Assignment 5
Manipulation

Luke Doman
Team 10

Introduction

The goal of this assignment was to take all the skills we have developed from the previous assignments and put them to use in solving a more complex problem, picking up a can with a two joint robotic manipulator. This is the first time that we diverge from using a mobile platform for our robot's hardware design. We know some information that will assist us with this task, such as that the soda cans will be placed in line with our robot's manipulator, so we only need to focus on controlling our two joints in two dimensional space.

Mechanical Design

When initially approaching this problem we first decided to determine which gear types we would like to use for each joint before building anything. The options for gear types being the traditional types of gears and worm gears. Each approach has its own pros and cons. The traditional gears would certainly be easier to setup, and can be chained to create a very powerful gear train without much difficulty. On the down side, these gears require a constant motor force to stay in a position after it has been moved. For example, if we were to raise the shoulder joint to forty five degrees and cut motor power the shoulder would fall flat on the table. The worm gear is a very simple solution to this particular issue.

The worm gear consists of a screw gear and a modified traditional gear with ridges that are angled to mesh with the screw. The screw gear is driven which in turn causes the traditional gear to move. Due to this design, the traditional gear cannot cause the screw gear to rotate. This is advantageous in the aforementioned scenario because we can cut motor power at any time and the shoulder joint will remain stationary. While that scenario alone was compelling enough to make our team consider using the worm gear, there was another less obvious reason as well.

Our robot needed to pick up a can and drop it behind itself. This meant that the shoulder joint would have to extend beyond 90°. If we were to use the traditional gear train, after we cross the 90° mark we would need to switch the motor direction on the fly just to ensure the shoulder doesn't end up slamming into the ground. While this is certainly an addressable problem by using a potentiometer or quad encoder, our team felt that designing a code base that has to accommodate this situation could impact our progress.

For these reasons we concluded that we would use a worm gear for both the shoulder and elbow joints. However, we didn't believe a worm gear would have enough torque on its own to lift both the shoulder and elbow. What's also very ideal about the worm gear though is that the architecture is scalable to include traditional gears while still retaining the benefits of the worm gear. We expanded our gear train by putting a 12-tooth gear on the shaft that is driven by the worm gear. We then paired this 12-tooth with a 60-tooth gear. Since the gear ratio of the worm is 24:1 this gives us 120 times the initial torque of the motor.

With our gear train in place we began to design our shoulder and elbow. Our initial approach for the shoulder took two of the long right angle steel bars and screwed them together. This gave us more than sufficient length and provided ample mounting options for both our elbow and the touch sensor on the bottom that would act as a failsafe. We successfully constructed and tested our shoulder, and it was able to lift all of the large wheels which we thought would be indicative of the real weight of the elbow and claw.

Unfortunately while the wheels may have weighed similarly to the elbow and claw, they did not reflect how that weight would actually be distributed in practice. After adding our elbow, which consisted of two thin 1x25 bars attached via spacers, the leverage of that weight was too much for the gear train to handle. This made up reconsider several aspects of our mechanical architecture and we began to scrutinize each component.

The first target of our correction effort was our shoulder. While it was a sturdy shoulder, it was realistically much longer than it needed to be, which hurts us with leverage, and weighed a great deal. We redesigned it to mirror the elbow by using the 1x25 bars. This new shoulder was both lighter and shorter, but even with these improvements testing showed we were still unable to lift both the shoulder and elbow. Our next area of improvement then had to be our gear train.

We didn't want to give up the worm gear, but didn't have that much space on our platform for an expanded traditional gear train. We did identify one easy area of expansion though, and that was to add an additional motor and worm gear. By replacing the 3in shaft that our worm gear drove and had the 12-tooth gear on it with a 12in shaft, we could add an additional motor and worm gear on the other side of our platform without modifying our gear train. This doubled our input torque and once it was implemented in both the hardware and software it addressed our problem and performed without a hitch.

