Manipulation

I. Intro

This assignment was our longest by far and required six weeks for completion. We had to build a 2-joint planar robotic manipulator that could reach out and pick up three coke cans placed linearly in front of it, and drop said cans behind itself. The manipulator was supposed to function similarly to a human arm. The notable difference was that the hand at the end of the arm was replaced with a simple claw. My approach to this problem was to build four separate pieces: The base, the shoulder, the elbow, and the claw, and combine them to form the final product.

II. Mechanical Design

I will separate this section into the four parts I described above, and describe the process it took to build each piece of the robot from our initial model to our final working manipulator.

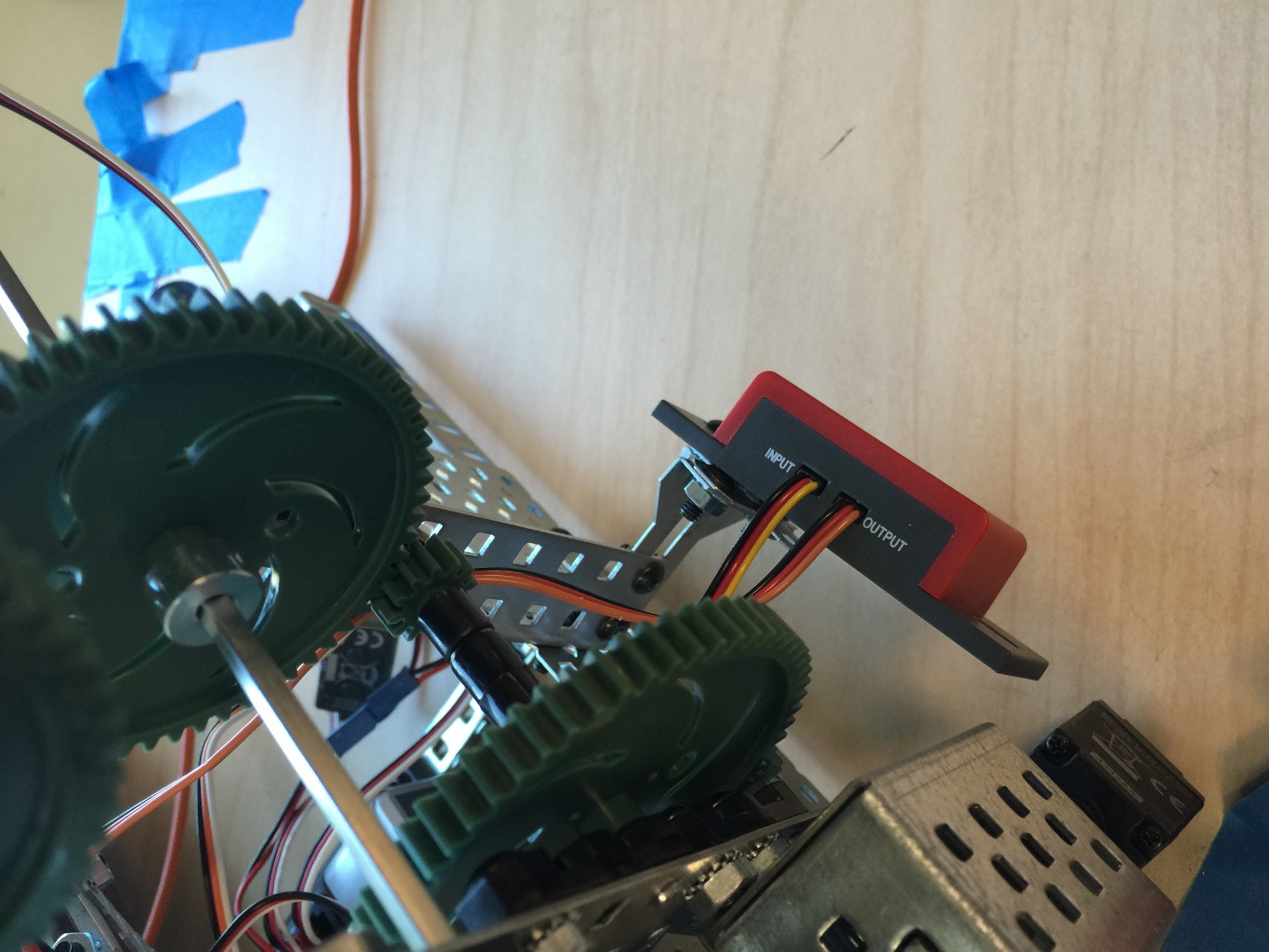
1. The Base

The base of our robot was designed very similarly to that of SquareBot. However, the number one difference was the lack of wheels and motors beneath the robot. This allowed us to build a very minimalist base with only 3 metal pieces, the vex controller.

After building the rest of our robot, we decided to flip our manipulator so as to increase our reach, and avoid smashing our claw into any of the wires or other sensitive objects on the (then) front of our robot.

We soon realized that the weight of the manipulator on the opposite side of robot was making the robot fall over. In order to correct this issue, we added a metal plate to the back of our robot and placed some text books upon it. This sufficiently balanced the weight, and allowed the manipulator to move freely without toppling the robot. Later, these book were replaced by wooden blocks, remnants of the maze project.

