Arthur Hernandez

Tuesday December 7th

Autonomous Robotics

Assignment 5: Manipulation Report

Introduction

A robot manipulator isn't like any other normal robot. Its main functionality is to provide a task by moving different parts or jointed segments of itself. These types of robots usually take form of arms and bodies and are commonly referred to as robotic arms. The goal of our final assignment is to explore issues in the construction and control of robotic manipulators. Our manipulators will be 2-joint planar robots that will reach out in front of them, grab 3 cans one at a time from a selected section of the table using a gripper, and drop them anywhere behind the manipulators "shoulder" joint.

Mechanical Design

The Robot consisted of 3 main parts: the base or body, the upper arm, and the lower arm. Naturally first, we had to start with the base. Its design revolves around the fact that in order to counteract the robots extended arm weight, other weight had to be collected on the opposite side. We also had to make sure that there was enough space for the first gear train to sit on the inside height and width wise. The base would be used to hold all of the drivers for the arm components in place so we needed it to be as sturdy and secure as possible. For these reasons we created a rectangular base with the front end opened so that the arm could have a full range of motion. We placed the battery pack, the microcontroller, and some heavier items on the other side. Although the outer rectangle base was open, it was not frail as plates bridged from one side to the other.

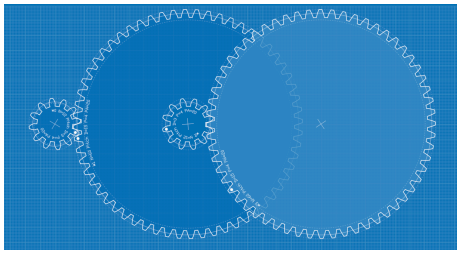


Figure 1

My team thought it would be easiest to create a

gear train design that would work for both the shoulder joint and the elbow joint. This type of design used 2 12 tooth and 2 60 tooth gears. Our gear design can be seen in figure 1 from left to right. The motor was connected using a short axle through each side of the the base plates that sandwiched the first 12 tooth gear. The first 60 tooth gear came after. This one having an axle connection to the 2nd 12 tooth gear. The last 60 tooth gear was then connected to that. This last gear will be used to get optical encoder information about the amount of turns the motor/train has outputted and to turn the shoulder (See figure 2).

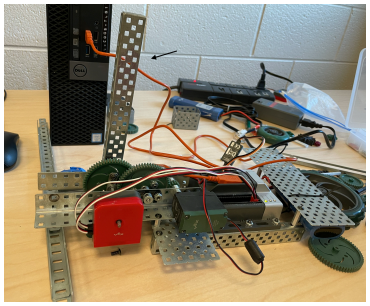
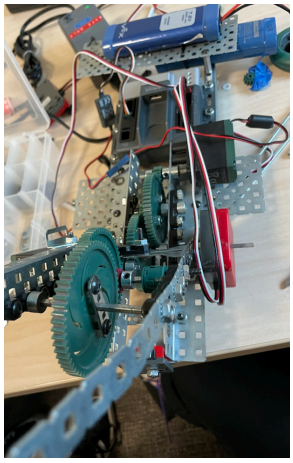


Figure 2

The lower arm gear train proved to be much more difficult to place in a secure structure. For one, the lower arm gear train will be moving on the upper arm at the elbow the entire time. Our main issue was having run out of metal pieces in our kit that would fit our design for the other side of the lower arm. We needed a strong yet light piece of metal but we had to improvise with some of the flexible, yet very light 1x 20 pieces. Since it was flexible we would have to build a brace connecting both sides of the upper arm. This brace was made with 2 T-shaped connected together then connected to each side of the upper arm. This seemed to work for holding everything together and so we added the encoder and motor (See figure 3 for an angled view and 4 for a birds eye view of the gear train with the brace). In order to test our robot we added 2 different buttons to control up and down motor movements for the shoulder, and another 2 for the elbow. This would help us figure out where our highest and lowest bounds would be so that our robots claw would never hit the table. After adding the claw to the end of the upper elbow with 4 very secure screws we decided to place a sonar sensor directly to and below the claw. This would help us determine whether or not a can was in view.