With our weight issues solved we began to fine tune our robot so that it could perform as expected. In order to measure the angle of our joints we added quad encoders to each of the joints driven shafts. These encoders would reset when their respective touch sensor was tripped. The shoulder's touch sensor was located on it's underside where it met the elbow joint. So intuitively this resets to zero when it touches the ground. For the elbow however, we placed the touch on its upper side. The logic for this is that if we were to calibrate this when there are cans in front of us, we couldn't lay the two joints flat without crushing or moving the cans. So the elbow joint calibrates by folding back onto the shoulder. This means that we measuring angles in opposite directions which is something compensated for in the software.

Installing the claw was a relatively simple process. We mounted it and the servo that powers it to the end of our elbow joint. To reach optimal performance in grabbing cans though it required a great deal of tweaking. Since the servo starts in a "zeroed" position where it can turn 127 indexes in either direction we tried various starting points. Our final solution allowed for a maximum amount of width in the open position while trading off a tight grip in the closed. This allowed for us to not miss on the majority of attempts to pick up a can.

Software Design

Our software architecture is based around a main control loop that can be seen in Figure 1 below. This assignment required some more complex operations to be performed at a software level than before, specifically in regards to the kinematics of our robot. Before we were able to use the claw we had to

demonstrate our robot's ability to hold weight and not perform any dangerous operations, like slamming into the ground.

