + 45)^4)^(1/2) + ((l0^2*t1^2 - 2*s
in(theta2 - theta1 + 90)*l0*l2*t1*t2 - 2*l0*t1^2*z + l2^2*t2^2 - 2*cos(theta2 - theta1 + 90)*
l2*t1*t2*x + 2*sin(theta2 - theta1 + 90)*l2*t1*t2*z + t1^2*x^2 + t1^2*z^2)/cos(theta2/2 - the
ta1/2 + 45)^4)^(1/2) - 2*l0*t1*cos(theta2 - theta1 + 90) + 2*t1*z*cos(theta2 - theta1 + 90) +
 2*t1*x*sin(theta2 - theta1 + 90))/(2*(l2*t2 - t1*x*cos(theta2 - theta1 + 90) - l0*t1*sin(the
ta2 - theta1 + 90) + t1*z*sin(theta2 - theta1 + 90))))
```

7 Communication Protocol

To enable communication between our arm and a lab computer running MATLAB, we built on the framework provided to us. On the MATLAB side, we utilize the provided `PacketProcessor` class with wrapper functions to make our code more concise. On the arm side, we utilize several classes which extend `PacketEventAbstract` in a way similar to the approach taken in `PidServer`. These classes are `StatusServer`, `GainServer`, and `ClawServer`.

`PidServer` handles the `SET_PID` command (ID = 37). This command sets the setpoints of the PID controllers of the arm based off of the given arguments. Its arguments are:

1. Position setpoint for axis 0 (base axis) in raw encoder ticks
2. Velocity target for axis 0 (base axis) in encoder ticks per second
3. Force target for axis 0
4. Position setpoint for axis 1 (shoulder) in raw encoder ticks
5. Velocity target for axis 1 (shoulder) in encoder ticks per second
6. Force target for axis 1
7. Position setpoint for axis 2 (elbow) in raw encoder ticks
8. Velocity target for axis 2 (elbow) in encoder ticks per second
9. Force target for axis 2

The velocity and force target systems are not currently implemented. The `SET_PID` command returns the the following values: (position of encoder on axis 0, 0 (reserved), 0 (reserved), position of encoder on axis 1, 0 (reserved), 0 (reserved), position of encoder on axis 2, 0 (reserved), 0 (reserved)).

`StatusServer` handles the `STATUS` command (ID = 38). This command takes no arguments and returns the current values of the internal sensors of the arm is the following order:

1. position of encoder 0 (base axis) in raw encoder ticks
2. position of encoder 1 (shoulder) in raw encoder ticks
3. position of encoder 2 (elbow) in raw encoder ticks
4. velocity measured by encoder 0 in encoder ticks per second
5. velocity measured by encoder 1 in encoder ticks per second
6. velocity measured by encoder 2 in encoder ticks per second
7. base axis strain gauge ADC reading, averaged over 5 samples
8. shoulder strain gauge ADC reading, averaged over 5 samples
9. elbow strain gauge ADC reading, averaged over 5 samples

`GainServer` handles the `SET_GAIN` command (ID = 39). This command allows for the PID coefficients of any of the PID controllers on the device to have their gains set via MATLAB. The arguments of this command are (zero-indexed axis number corresponding to the PID controller you want to configure, new k_p, new k_i, new k_d). Before changing the gains of the specified PID controller, the `GainServer` disables that specific controller. Once the gains are updated, it is re-enabled. The `SET_GAIN` command does not provide any return values.

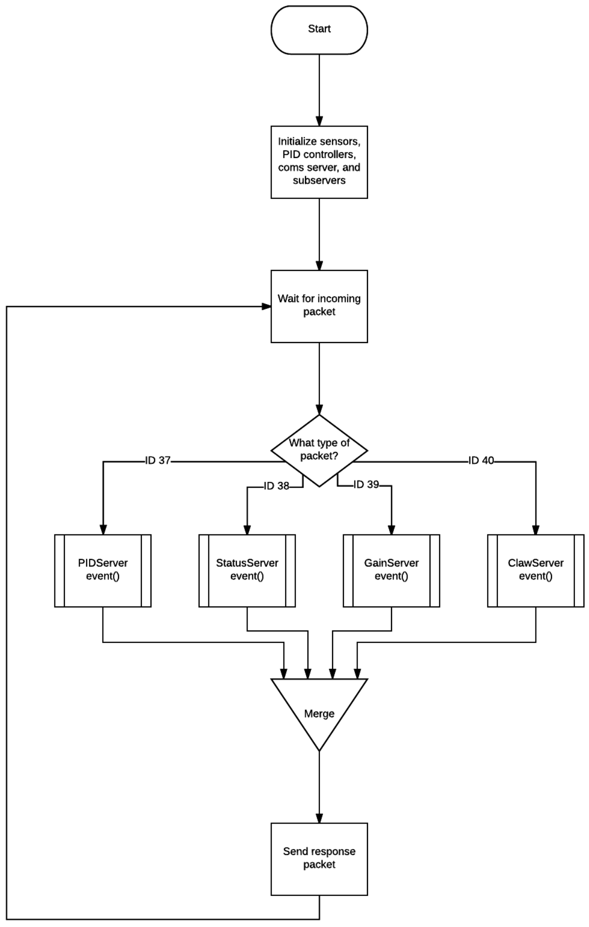
`ClawServer` handles the `SET_CLAW` command (ID = 40). This command allows for the claw at the tip of the arm to be controlled via MATLAB. It takes a single argument: a float between 0.0 and 1.0 which corresponds to the position of the servo within its overall range of motion. The `SET_CLAW` command does not provide any return values.