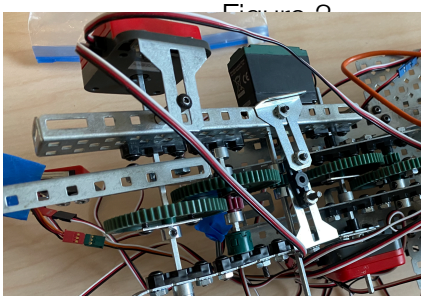


Figure 4

This did not end up working because the claw would think that the table was a can. It would also try to grasp a can too high above it or just not work at all when it was directly in front of one. For this reason we changed the ultrasonic sonar sensor to be placed on the robot's body instead. This would mean it was stationary, but its measurements were reliable. Our bot was also not giving us enough reaching distance that would fit all 3 cans. One way that this could be fixed was to allow for the bot to reach closer to itself. This was done by lifting the bot and placing it on 4 "legs" that were a mix of metal cubes and rubber wheels. Our sonar sensor holding onto the front of these legs not obscured by anything (See Figure 5).

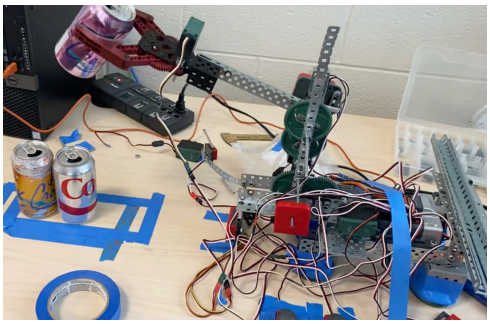


Figure 5

Software Design

Our team has gotten into a routine of creating a new program to test different parts of the assignment. Therefore, the software design varied significantly from one part of the project to another. For example in order to place our claw onto the bot we had to first prove that our bot would not exit any bounds and hit the table. For the shoulder we used manipulatorSonarOffClaw.c (See Appendix). This program would only use the shoulder optical encoder to test boundaries. These bounds were found manually, having an upper bound of -30 and lower bound of 30. A couple of if statements would check to see if it were out of bounds. If it was too high the motor would instead go down. If it were too low the motor would go up.

Our second Software design was for also checking the upper and lower bounds of the elbow so that the claw would never smash into the table ShoulderTestCodePassedCheck.c. Here the same design that was used for the shoulder was kept, but was also implemented for the elbow. Furthermore instead of having to test the bounds by coding the motor to move for some time a certain direction, we added 4 buttons and coded them each to either elbowUp(), elbowDown() shoulderUp(), or shoulderDown(). These buttons were incredibly useful later on since they could be used real time and could be turned on or off by changing a while loop