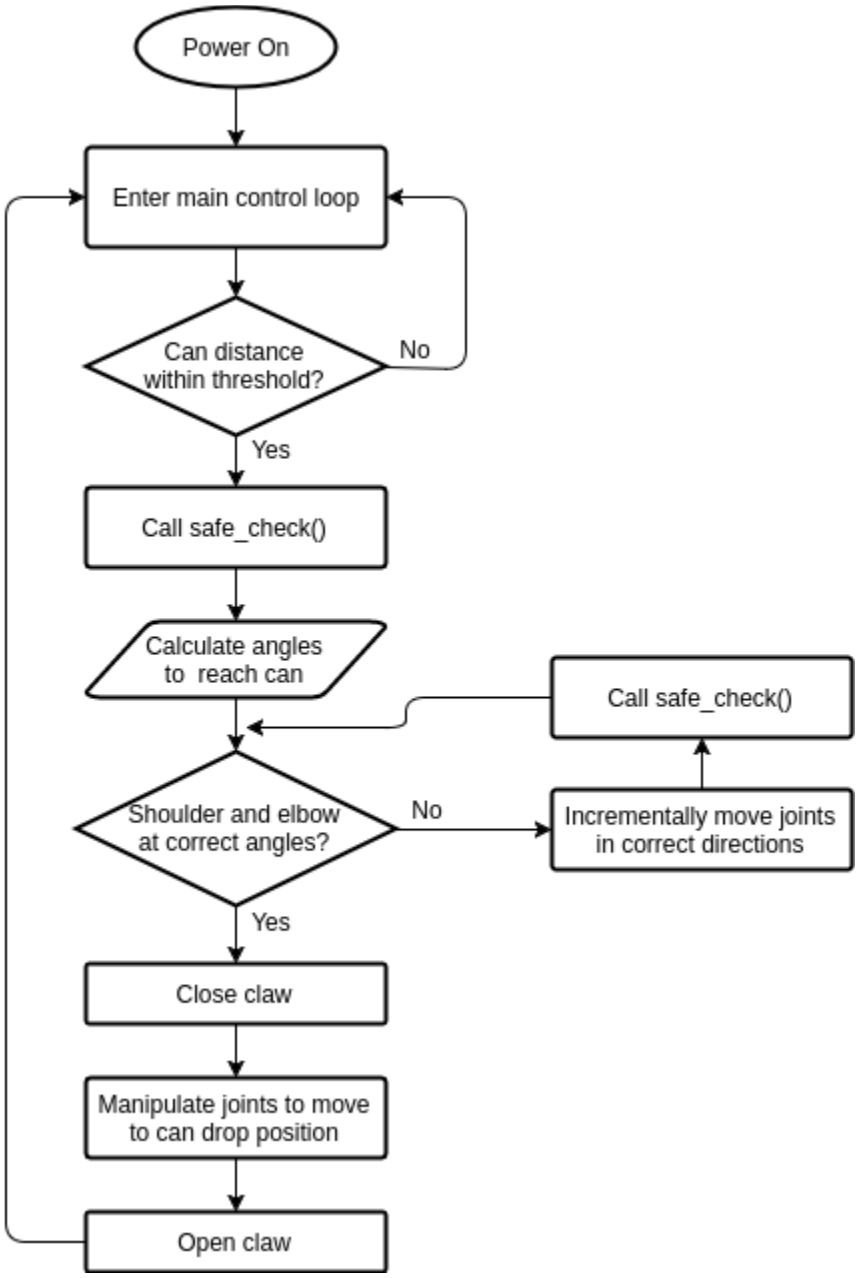

Figure 1: Main Control Loop

As noted in the mechanical design section above, our robot has a touch sensor below the elbow joint, but not one below the claw. This means that we have to determine from a software level where the claw is located in two dimensional space to prevent it from hitting the ground. This and all of the safety logic for this assignment is in a safe_check function that is shown in Figure 2 below. We achieved this using forward kinematics with the help of our quad encoders. Getting this to perform correctly took quite a bit of effort though. The first step was to calibrate our quad encoders to perfectly report the degrees of

rotation it observes. We recorded data of both encoders independently at different angles to find the correct equations. Once calibrated we began to test our kinematics code.