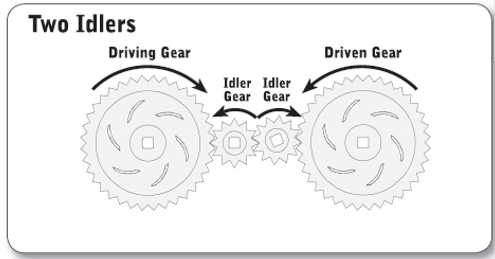
The final adjustment to the base took place after all other adjustments had been made on the robot. A sonar sensor was mounted onto what was now the front of the robot in order to detect the distance between the robot and the cans that required manipulation. We ran into an issue wherein the arm was slightly angled, and thus moved out of line with the sonar as it bent at the shoulder and subsequently the elbow. We fixed this by adding a metal piece to secure onto the original sonar bracket, and thereby allow the sonar to move slightly to the left and angle along with the arm.



1. The Shoulder

Easily the most hardware on the robot existed in the shoulder. The shoulder underwent massive modification before it was fully functional. Our first attempt at the shoulder joint was a simple motor attached to a small 12 –tooth gear powering two large 60-tooth gears. The overall torque on the shoulder was a measly 5x, and it was unable to lift more than one metal bar that we planned to use for the shoulder without bending the gears and making awful sounds. We quickly scrapped this idea, and looked to using the worm gear that we had learned about during the manipulation lecture. Though we would later use the worm gear for our final product, we were unable to put the gear train together and proceeded to scrap this idea as well.

After having wasted a week without any significant progress, we turned to the lecture slides for guidance. We discovered that our initial idea had failed due to insubstantial torque (as we originally guessed), and that we could calculate torque using the values determined from the number of teeth per gear. Thus, we constructed a gear train, where multiple 12-tooth gears connected to a 60-tooth gear. However, this too was not going to work, since we had completely ignored the information about idler gears. The small gears were not contributing to torque, because they were positioned between the large gears. This is illustrated well in this diagram:



Soon after this discovery, we made a tremendous leap forward and designed a proper gear train with 25x torque.

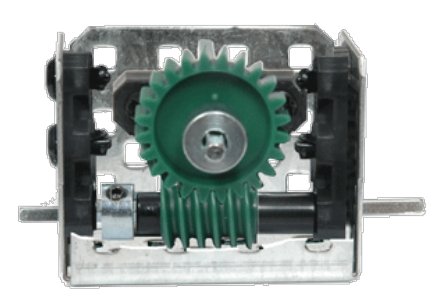
This gear train was able to lift the metal bar that would later become the upper arm. In addition, it was able to lift several large wheels attached to the end of the metal bar. Our professor expressed doubts that 25x torque would be enough to lift the entire arm once the robot was completed without damaging the gears and risking a hardware malfunction. Thus, we added one more 12-tooth to 60-tooth gear combination and increased our torque to 125x.

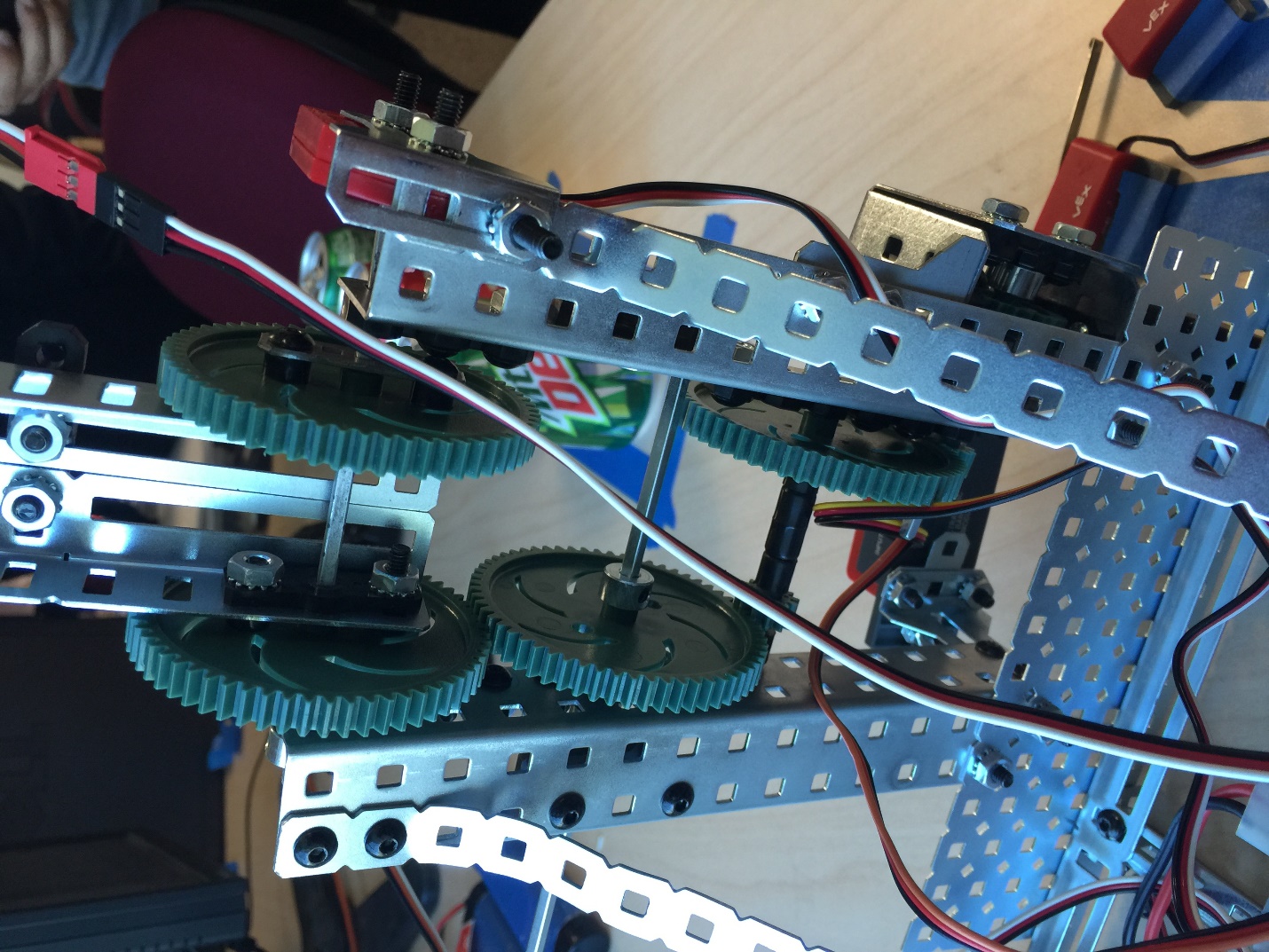
Now that we had the gear train, we had to attach a metal arm to the shoulder and add a potentiometer to take readings. We completed both of these steps at the same time, and made sure to set the potentiometer readings to a little over 0 (so as to avoid breaking the internal mechanism should motor push the arm too far) with the arm at the very bottom of the shoulders reach.