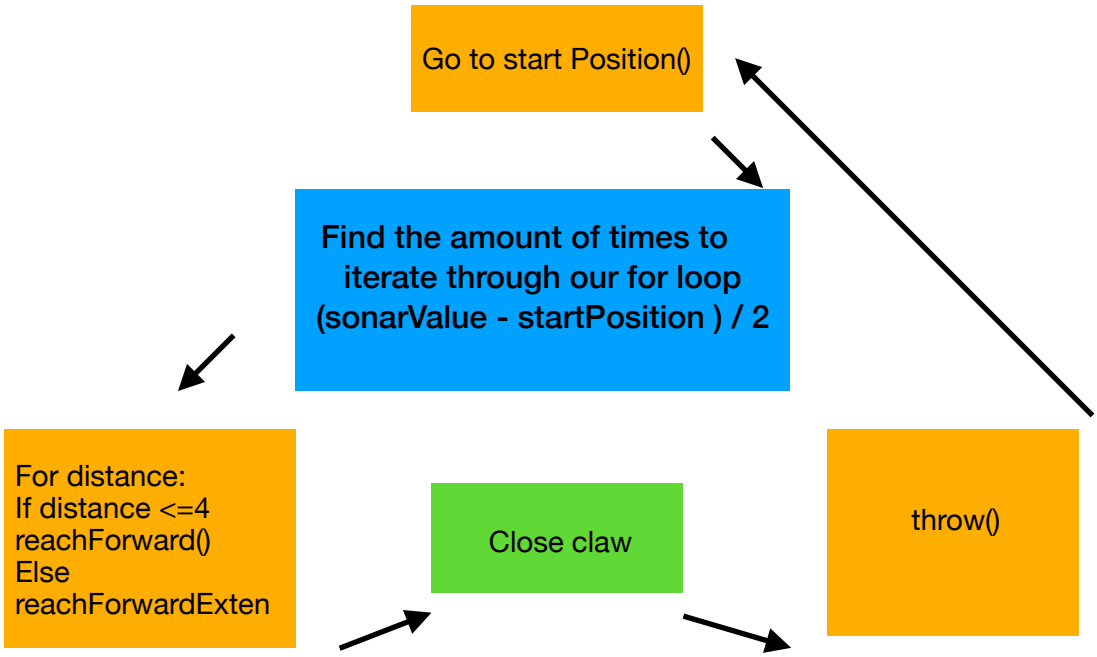
clause instead of an entire method. To this same program we added our claw and started brainstorming ways that this assignment could be completed. We wanted to make sure that our plan to place the sonar sensor on the claw was viable so we wrote clawTest.c that just closes the claw if the sensor feels something very close in front of it. We thought it would be easier to use calibrated movements for shoulderDown() and elbowUp() together instead of having to find our reverse kinematics equation and were able to figure out that with a reoccurring starting point this would be possible.

We brought all of these Ideas into our final fresh software design fullManipulator.c. From our experiments we were able to produce numbers that would make our claws movement lateral and equidistant at a rate of 2 centimeters every time.
Our necessary starting points were -4 for the shoulder and 2 for the elbow encoders. This starting position is 14.5 cm away from the sonar sensor to the tip of the claw. This means that we could move at intervals of 2cm for anywhere past that up to 33.5 cm giving us 19cm of reach space. Whenever the claw gets farther than 22.5 cm The claw starts getting very close to the ground. For this reason we added a separate method to call elbow up twice instead of just once when our arm distance was more than > 22.5cm. The way we calculated the current distance of the claw was by making a for loop that would run int distance = (int)((sonarV-14.5) / 2) times. Each time calling either reachForward() (the normal method) when distance is <. 4 (aka <= 22.5 cm) or reachForward() extended when > 22.5cm (See Figure 6).

Distance	**<= 22.5 cm reachForward()**	**> 22.5cm reachForwardExtended**
shoulderDown() 500 Msec @ motor = -20	called twice	Called twice
elbowUp() 600msec @ motor = -30	Called once	Called twice

Figure 6

After this we just created a throw() method that would open the claw when shoulderUp() reached a height of -6 and elbowUp() reached behind the shoulder at -130 on the encoder. With these these methods we created a simple yet powerful iteration 3 times for 3 cans.



Performance Evaluation

I thought that our although everything seemed to work well enough, there could have been even more improvements to our mechanical design. On occasions during the testing phases there would be problems with the upper arm gear train. I believe that these problems came from the fact that everything was not secured and sandwiched tight enough to keep gears from creating lots of friction or having them get jammed or having them come loose. Another mechanical design issue that we never completely solved was that the corner at where the upper arm connected to the base was getting suck. The bots movement was fine everywhere before or after the metal pieces corner, so we decided it was too small of an issue to be concerned when under a time crunch (See Figure 7). Another simple issue was that after picking up a can, our throw method was not as detailed as it could have been causing cans close to the can being picked up to get slid around. This would mean that cans could get push out of the reach area. It could have been simply fixed by moving the elbow down a little after bringing the shoulder up. I was very impressed by how often the bot would pick up the can if it were not