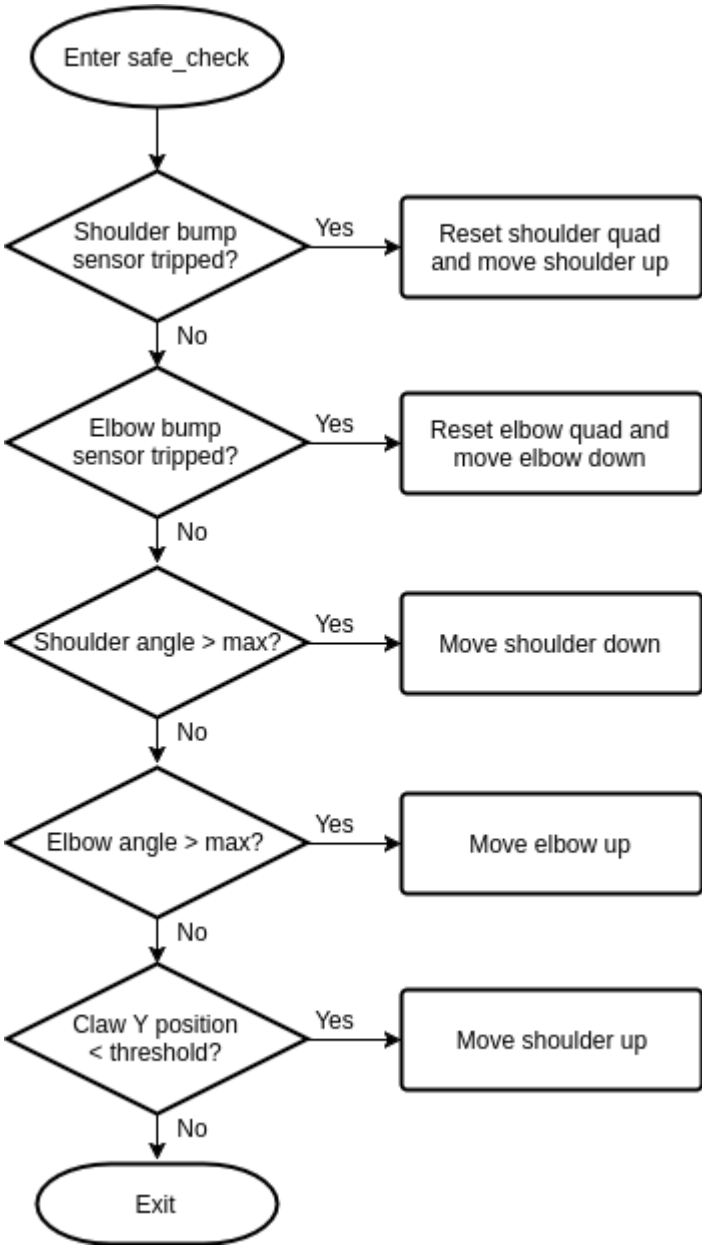

Figure 2: Safety Logic

Initially we received incorrect coordinates from our kinematics code which turned out to be because we used degrees instead of radians in the sin and cosine operations. Once we corrected our mistakes there, we started to see better looking data. While it was better it wasn't always correct. It appeared that the shoulder acted as a plane for the elbow, and y coordinate values would be positive or negative if the elbow was above or below the shoulder. We began to scrutinize our logic and the issue was related to our elbow measuring it's angle in the opposite direction of our shoulder.

We had been subtracting our reported degrees from 180 to find the correct angle of the elbow. While this does shift the 0 degrees point to be correct, it is still measuring from the wrong direction. The correct approach is to subtract our measurement from 360. Once this fix had been implemented our

results started to look correct, but there was a lack of consistency with them. After this lengthy debugging process we weren't sure where the source of this problem was since we revisited everything that may affect it.

We completed a final peer review of the logic for and everything appeared in order so we began to run several tests just to try and understand why the data is reported the way it is. The only thing we concluded was that the data was inconsistent, but a slight detail was that it was inconsistently inconsistent. When it performed poorly there was no correlation to the other times it performed poorly. This made us question the hardware itself, and this was a simple test to perform. We had never used our quad encoders in prior assignments so we couldn't verify it was functional from our own experience. We swapped out our quad encoder with one of the groups who already finished. This way we knew it must have worked since they had used it successfully. Sure enough, once the encoder was swapped our data began to be consistently correct. While this was a difficult issue to identify, it made us revisit all of our logic and step by step eliminate possible candidates for failure.

With being confident in our code and demonstrably able to control our two joints we were ready to proceed to the task of lifting cans. Controlling the claw was a simple task since it is just setting servo positions. The interesting problem was the inverse kinematics. Using the distance to the can detected using an ultrasonic sensor and a constant can height, we provide our code an X,Y position we want the claw to end up at. Inverse kinematics allows us to find the angles we need to move our shoulder and elbow to in order for the claw to be at that position.

Since our shoulder and elbow quad encoders are calibrated to exact degrees, once we found the desired angles all we needed to do was to move each joint until the encoder reached the desired value. Once the claw grabbed the can, the shoulder and elbow would just move to a hardcoded position that we set as the can drop off point. In between cans the shoulder and elbow joints would reset to ensure optimal performance.

Performance Evaluation

Our robot consistently ran into issues as we progressed throughout the assignment. Initial design decisions cost us valuable time and slowed our progress. This applied to smaller aspects of the robot such as our platform all the way to our choice of gear train. While the dual worm gears worked well, they allowed for a great deal of sway. This was problematic because it caused our shoulder gear train to slip gears consistently as it passed vertical. To compensate for this we prevented our shoulder from reaching 90 degrees via software.

However we still experienced this sway when the elbow passed vertical. This was slippage we couldn't overcome without redesigning the hardware, which we didn't have time to do. Fortunately though this was only a one tooth slip. The gear slipping by one tooth did not impact our performance when picking up cans. It was however uncomfortable to watch. Despite our challenges our robot was able to complete the task of picking up and dropping the three cans behind it.

Conclusion

While our robot managed to succeed we were left feeling like we could have done it much better if we started over from scratch. The difficulty of this task was a significant step up from previous assignments which made this a really interesting, but difficult, problem to tackle. This assignment also really emphasized the importance of a thorough debugging process, which enabled us to identify that we had a real hardware failure outside of our control.