We then required hard-switches to prevent the arm from smashing into the tabletop. This switch came in the form of a bumper which was later removed to make way for soft-switches (entirely coded into the functions and not dependent on external hardware such as bumpers).

After we designed the rest of the arm, we found that the reach of the robot was too short. As mentioned under the ‘base’ section, we decided to switch the entire arm to the other side of the robot. This caused issues with the back of the arm hitting the gear train during motion. Thus, we had to make sizable changes to the gear train and add idler gears to increase the distance between the back of the arm and the topmost gears.

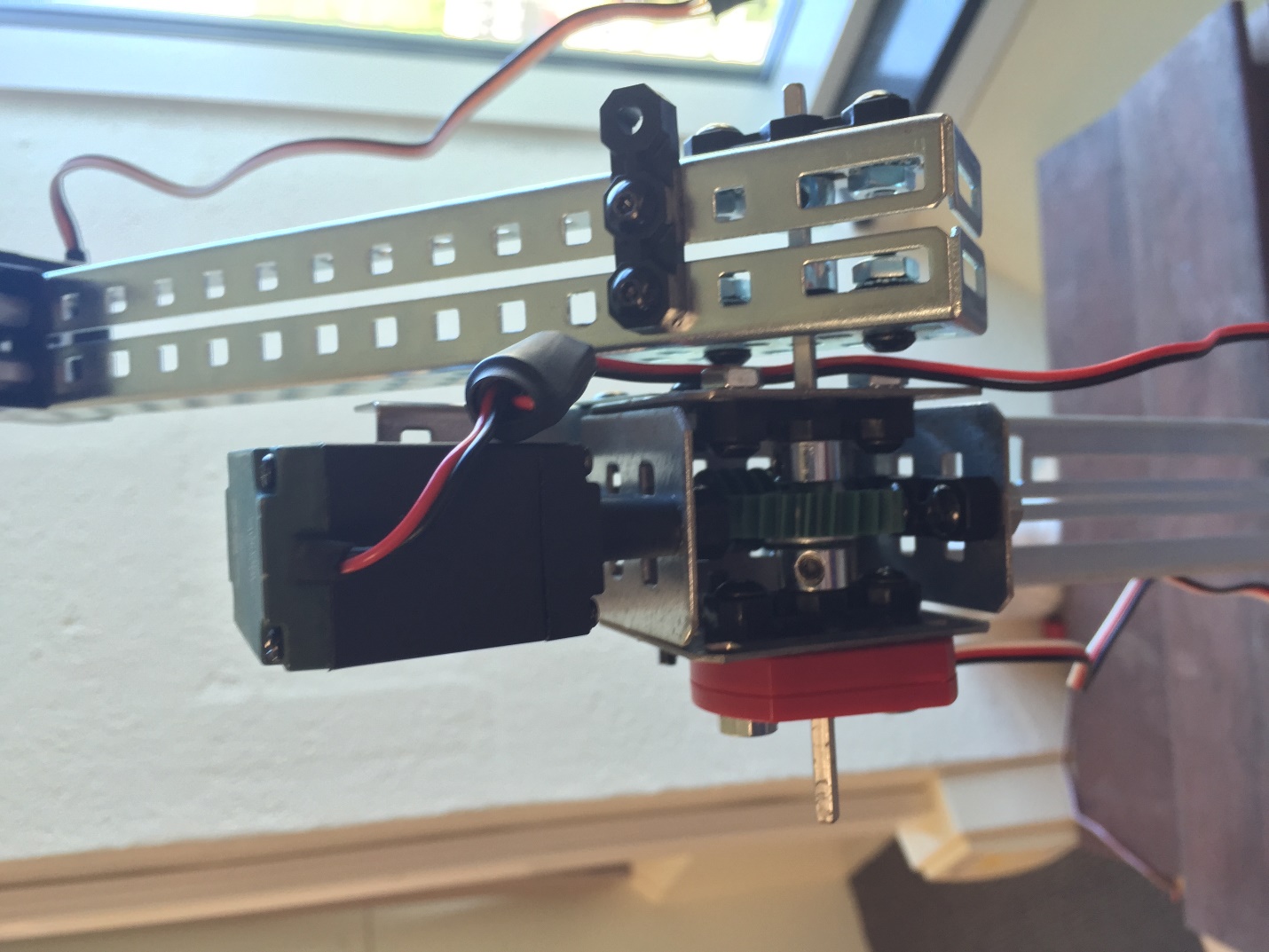
The most sizeable change to the shoulder came towards the end of our robot construction. We found that we were not able to prevent the arm from touching the gear train without sizably reducing the torque. Thus, we decided to radically switch our implementation and return to the worm gear. We attached our motor to the bottom of the gear assembly as shown and then ran a rod through the inner chassis to the shoulder gear train. Here we added another 5x torque by connecting a 12-tooth gear to a 64-tooth gear. Thus, the final torque in the shoulder gear train came out to 24x (worm gear) \* 5x (lower compound gear) \* 5x (upper compound gear) = 600x torque. This gear train proved extremely stable and did not struggle with any of the lifting process. The only minor drawback was speed, but I will cover that in the performance evaluation below.