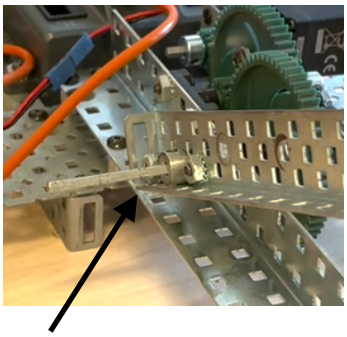


Figure 7

pushed out of bounds. Even if it were slightly too left or right the approach of the claw would shimmy it into the range. We could have solved our kinematics more accurately but they were really not necessary unless we were going to need a much more accurate bot. There were no large scale issues that compromised the effectivity of the bot other than having the cans be pushed too far out of reach and because of this I think our bot was successful.

Conclusion

Overall this was one of the most enjoyable projects from the semester. Combining the learning of physics, mathematics, and coding isn't as bad as it sounds. If I were to start this project over, I would have worried less on the weight of it as it did not prove to be an issue. It would have allowed for other parts to go more smoothly. I also would make sure to follow the same process of breaking down the problem into smaller subproblems and creating separate methods for those. It felt much easier to handle and create something that worked well. Me and my team learned how to multitask much more in this assignment than in any previous ones. We never really felt like we were running out of time and too the project piece by piece keeping the final product in mind.

Appendix

clawTest.c

```
#pragma config(Motor,  port4,           clawServo,     tmotorServoStandard, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

// claw test

task main()
{
  //for(int i = 0; i < 10; i++){
  //  motor[clawServo] = i;
  //  wait1Msec(10);
  //}
  wait1Msec(1000);
  motor[clawServo] = -75;
  // -75 holds a can
  wait1Msec(2000);
}
```

manipulatorSonarOffClaw.c

```
#pragma config(Sensor, dgtl10, shoulderLimit,  sensorTouch)
#pragma config(Sensor, dgtl11, shoulderEncoder, sensorQuadEncoder)
#pragma config(Motor,  port2,           shoulderM,     tmotorServoContinuousRotation,
openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

//global variables
// shoulder limits
int sUpperBound = -30;
int sLowerBound = 30;

// currently we have one gear train and a bicep, one motor and
// shoulder motor positive is up, negative is down.
// integer function that takes the quad encoder location and spits out if the shoulder arm is
within bounds
int outOfBounds(int location){
   if(SensorValue(shoulderLimit)){
      return(2);
   }
   if(location < sUpperBound){
      return(1);
   }
   if(location > sLowerBound){
      return(2);
   }
   return(0);
}
// test to see if the system works by moving between the extremes of motion
void testLoop(){

   int shoulderEVal = SensorValue(shoulderEncoder);
   motor[shoulderM] = 20;
   while(true){
      if(SensorValue(shoulderLimit)){
         motor[shoulderM] = 20;
      }
      shoulderEVal = SensorValue(shoulderEncoder);
      if(outOfBounds(shoulderEVal) == 2){
         motor[shoulderM] = 20;
      }
      else if(outOfBounds(shoulderEVal) == 1){
         motor[shoulderM] = -20;
      }

   }
}

task main()
```

```
{
  // log locations and run test
  writeDebugStream("%d\n",SensorValue(shoulderEncoder));
  testLoop();
}
```