Appendix

```
#pragma config(Sensor, dgtl1,  turn_left,      sensorTouch)
#pragma config(Sensor, dgtl2,  turn_right,     sensorTouch)
#pragma config(Sensor, dgtl3,  shldr_bump,     sensorTouch)
#pragma config(Sensor, dgtl4,  s_quad,         sensorQuadEncoder)
#pragma config(Sensor, dgtl6,  base_sonar,     sensorSONAR_inch)
#pragma config(Sensor, dgtl8,  e_quad,         sensorQuadEncoder)
#pragma config(Sensor, dgtl10, elbow_bump,     sensorTouch)
#pragma config(Sensor, dgtl11, shldr_move,     sensorTouch)
#pragma config(Sensor, dgtl12, shldr_move2,    sensorTouch)
#pragma config(Motor,  port2,           s_motor_2,     tmotorVex393, openLoop, reversed)
#pragma config(Motor,  port3,           s_motor_1,     tmotorVex393, openLoop, reversed)
#pragma config(Motor,  port4,           e_motor,       tmotorVex393, openLoop)
#pragma config(Motor,  port5,           claw_servo,    tmotorServoStandard, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

// Constants
int S_SPEED = 127;
int E_SPEED = 127;
int NUM_SAMPLES = 10;
int SHLDR_MIN = 0;
int SHLDR_MAX = 100;
float ELBOW_MIN_Y_POS = 1.5;
float ELBOW_MIN_X_POS = -1.5;
float LEN_SHLDR = 12.5;
float LEN_ELBOW = 12.5;
float SHLDR_RATIO = 1.119;
float ELBOW_RATIO = 1.158;
int SAFE_ELBOW_MAX = 270;
int CAN_HEIGHT = 2;
int SHLDR_CAN_RELEASE_ANGLE = 40;
int ELBOW_CAN_RELEASE_ANGLE = 55;
int s_dists[] = {69, 53, 39, 28, 19};
int e_dists[] = {235, 215, 200, 184, 173};

// Variables
int left_bump;
int right_bump;
int e_safe_bump;
int s_safe_bump;
int shldr_angle;
int elbow_angle;
float can_dist;
int shldr_target_angle;
int elbow_target_angle;
int s_bump;
int s2_bump;

// dir: -1(down) or 1(up)
```

```
// time: ms
// speed: 0-127
void move_shoulder(int dir, int time, int speed)
{
   speed = (dir == -1) ? -speed : speed;
   motor[s_motor_1] = speed;
   motor[s_motor_2] = speed;
   wait1Msec(time);
   motor[s_motor_1] = 0;
   motor[s_motor_2] = 0;
}

// -1 moves up 1 moves down
void move_elbow(int dir, int time, int speed)
{
   speed = (dir == -1) ? -speed : speed;
   motor[e_motor] = speed;
   wait1Msec(time);
   motor[e_motor] = 0;
}

void reset_shoulder()
{
   SensorValue[s_quad] = 0;
   shldr_angle = SensorValue(s_quad);
   writeDebugStreamLine("shoulder dist: %d", shldr_angle);
}

void reset_elbow()
{
   SensorValue[e_quad] = 0;
   elbow_angle = SensorValue(e_quad);
   writeDebugStreamLine("ELBOW DIST: %d", elbow_angle);
}

float get_x_pos(int s_pos, int e_pos)
{
   float theta1 = s_pos * SHLDR_RATIO + 5.195;
   float theta2 = 360 - (e_pos * ELBOW_RATIO + .885); // 360 appears to give right x
   float alpha = PI - theta1 - theta2;
   float x = (LEN_SHLDR * cosDegrees(theta1) - LEN_ELBOW * cosDegrees(alpha));
   return x;
}

float get_y_pos(int s_pos, int e_pos)
{
   float theta1 = s_pos * SHLDR_RATIO + 5.195;
   float theta2 = 360 - (e_pos * ELBOW_RATIO + .885);
   float alpha = PI - theta1 - theta2;
```

```
   float y = (LEN_SHLDR * sinDegrees(theta1) + LEN_ELBOW * sinDegrees(alpha));
   return y;
}

float find_elbow_angle(int x, int y)
{
   float num = pow(x,2) + pow(y,2) - pow(LEN_SHLDR,2) - pow(LEN_ELBOW,2);
   float denom = 2 * LEN_SHLDR * LEN_ELBOW;
   return acos(degreesToRadians(num/denom));
}

float find_shldr_angle(int x, int y, float theta2)
{