1. The Elbow

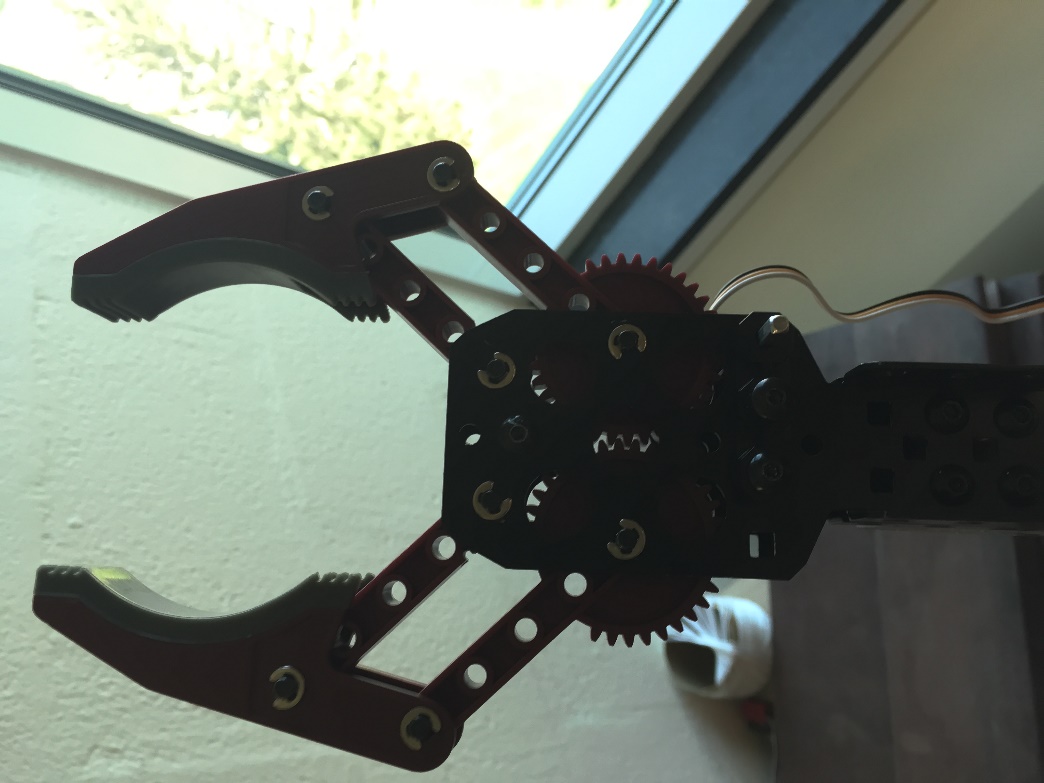
In comparison to the shoulder, the elbow was fairly easy to assemble. Like the shoulder, it required a gear train and a potentiometer, along with a second metal rod to constitute the forearm. However, unlike the shoulder, it only needed to carry a small amount of weight, and the gear train to have a large degree of error in its motion. Also unlike our shoulder, we were able to design the elbow in one shot, and we did not change the design after this first implementation. We used the same worm gear setup that is shown above for the shoulder, but attached a potentiometer the other end of the 36-tooth reinforced gear. This potentiometer was set with a value above zero for its resting point, however, it stopped working a few weeks into our project, and required replacement.