ShoulderTestCodePassedCheck.c

```
#pragma config(Sensor, dgtl1,  shoulderUpButton, sensorTouch)
#pragma config(Sensor, dgtl2,  shoulderDownButton, sensorTouch)
#pragma config(Sensor, dgtl3,  elbowUpButton,  sensorTouch)
#pragma config(Sensor, dgtl4,  elbowDownButton, sensorTouch)
#pragma config(Sensor, dgtl5,  sonarC,         sensorSONAR_cm)
#pragma config(Sensor, dgtl7,  elbowLimit,     sensorTouch)
#pragma config(Sensor, dgtl8,  elbowEncoder,   sensorQuadEncoder)
#pragma config(Sensor, dgtl10, shoulderLimit,  sensorTouch)
#pragma config(Sensor, dgtl11, shoulderEncoder, sensorQuadEncoder)
#pragma config(Motor,  port2,           shoulderM,     tmotorServoContinuousRotation,
openLoop)
#pragma config(Motor,  port3,           elbowM,        tmotorServoContinuousRotation, openLoop)
#pragma config(Motor,  port4,           clawServo,     tmotorServoStandard, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

//global variables
// Optical shaft encoder values the arm shouldnt go past
// Shoulder
int sUpperBound = -30;
int sLowerBound = 30;
//Elbow
int eUpperBound = 40;
int eLowerBound = -40;
//Limits for the claw servo
int clawOpen = 125;
int clawClose = -75;
bool clawFlip = false;
int sonarV = 10;

// method to close the claw when the sonar reads 0 to close around a can

bool clawCheck(){
  sonarV = SensorValue(sonarC);
  if(sonarV == -1){
    // Do nothing if reading a -1
  }
  // check if the sonar distance is less than 5 and close the claw
  else if(sonarV < 5){
    // clawflip ensures that later in the loop the claw stays in the position
    clawFlip = true;
    return true;
  }

  else{
    clawFlip = false;
```

```
        return false;
    }
}
// methods to consistently move the manipulator

// shoulder positive value is up, negative is down
void shoulderUp(){
    motor[shoulderM] = 20;
    wait1Msec(1000);
    motor[shoulderM] = 0;
}

void shoulderDown(){
    motor[shoulderM] = -20;
    wait1Msec(1000);
    motor[shoulderM] = 0;
}
// for elbow, negative motor is up, positive is down
void elbowUp(){
    motor[elbowM] = 25;
    wait1Msec(1000);
    motor[elbowM] = 0;
}
void elbowDown(){
    motor[elbowM] = -20;
    wait1Msec(750);
    motor[elbowM] = 0;
}

void reachForward(){
    elbowUp();
    shoulderDown();
}

// shoulder motor positive is up, negative is down.

// Move the manipulator to the start position
// Starting position is shoulder at -30, elbow at -72
void startMove(){

    while(SensorValue(shoulderEncoder) > -28){
        writeDebugStream("shoulderup\n");
        shoulderUp();
    }
    while(SensorValue(elbowEncoder) > -60){

        elbowDown();
    }
}

// Throw the can once its picked up
void throw(){
    //shoulder -30 elbow 80
    while(SensorValue(shoulderEncoder) > -30){
```

```
        shoulderUp();
    }
    while(SensorValue(elbowEncoder)< 70){
        elbowDown();
    }
}

task main(){
    // Loop for button control
    while(true){

        //wait10Msec(10);
        writeDebugStream("Shoulder Value: %d\n",SensorValue(shoulderEncoder));
        writeDebugStream("Elbow Value: %d\n",SensorValue(elbowEncoder));
        sonarV = SensorValue(sonarC);

        if(SensorValue(shoulderUpButton) == 1){
            shoulderUp();
        }
        if(SensorValue(shoulderDownButton) == 1){
            shoulderDown();
        }
        if(SensorValue(elbowUpButton) == 1){
            elbowUp();
        }
        if(SensorValue(elbowDownButton) == 1){
            elbowDown();

            writeDebugStream("button pressed ELBOW: %d\n",SensorValue(elbowEncoder));
        }

        clawCheck();
        if(clawFlip){
            writeDebugStream("closing servo\n");

            motor[clawServo] = clawClose ;
            wait1Msec(30);
            //wait1Msec(500);
        }
        else{
            writeDebugStream("opening servo\n");
            motor[clawServo] = clawOpen;
            wait1Msec(30);
            //reachForward();
        }
    }

    // simple algorithm to pick up a can as it is found. Using the sonar atached to the claw

    startMove();
    bool switcher = false;
    // while the claw doesnt hold a can.
    while(!clawCheck()){
        // alternates elbow up and shoulder down resulting in reaching directly forward
```

```
        if(switcher){
            elbowUp();
            elbowUp();

        }
        else if(!switcher){
            shoulderDown();
        }
        switcher = !switcher;

    }
    // when it has a can in the claw,throw
    throw();




    // Print final values
    writeDebugStream("%d\n",SensorValue(elbowEncoder));
    writeDebugStream("%d\n",SensorValue(shoulderEncoder));

}
```