   float ft = atan(degreesToRadians(y/x));
   float nom = LEN_SHLDR * sin(theta2);
   float denom = sqrt(pow(x,2) + pow(y,2));
   float st = asin(degreesToRadians(nom/denom));
   return ft - st;
}

void safe_check()
{
   // Get current sensor values
   s_safe_bump = SensorValue(shldr_bump);
   e_safe_bump = SensorValue(elbow_bump);
         shldr_angle = SensorValue(s_quad);
         elbow_angle = abs(SensorValue(e_quad));

   float claw_x_pos = get_x_pos(shldr_angle, elbow_angle);
         float claw_y_pos = get_y_pos(shldr_angle, elbow_angle);
         //writeDebugStreamLine("s,e: %d, %d", shldr_angle, elbow_angle);
         //writeDebugStreamLine("x,y: %.2f, %.2f", claw_x_pos, claw_y_pos);

         // Ensure safe parameters
   if(s_safe_bump == 1){
      reset_shoulder();
            move_shoulder(1, 500, S_SPEED);
         }
         if(e_safe_bump == 1){
      reset_elbow();
            move_elbow(1, 100, E_SPEED);
         }
         else if(shldr_angle < SHLDR_MIN) {
            move_shoulder(1, 500, S_SPEED);
         }
         else if(shldr_angle >= SHLDR_MAX) {
            move_shoulder(-1, 500, S_SPEED);
         }
         else if(claw_y_pos < ELBOW_MIN_Y_POS) {
```

```
            writeDebugStreamLine("Claw y value below threshold");
            if(elbow_angle < 90)
            {
               move_elbow(1, 50, E_SPEED);
            }
            else
            {
               move_elbow(-1, 50, E_SPEED);
            }
         }
         else if(claw_x_pos < ELBOW_MIN_X_POS) {
      writeDebugStreamLine("Claw x value above threshold");
            move_elbow(1, 50, E_SPEED);
         }
         else {
            // Do nothing
         }
}

void move_shoulder_to(int pos)
{
   int dir = (shldr_angle - pos >= 0) ? -1 : 1;
   if(dir == 1)
   {
      while(shldr_angle <= pos)
            {
               safe_check();
               move_shoulder(dir, 20, S_SPEED);
            }
   }
   else
   {
      while(shldr_angle >= pos)
            {
               safe_check();
               move_shoulder(dir, 20, S_SPEED);
            }
   }
}

void close_claw()
{
    motor[claw_servo] = -127;
}

void open_claw()
{
    motor[claw_servo] = 127;
}
```

```
void move_elbow_to(int pos)
{
   int dir = (elbow_angle - pos >= 0) ? -1 : 1;
   int counter = 0;
   if(dir == 1)
   {
      while(elbow_angle <= pos)
            {
               if(counter > 400)
               {
                  open_claw();
                  break;
               }
               counter++;
               safe_check();
               move_elbow(dir, 20, S_SPEED);
               //wait1Msec(20);
            }
   }
   else
   {
      while(elbow_angle >= pos)
            {
               if(counter > 400)
               {
                  open_claw();
                  break;
               }
               counter++;
               safe_check();
               move_elbow(dir, 20, S_SPEED);
               //wait1Msec(20);
            }
   }
}

void force_reset_shoulder()
{
   s_safe_bump = SensorValue(shldr_bump);
   while(s_safe_bump == 0)
   {
      s_safe_bump = SensorValue(shldr_bump);
      writeDebugStreamLine("safe bump: %d", s_safe_bump);
      move_shoulder(-1, 100, S_SPEED);
   }

   reset_shoulder();
}
```

```
// Run this after force_reset_shldr