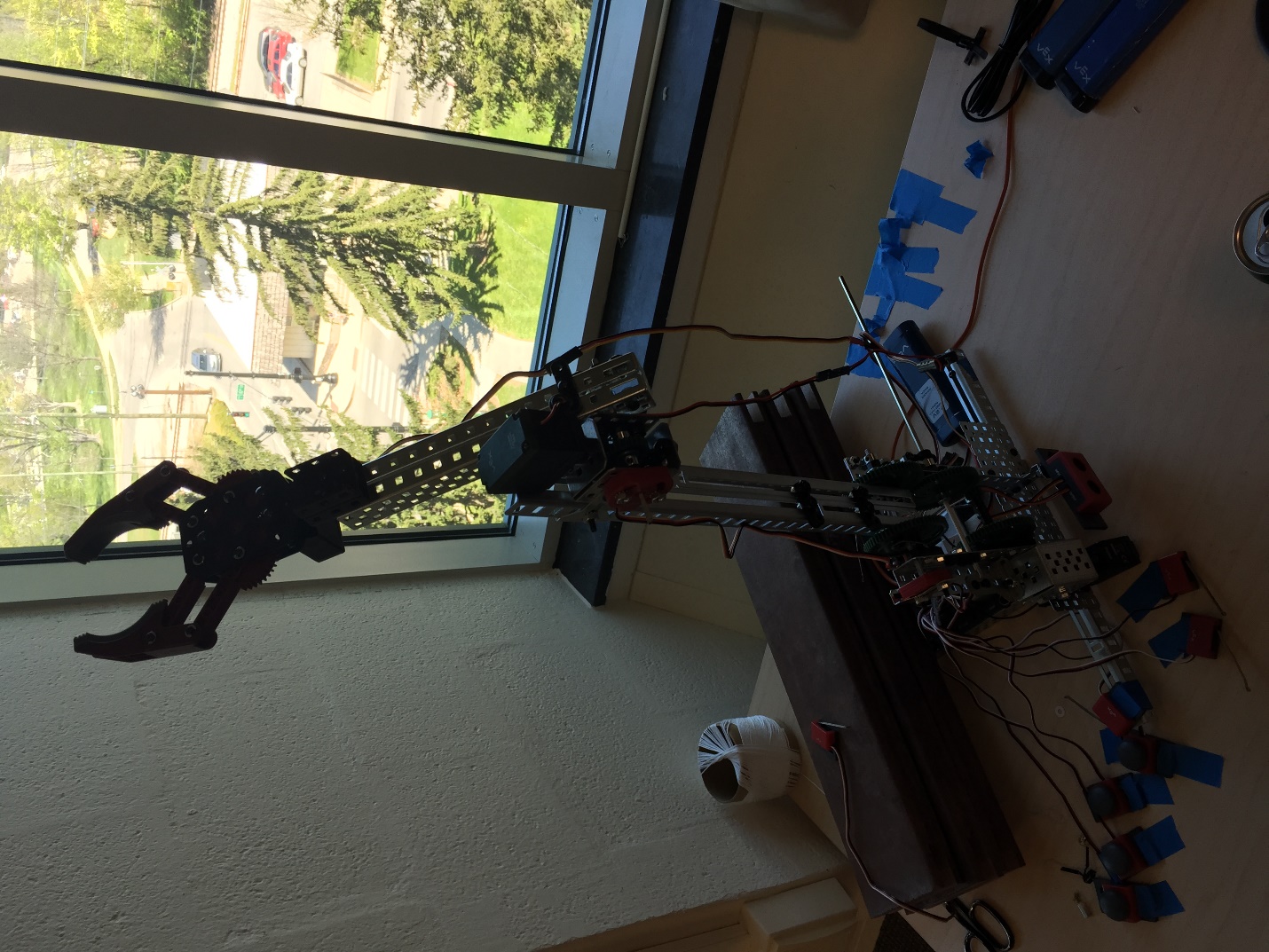
1. The Claw

Though seemingly simple, the claw ended up being the second most complicated and painstaking piece of the manipulator to plan and assemble. For starters, the claw itself did not respond well to the servo motor attached at its base. We had initially attached a regular motor to the claw, but due to the massive weight of the motor, we opted for a lighter servo motor. The problem with this approach was that the servo motor had to be set to its maximum value and then attached to the claw while it was in a wide enough position to fit around the cans, but not wide enough to prevent the closing value of the servo motor from grabbing the cans. In addition, every time the servo was attached, multiple washers had to be fit onto the gear-rod to prevent error in the motion of the gear train.

Even after we originally assembled our claw and repeatedly reassembled it to retain the perfect servo position for picking up cans, we had the occasional hiccup that caused major delays in our project. For instance, the servo stopped responding during the second to last week of our project. We thus had to remove the servo and redo all of the fine motor adjustments that we had painstakingly worked on for days. During this time, we also replaced the spring loaded claw with a non-spring loaded claw. We found no difference in performance between the two claws, but we were told that students had run into issues with the spring loaded claw. Since every group had required a switch, we decided to make the replacement while we were already working on the claw. I believe this adjustment saved us another days work during the last week.