8 Flowcharts

8.1 Arm

The high-level flow chart for the software running on the arm is shown in Figure 4. The functionality encapsulated by the four *Server blocks is described in section 6 above.

Figure 4: High-Level Flow Chart For Arm Side



8.2 MATLAB

The high-level flow chart for our MATLAB code is shown in Figure 5 below. Two additional flow charts are provided to flesh out certain portions of the software's functionality. The first, shown in Figure 6, details the logic contained in the `nextColor()` function, which determines which color object the arm will scan for next and also controls when the program terminates. The second, shown in Figure 7, details the logic contained in the `detectColor()` function, which determines whether any sufficiently large objects of the given color are in the relevant field of view of the camera and reports the position of the largest if there are any present.

Figure 5: High-Level Flow Chart For MATLAB Side

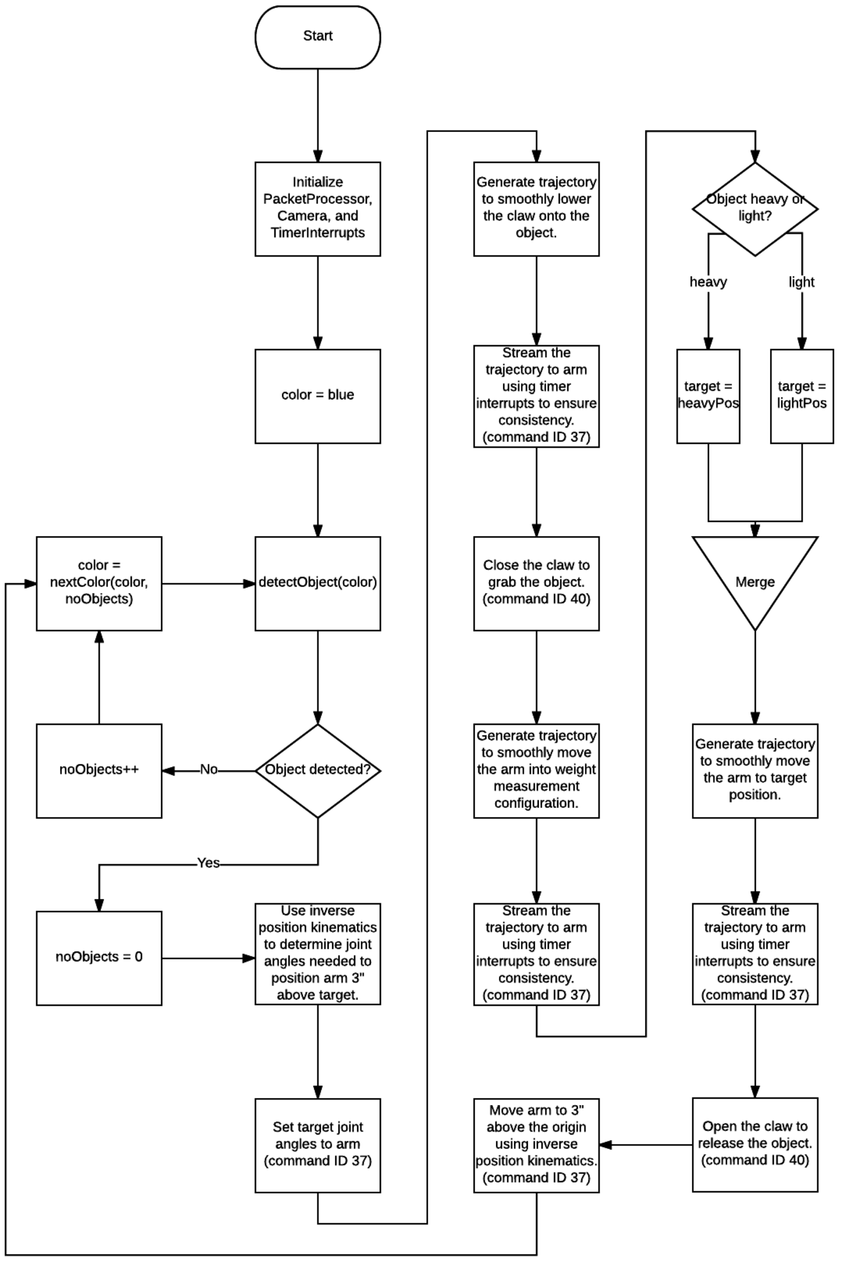

Start
Initialize PacketProcessor, Camera, and TimerInterrupts
color = blue
detectObject(color)
Object detected?