shoulderTestCodePassedCheck.c

```
#pragma config(Sensor, dgtl1,  shoulderUpButton, sensorTouch)
#pragma config(Sensor, dgtl2,  shoulderDownButton, sensorTouch)
#pragma config(Sensor, dgtl3,  elbowUpButton,  sensorTouch)
#pragma config(Sensor, dgtl4,  elbowDownButton, sensorTouch)
#pragma config(Sensor, dgtl7,  elbowLimit,     sensorTouch)
#pragma config(Sensor, dgtl8,  elbowEncoder,   sensorQuadEncoder)
#pragma config(Sensor, dgtl10, shoulderLimit,  sensorTouch)
#pragma config(Sensor, dgtl11, shoulderEncoder, sensorQuadEncoder)
#pragma config(Motor,  port2,           shoulderM,     tmotorServoContinuousRotation,
openLoop)
#pragma config(Motor,  port3,           elbowM,        tmotorServoContinuousRotation, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

//global variables
int sUpperBound = -30;
int sLowerBound = 30;
int eUpperBound = 40;
int eLowerBound = -40;

//TODO remove the waits and add checks for the limit switches
// shoulder positive value is up, negative is down
void shoulderUp(){
    motor[shoulderM] = 20;
    wait1Msec(1000);
    motor[shoulderM] = 0;
}
// moves the shoulder down
void shoulderDown(){
```

```
    // check if the shoulder has moved too far
    if(SensorValue(shoulderLimit) == 1){
        motor[shoulderM] = 0;
        }else{
        motor[shoulderM] = -20;
        wait1Msec(1000);
        motor[shoulderM] = 0;
    }
}
// for elbow negative motor is up, positive is down
void elbowUp(){
    motor[elbowM] = -25;
    wait1Msec(1000);
    motor[elbowM] = 0;
}
void elbowDown(){
    if(SensorValue(elbowLimit) == 1){
        motor[elbowM] = 0;
        }else{
        motor[elbowM] = 25;
        wait1Msec(1000);
        motor[elbowM] = 0;
    }
}

// currently we have one gear train and a bicep, one motor and
// shoulder motor positive is up, negative is down.
// integer function that takes the quad encoder location and spits out if the shoulder arm is
within bounds
int sOutOfBounds(int location){
    if(SensorValue(shoulderLimit)){
        return(2);
    }
    if(location < sUpperBound){
        return(1);
    }
    if(location > sLowerBound){
        return(2);
    }
    return(0);
}

// integer function that takes th quad encoder location and spits out if the elbow arm is within
bounds
int eOutOfBounds(){
    int location = SensorValue(shoulderLimit);
    if(location){
        return(2);
    }
    if(location < eUpperBound){
        return(1);
    }
    if(location > eLowerBound){
        return(2);
```

```
    }
    return(0);
}
// check the elbow limit
int eLimitCheck(){
    return(SensorValue(elbowLimit));
}

// loop to move the shoulder between its extremes
void testLoop(){

    int shoulderEVal = SensorValue(shoulderEncoder);
    motor[shoulderM] = 20;
    while(true){
        if(SensorValue(shoulderLimit)){ // not set up yet
            motor[shoulderM] = 20;
        }
        shoulderEVal = SensorValue(shoulderEncoder);
        if(sOutOfBounds(shoulderEVal) == 2){
            motor[shoulderM] = 20;
        }
        else if(sOutOfBounds(shoulderEVal) == 1){
            motor[shoulderM] = -20;
        }
    }
}

// elbow test similar to shoulder test, moving the arm between its extremes
void elbowTest(){
    //motor[elbowM] = 25;
    //wait1Msec(1000);
    //motor[elbowM] = 0;
    int shoulderEVal = SensorValue(shoulderEncoder);
    motor[shoulderM] = 20;
    while(true){
        if(SensorValue(shoulderLimit)){
            motor[elbowM] = 20;
        }
        shoulderEVal = SensorValue(shoulderEncoder);
        if(eOutOfBounds() == 2){
            motor[elbowM] = 20;
        }
        else if(eOutOfBounds() == 1){
            motor[elbowM] = -20;
        }

    }
}

// main loop
task main()
{
    // loop to respond to button controls
    while(true){
```

```
        if(SensorValue(shoulderUpButton) == 1){
            shoulderUp();
        }
        if(SensorValue(shoulderDownButton) == 1){
            shoulderDown();
        }
        if(SensorValue(elbowUpButton) == 1){
            elbowUp();
        }
        if(SensorValue(elbowDownButton) == 1){
            elbowDown();
        }
    }

    writeDebugStream("%d\n",SensorValue(elbowEncoder));
    shoulderUp();
    writeDebugStream("%d\n",SensorValue(elbowEncoder));
}
```