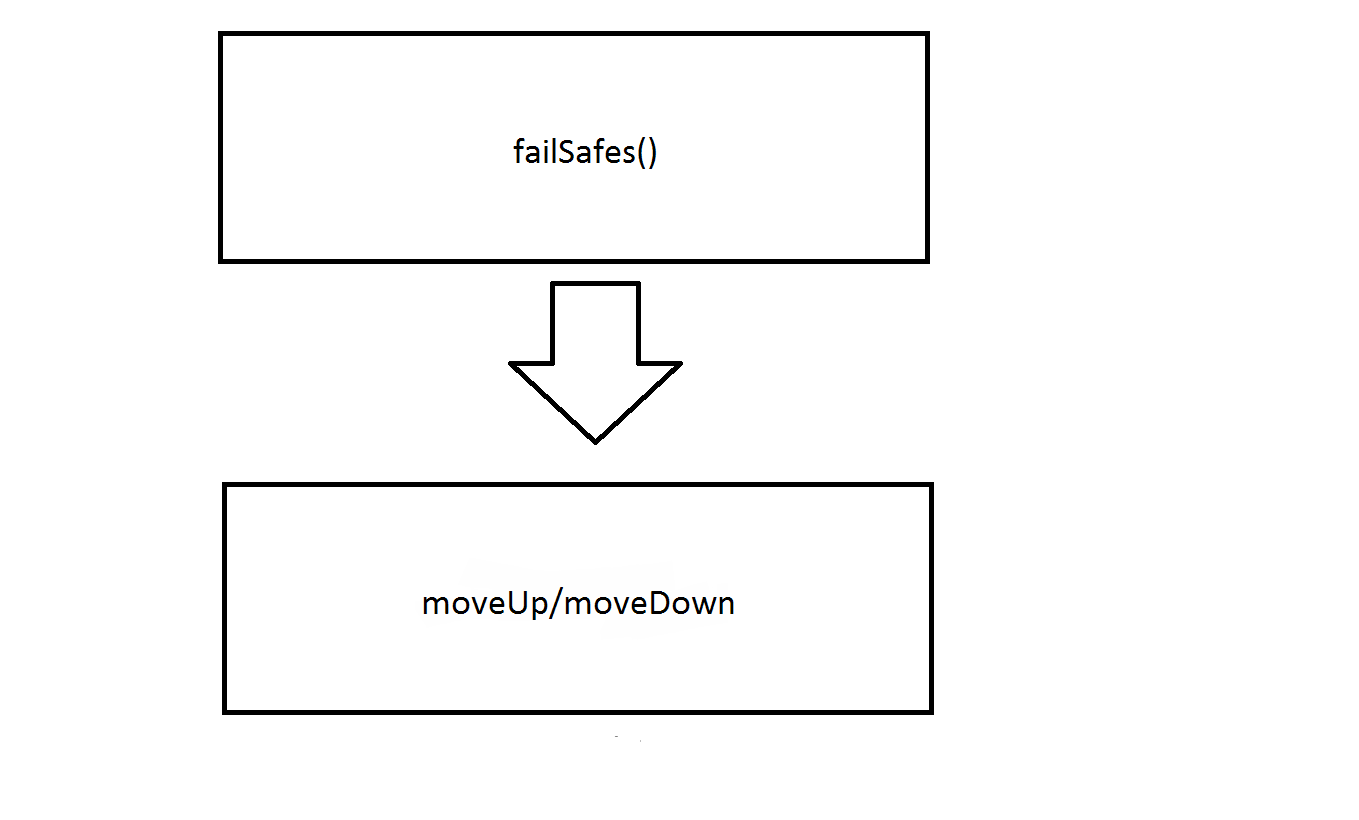
As we completed each piece of our robot, we assembled it onto the last piece to construct the final build. There are some notable exceptions to this. While the shoulder gear train was always connected to the base, we had to remove the elbow from the robot at least once to connect the claw to the arm of the robot. We found that the upper arm and forearm were easier to work on separately and connect to the main robot after we finished.



As you will notice in the picture above, seven switches were attached to the robot and taped to the desk beside it. I will talk more about these switches during the ‘Software Design’ section of this report. The actual hardware installation was very easy, and must be mentioned only to complete my step-by-step reproduction of the robot.