void force_reset_elbow()
{
   e_safe_bump = SensorValue(elbow_bump);
   while(e_safe_bump == 0)
   {
      e_safe_bump = SensorValue(elbow_bump);
      move_elbow(-1, 20, E_SPEED);
      wait1Msec(20);
   }

   reset_elbow();
}

void get_cans()
{
   for(int i = 0; i < 3; i++) // iter over num cans
   {
      open_claw();
      wait1Msec(500);
      writeDebugStreamLine("opened claw");
      move_elbow_to(20);
      move_shoulder_to(50);
      move_elbow_to(180);
      move_shoulder_to(75);
      writeDebugStreamLine("staged");
      writeDebugStreamLine("ACQUIRING TARGET");
      int scan = SensorValue(base_sonar);
      writeDebugStreamLine("scan: %d", scan);
      float perc = floor(((scan-14)/8.5)*5);
      writeDebugStreamLine("perc: %f", perc);
      int target_s = s_dists[(int)perc];
      int target_e = e_dists[(int)perc];
      writeDebugStreamLine("TARGET ACQUIRED");
      writeDebugStreamLine("target s: %d", target_s);
      writeDebugStreamLine("target e: %d", target_e);
      wait1Msec(1000);
      move_elbow_to(target_e);
      wait1Msec(4000);
      writeDebugStreamLine("moved elbow");
      move_shoulder_to(target_s);
      writeDebugStreamLine("moved shldr");
      close_claw();
      wait1Msec(500);
      writeDebugStreamLine("close claw");
      move_shoulder_to(shldr_angle+5);
      move_elbow_to(elbow_angle-60);
      writeDebugStreamLine("move to drop pos");
```

```
        move_shoulder_to(shldr_angle-5);
        move_elbow_to(elbow_angle-20);
        move_shoulder_to(SHLDR_CAN_RELEASE_ANGLE);
        move_elbow_to(ELBOW_CAN_RELEASE_ANGLE);
        open_claw();
        wait1Msec(500);
        force_reset_shoulder();
        force_reset_elbow();
    }
}

task main()
{
    wait1Msec(2000); // give stuff time to turn on

    // Reset joints
    force_reset_shoulder();
    force_reset_elbow();

    // Pick up cans
    get_cans();

    // Testing control loop
    while(true)
    {
        left_bump = SensorValue(turn_left);
                    right_bump = SensorValue(turn_right);
                    s_safe_bump = SensorValue(shldr_bump);
                    shldr_angle = SensorValue(s_quad);
                    s_bump = SensorValue(shldr_move);
                    s2_bump = SensorValue(shldr_move2);

                    //writeDebugStreamLine("shoulder angle: %d", shldr_angle * SHLDR_RATIO + 5.195);
                    //writeDebugStreamLine("elbow dist: %d", elbow_angle * ELBOW_RATIO + .885);
        //writeDebugStreamLine("elbow dist: %d", elbow_angle);
                    safe_check();

                    if(left_bump == 1)
                    {
                      move_elbow(-1, 20, E_SPEED);
                      //move_shoulder(-1, 20, S_SPEED);
                    }
                    else if(right_bump == 1)
                    {
                      move_elbow(1, 20, E_SPEED);
                      //move_shoulder(1, 20, S_SPEED);
                    }
                    else if(s_bump == 1)
                    {
```

```
            move_shoulder(-1, 20, S_SPEED);
        }
        else if(s2_bump == 1)
        {
            move_shoulder(1, 20, S_SPEED);
        }
        else
        {
            motor[s_motor_1] = 0;
            motor[s_motor_2] = 0;
        }
    }
}
```