fullManipulator.c

```
#pragma config(Sensor, dgtl1,  shoulderUpButton, sensorTouch)
#pragma config(Sensor, dgtl2,  shoulderDownButton, sensorTouch)
#pragma config(Sensor, dgtl3,  elbowUpButton,  sensorTouch)
#pragma config(Sensor, dgtl4,  elbowDownButton, sensorTouch)
#pragma config(Sensor, dgtl5,  sonarC,         sensorSONAR_cm)
#pragma config(Sensor, dgtl7,  elbowLimit,     sensorTouch)
#pragma config(Sensor, dgtl8,  elbowEncoder,   sensorQuadEncoder)
#pragma config(Sensor, dgtl10, shoulderLimit,  sensorTouch)
#pragma config(Sensor, dgtl11, shoulderEncoder, sensorQuadEncoder)
#pragma config(Motor,  port2,           shoulderM,     tmotorServoContinuousRotation,
openLoop)
#pragma config(Motor,  port3,           elbowM,        tmotorServoContinuousRotation, openLoop)
#pragma config(Motor,  port4,           clawServo,     tmotorServoStandard, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

//global variables
// Optical shaft encoder values the arm shouldnt go past
// Shoulder
int sUpperBound = -30;
int sLowerBound = 30;
//Elbow
int eUpperBound = 40;
int eLowerBound = -40;
//Limits for the claw servo
int clawOpen = 125;
int clawClose = -75;
bool clawFlip = false;
int sonarV = 10;
bool buttonControl = false;

// method to close the claw when the sonar reads 0
// TODO keep the claw closed as needed
```

```
bool clawCheck(){
   sonarV = SensorValue(sonarC);
   if(sonarV == -1){
      // skip if registering a -1
   }
   else if(sonarV < 5){
      // if can is within reach
   clawFlip = true;
      return true;
   }
   else{
      clawFlip = false;
      return false;
      //motor[clawServo] = clawOpen;
   }
}


// shoulder positive value is up, negative is down
// methods to consistently move the shoulder up and down
void shoulderUp(){
   motor[shoulderM] = 20;
   wait1Msec(500);
   motor[shoulderM] = 0;
}

void shoulderDown(){
   motor[shoulderM] = -20;
   wait1Msec(500);// dropped from 1000
   motor[shoulderM] = 0;
}
// for elbow, negative motor is up, positive is down
// method for moving the elbow consistently
void elbowUp(){
   motor[elbowM] = -30; // up from 25
   wait1Msec(600);
   motor[elbowM] = 0;
}
void elbowDown(){
   motor[elbowM] = 20;
   wait1Msec(750);
   motor[elbowM] = 0;
}

// consistently move the arm forward as needed
void reachForward(){
   elbowUp();
   shoulderDown();
   shoulderDown();
}
// double the elbow ups if needed
void reachForwardExtended(){
   elbowUp();
   elbowUp();
```

```
    shoulderDown();
    shoulderDown();
}

void shoulderReach(){
    shoulderDown();
    shoulderDown();
    motor[elbowM] = 30; // up from 25
    wait1Msec(300);
    motor[elbowM] = 0;
}

// shoulder motor positive is up, negative is down.

// Starting position is shoulder at -30, elbow at -72// position keeps changing
void startMove(){