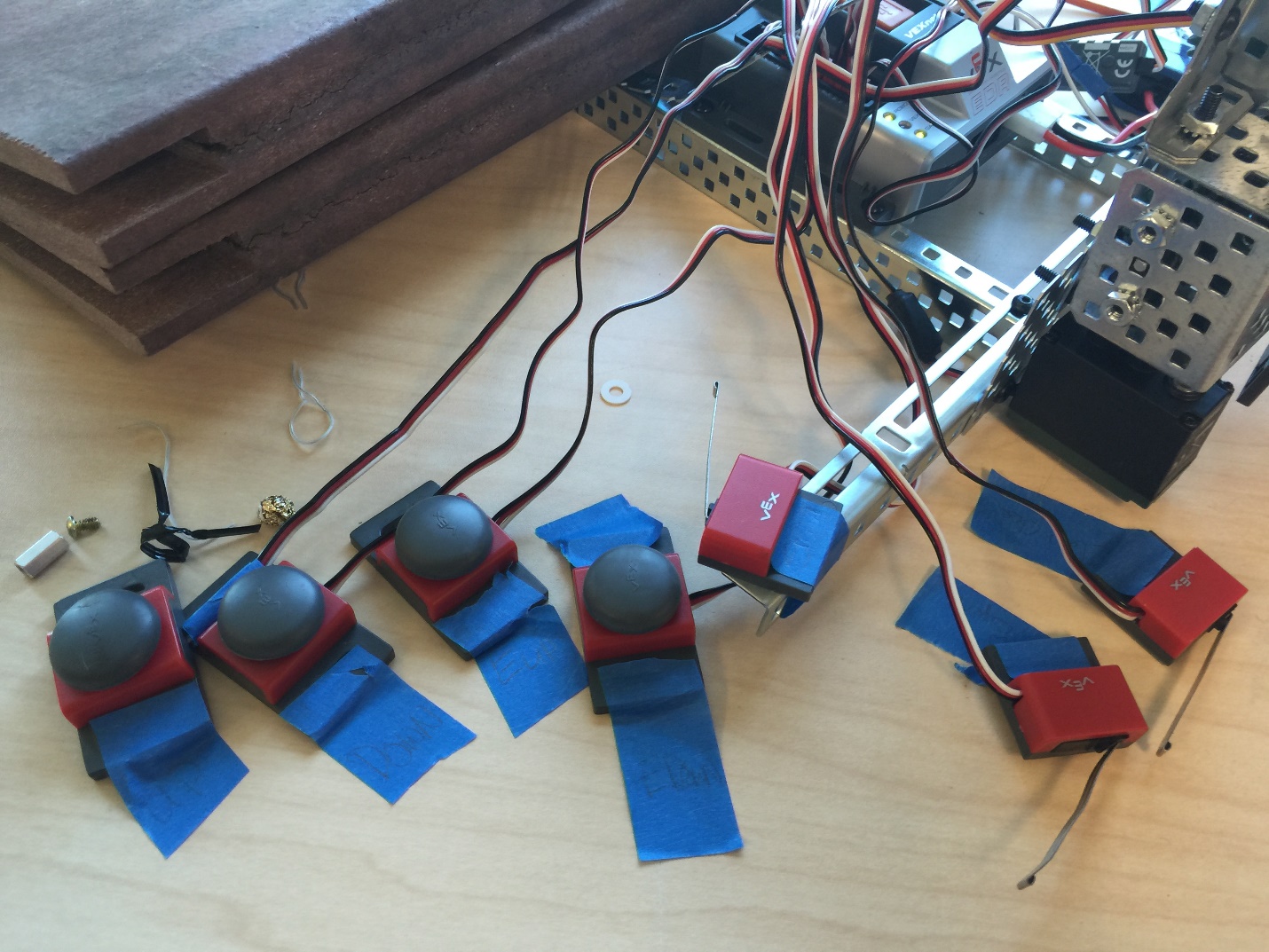
III. Software Design

The software design was very simple at the start of our project, and became far more complicated during the last week of class. This is usually how our algorithm and code development has proceeded for our previous projects as well. Our first piece of code was used for testing the motion of the shoulder, and contained three methods. A failsafe, which ended the program as soon as the limit switch was triggered or we hit an emergency stop bumper attached the side of the robot; a moveUp function, which lifted the arm on the shoulder joint; and a moveDown function that lowered the arm on the shoulder joint. The moveUp and moveDown functions had soft-coded stopping values based on potentiometer readings we had taken. The values were deemed the safest places for the shoulder to stop before any risk of injure could arise.



The code stayed in this state for the first few weeks disregarding the minor addition of two more functions: elbowUp and elbowDown, which as their name stated, raised and lowered the forearm on the elbow joint. However, when we decided to switch the entire arm to face the other side of the robot, we were required to redo many of the values hardcoded into the first implementation, and remove the limit switch. At this point, we removed the failsafes program and relied entirely on the emergency “stop” bumper and the soft-coded stop values based on the potentiometer.

Once we had assembled the entire robot, we needed to find values for the shoulder and elbow potentiometers to create empirical formulas for the elbow and shoulder motion based on sonar readings. At this point, I created a method ‘switches’ that allowed me to attach seven bumpers to the base of my robot and control each degree of motion using a bumper. I could control the shoulders up and down motion, the elbows up and down motion, and the claws position. The last bumper continued to function as an emergency stop.



At this point, I used a single coke can, and measured the shoulder and elbow potentiometer reading for every 10 points on the sonar starting at 200 and continuing to 436. The value for 436 was arbitrary, and I chose to stop recording points because I believed I had enough data to empirically calculate the amount of motion required for both the shoulder and elbow. I compiled the data into an excel spreadsheet, and found the linear trendlines for both the shoulder and elbow motion in comparison to the sonar readings. I then created a finalPrgm function in my code, which was run by itself in the main method. This function took a reading from the sonar and set values for the elbow and shoulder potentiometers based on the empirical formula I derived. It then opened the claw, moved the shoulder down into position, moved the elbow down into position, picked up the can by closing the claw, and then lifted the elbow and shoulder until it was back at the top of the robot and dropped the can. This program used the other functions I had defined, and required some modification of the original functions. I allowed moveDown and elbowDown to contain a float parameter in order to set the softcoded limits to the calculated values.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Pot | Shoulder | Elbow | | 200 | 1184 | 2500 | | 209 | 1184 | 2490 | | 221 | 1181 | 2450 | | 229 | 1181 | 2427 | | 243 | 1181 | 2400 | | 251 | 1170 | 2410 | | 264 | 1160 | 2290 | | 272 | 1145 | 2258 | | 284 | 1135 | 2230 | | 293 | 1126 | 2180 | | 303 | 1128 | 2134 | | 313 | 1115 | 2120 | | 331 | 1120 | 2100 | | 343 | 1084 | 2008 | | 356 | 1083 | 1985 | | 366 | 1069 | 1942 | | 380 | 1052 | 1875 | | 395 | 1020 | 1815 | | 411 | 1013 | 1780 | | 421 | 990 | 1724 | | 436 | 966 | 1640 | |

After a few trial runs of my program, I found that moving the elbow before the shoulder prevented the cans from being knocked over by the claw, and I thus switched the order of the operations inside the finalPrgm function on both the way down and the way up.

The last change I made to the code accounted for pushed cans. I created an ‘if’ statement inside the empirical calculation that set the elbow and shoulder potentiometer softcoded stop values to pre-defined, tested values if the sonar read a distance above 436. I know that the cans would not be pushed very far, so I created an ‘if’ statement that would account for cans outside the taped off area.

IV. Performance Evaluation

Our robot performed admirably during its final demonstration. In fact, we were able to stop working two days in advance. Thanks to switching the arm from what was once the front of the robot, we were able to increase our reach by a huge amount. Our robot could have picked up far more than three cans, though I am unsure if our empirical formula would have been able to pick up more than three cans without some tweaking.

Due to a couple of unfortunate potentiometer drawbacks, our robot did not have the reach that we would have expected. Thus, it was unable to drop cans far behind it, and had to settle for dropping cans just beyond its center. If I could replace one piece of Vex hardware on our manipulator, it would have been the potentiometers. This being said, I am unsure what hardware I would use to replace them. Our instructor mentioned a piece of hardware that could spin 360-degrees without breaking, but if the hardware was heavy then it might have negatively affected our robots performance.

The kinematics on our robot, while effective, were not entirely accurate. Several of our cans were grabbed just by the tip of our claw or grabbed at very unstable angles. I can only concur that the kinematics were less refined than what I might have hoped. I believe that taking more data at a smaller sensor-value interval would have helped refine the data and procure more accurate kinematics. In addition, our robot was entirely reliant on a single reading from the sonar sensor before moving. If the light reflected poorly off of one of the cans, the robot would grab at nothing. Thus, it might have been a good idea to implement a method that took several sonar readings and averaged the values to find a more accurate position of the can.