No
noObjects++
color = nextColor(color, noObjects)
Yes
noObjects = 0
Use inverse position kinematics to determine joint angles needed to position arm 3" above target.
Set target joint angles to arm (command ID 37)
Generate trajectory to smoothly lower the claw onto the object.
Stream the trajectory to arm using timer interrupts to ensure consistency. (command ID 37)
Close the claw to grab the object. (command ID 40)
Generate trajectory to smoothly move the arm into weight measurement configuration.
Stream the trajectory to arm using timer interrupts to ensure consistency. (command ID 37)
Object heavy or light?
heavy
light
target = heavyPos
target = lightPos
Merge
Generate trajectory to smoothly move the arm to target position.
Stream the trajectory to arm using timer interrupts to ensure consistency. (command ID 37)
Open the claw to release the object. (command ID 40)
Move arm to 3" above the origin using inverse position kinematics. (command ID 37)

Figure 6: High-Level Flow Chart For nextColor()

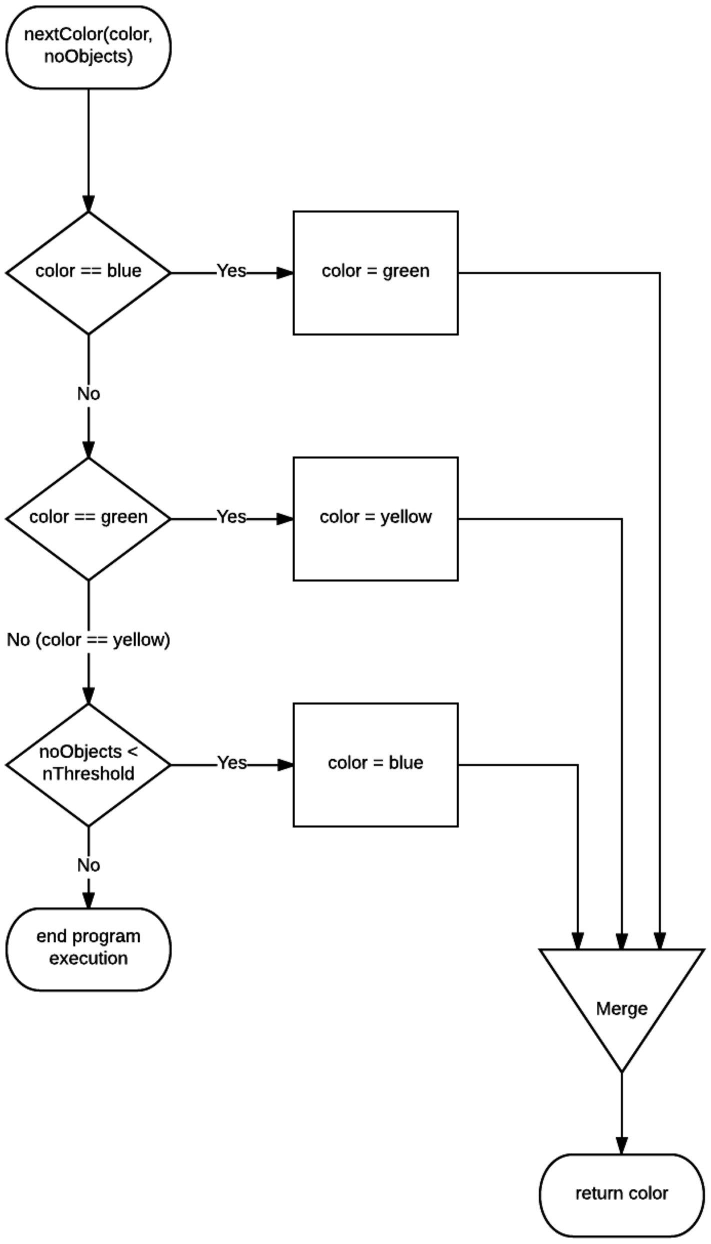
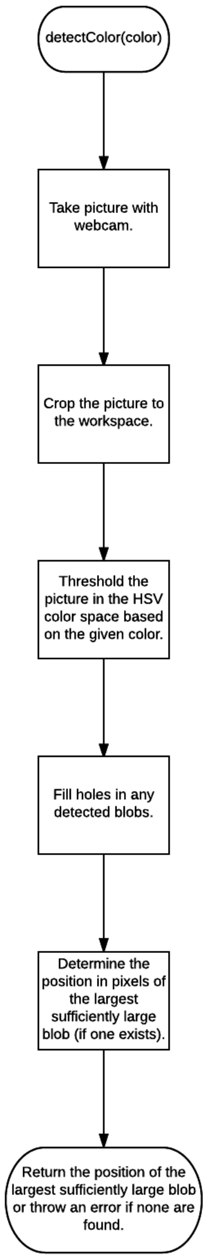


Figure 7: High-Level Flow Chart For detectColor()


detectColor(color)
Take picture with webcam.
Crop the picture to the workspace.
Threshold the picture in the HSV color space based on the given color.
Fill holes in any detected blobs.
Determine the position in pixels of the largest sufficiently large blob (if one exists).
Return the position of the largest sufficiently large blob or throw an error if none are found.