    //while(SensorValue(shoulderEncoder) > -28 && SensorValue(elbowEncoder) > -28)
    while(SensorValue(shoulderEncoder) > -4){
        shoulderUp();
    }
    while(SensorValue(elbowEncoder) > 2){ // should be 6
        elbowDown();
    }
}

// fix
// Once a can is picked up, this will throw it behind the robot
void throw(){
    //shoulder -30 elbow 80
    while(SensorValue(shoulderEncoder) > -6){
        shoulderUp();
    }
    while(SensorValue(elbowEncoder) < 130){ // should be 6
        elbowUp();
    }
    clawFlip = false; // moved down, drop the can
}

task main(){
    // check if changed to button control
    while(buttonControl){//buttonControl
        //wait10Msec(10);
        writeDebugStream("Shoulder Value: %d\n",SensorValue(shoulderEncoder));
        writeDebugStream("Elbow Value: %d\n",SensorValue(elbowEncoder));
        sonarV = SensorValue(sonarC);

        if(SensorValue(shoulderUpButton) == 1){
            shoulderUp();
        }
        if(SensorValue(shoulderDownButton) == 1){
            shoulderDown();
        }
        if(SensorValue(elbowUpButton) == 1){
            elbowUp();
```

```
    }
    if(SensorValue(elbowDownButton) == 1){
        elbowDown();

        writeDebugStream("button pressed ELBOW: %d\n",SensorValue(elbowEncoder));
    }
    clawCheck();
    if(clawFlip){
        writeDebugStream("closing servo\n");

        motor[clawServo] = clawClose ;
        wait1Msec(30);
        //wait1Msec(500);
    }
    else{
        writeDebugStream("opening servo\n");
        motor[clawServo] = clawOpen;
        wait1Msec(30);
        //reachForward();
    }
}
// Main Loop for can pick up
int j = 0;
// for each can
for(j ; j < 3 ; j++){
    motor[clawServo] = clawOpen;
    // move to starting position
    startMove();
    // log positions for calibration as needed
    writeDebugStream("elbow: %d\n",SensorValue(elbowEncoder));
    writeDebugStream("shoulder: %d\n",SensorValue(shoulderEncoder));

    int i = 0; // should be 0
    sonarV = SensorValue(sonarC);
    // equation to convert sonar values in to claw distance from the start position
    int distance = (int)((sonarV-14.5) / 2);
    writeDebugStream("distance: %d\n",distance);
    // for distance away, move forward
    for(i; i < distance ; i++){
        // check if extended reach is needed to stay on target as the arm extends
        if(i >= 4){
            if(i >=8){
                shoulderReach();
            }else{
                reachForwardExtended();
            }

            }else{
            reachForward();
        }
    }
    // close the claw and throw the can
    motor[clawServo] = clawClose;
    wait10Msec(50);
```

```
        throw();
    }
    // log final positions
    writeDebugStream("%d\n",SensorValue(elbowEncoder));
    writeDebugStream("%d\n",SensorValue(shoulderEncoder));
}
```