V. Conclusion

This lab was incredibly challenging, and yet incredibly fun. I thoroughly enjoyed working with complex gearing and building something akin to a human ligament. In addition, I worked on more hardware during this lab than the previous labs, and I grew to appreciate the fine art of getting everything to work without investing too much time. As the software developer, I would often continually optimize the code in order to make up for inadequacies I found in the hardware. However, as the hardware designer and assembler, I found that many of those “inadequacies” were worthwhile in order to save days of effort. I have definitely acquire an appreciation for the convergence of software and hardware that I was unable to find in any of my other computer science classes.

I am very pleased with the quality of this class and the projects we have been assigned. This final project has given me an appreciation of robotics that I could not have found otherwise. I still wish that I had acquired a more balanced team, but at the same time, my group mates were very amicable and responded to my emails in a timely manner. I hope to continue with robotics in the future.

// sensor imports

#pragma config(Sensor, in7, elbowpot, sensorPotentiometer)

#pragma config(Sensor, in8, shoulderpot, sensorPotentiometer)

#pragma config(Sensor, dgtl1, up, sensorTouch)

#pragma config(Sensor, dgtl2, down, sensorTouch)

#pragma config(Sensor, dgtl3, eup, sensorTouch)

#pragma config(Sensor, dgtl4, edown, sensorTouch)

#pragma config(Sensor, dgtl5, stopit, sensorTouch)

#pragma config(Sensor, dgtl7, sonar, sensorSONAR\_mm)

#pragma config(Sensor, dgtl9, clawopen, sensorTouch)

#pragma config(Sensor, dgtl10, clawclose, sensorTouch)

#pragma config(Sensor, dgtl11, limit, sensorTouch)

#pragma config(Motor, port2, shoulder, tmotorVex393, openLoop)

#pragma config(Motor, port7, elbow, tmotorVex393, openLoop)

#pragma config(Motor, port9, claw, tmotorServoStandard, openLoop)

//\*!!Code automatically generated by 'ROBOTC' configuration wizard !!\*//

short StopAll = false; // an emergency stopping mechanism that was not implemented in the finalPrgm()

void clawOpen() // opened the claw using the servo motor

{

motor[claw] = 127;

}

void clawClose() // closed the claw using the servo motor

{

motor[claw] = -50;

}

void elbowDown(float x) // moved the elbow down until the potentiometer crossed the value x

{

while(SensorValue[elbowpot] < x) {

motor[elbow] = 30;

}

motor[elbow] = 0;

}

void elbowUp() // moved the elbow up until it was laying on top of the upper arm

{

while(SensorValue[elbowpot] > 20) {

motor[elbow] = -30;

}

motor[elbow] = 0;

}

void moveDown(float y) // moved the shoulder down until the potentiometer crossed the value y

{

while(SensorValue[shoulderpot] > y){

motor[shoulder] = -100;

}

motor[shoulder] = 0;

}

void moveUp() // moved the shoulder up until the claw had crossed the center line

{

while(SensorValue[shoulderpot] < 1700){

motor[shoulder] = 100;

}

motor[shoulder] = 0;

}

void finalPrgm() // the final program that was used to pick up 3 cans

{

for (int i = 0; i < 3; i++) { //run the program 3 times (can be changed to accommodate more cans)

int n = SensorValue[sonar]; // initial sonar reading

float x = -3.6441 \* n + 3262.9; // the elbow potentiometer value

float y = -.917 \* n + 1393.7; // the shoulder potentiometer value

if (n > 436) { // if a can has been bumped out of the taped-down grid

x = 1440; // values I found to work best

y = 893; // values I found to work best

}

clawOpen(); // open the claw

wait1Msec(500); // short waits to allow the computer time to think

elbowDown(x); // move the elbow first so as to avoid bumping cans

moveDown(y); // then move the shoulder to the empirical value

wait1Msec(500);

clawClose(); // close the claw around the can

wait1Msec(500);

moveUp(); // move the shoulder first to avoid bumping other cans

elbowUp(); // move the elbow so that the can crosses the center of the robot

wait1Msec(500);

clawOpen(); // drop the can

wait1Msec(500); // small wait time before running the next operation

}

}

void failSafes() // hard-coded failsafes. Not in use in the finalPrgm()

{

if(SensorValue[limit] == 1 || SensorValue[stopit] == 1){

motor[shoulder] =0;

wait1Msec(100);

StopAll = true;

}

}

void switches() { // run while(true) and allowed the switches to control the robot

if (SensorValue[down] == 1){ // equivalent of moveDown()

motor[shoulder] = -100;

}

if (SensorValue[up] == 1) { // equivalent of moveUp()

motor[shoulder] = 100;

}

if (SensorValue[edown] == 1) { // equivalent of elbowDown()

motor[elbow] = 30; // slower values to accommodate for tiny adjustments

}

if (SensorValue[eup] == 1) { // equivalent of elbowUp()

motor[elbow] = -30; // slower values to accommodate for tiny adjustments

}

if (SensorValue[clawopen] == 1) { // equivalent of clawOpen()

motor[claw] = 127;

}

if (SensorValue[clawclose] == 1) { // equivalent of clawClose()

motor[claw] = -50;

}

if (SensorValue[stopit] == 1) { // hard-coded stop button

motor[shoulder] = 0;

motor[elbow] = 0;

}

}

task main(){ // main method

finalPrgm(); // the final program to lift cans

} // end main