2022 VEX Project
MechEng 201

Joseph Kaw jkaw679

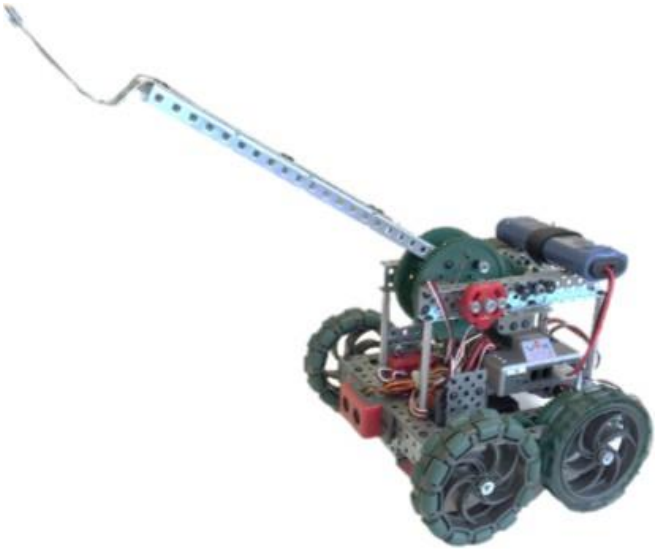


Table of Contents

Introduction and task

Due to our great autonomous solution in the small storage room, the dairy industry cooperative has now asked us to design another autonomos solution for the large storage room. This issue was raised due to the increasing cost of labour and by developing a solution to the payload using a scaled down robot we are enabled to essentially resemble the features in the real world.

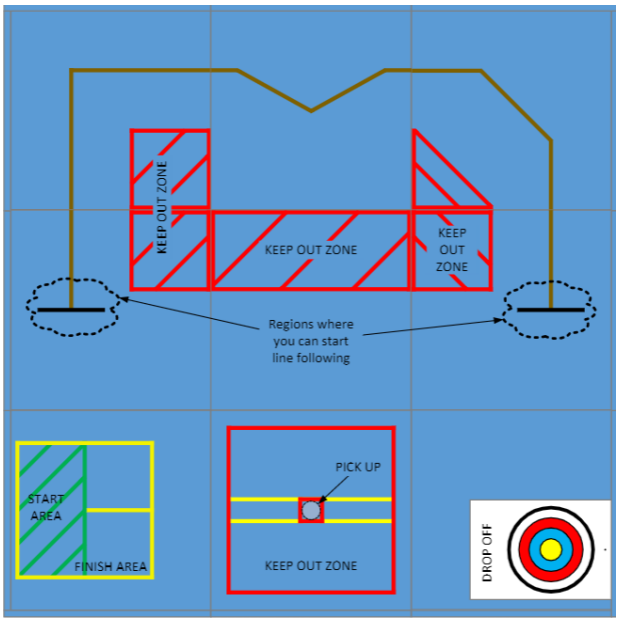


Figure 1: Large Warehouse Layout

The vex robot which represents an automated forklift is needed to initially leave its charging area which is our start zone. The payload, which is fragile, represents a weighted can between the yellow lines, and will need to be picked up and transported to the drop off zone (as close to the yellow circular region as possible). It will then return back to the charging area.

In addition to this the robot must start with its arm raised, it must not leave the warehouse walls, which in this diagram would be the outline of the board and also must not touch the stock shelves which are the red zones. In order to ensure the robot doesn't have a flat battery it will need to do the loop within 120s as if it exceeds this bound it will cause a halt in the production chain to go and retrieve the robot which is inefficient.

Our Solution - logic of our programme

Before we started our solution it was important for us to recognise the three ways the robot was going to move around the map. These would include Sonar-Based navigation, Encoder-Based Navigation and Line following navigation. (See Appendix B, figure 3)

Sonar-Based navigation

We used a function called *sonicDrive();* to drive straight using an ultrasonic sensor and a p controller to a specified distance in mm away from the obstacle. It contains four inputs and initialises the power as an integer with a value. It also initialised error as an integer to equalise the motors and finally the variable u as an integer which is the output of the p controller. Sets a value of range as distance away from an obstacle and drives straight until it detects an object in front of it. Uses the *drivestraight();* function until it's at the desired distance away from the obstacle. (See appendix A figure1 for flowchart)

Encoder-Based Navigation

The function *armtodeg()*; takes the arm encoder count and then changes it to a degree of the current position. After using the compass app we were able to see that the maximum angle the arm could be raised was 47 degree for our robot 8. We knew the product of 21 and 240.88 accounted for 360 degrees of arm rotation, which allowed us to return to the current position. (See appendix A figure 2 for flowchart)

The function *wheeltomilli();* takes the wheel encoder count and converts them to travelled distance in mm. As we know that there are 1800 counts for the product of 3, 103 and pi mm travelled, we can use this to return the distance travelled by the wheels. (See appendix A figure 3 for flowchart)

The function *moveArm();* rotates the arm to a specified angle in degrees using a P controller. We initialise values, pos : position, pwr : power, u : the output of p controller and error as integers. It initially sets the initial arm position as the reference point using the encoder arm. The error will be the difference between the desired degree value imputed and the current position using the call of the *armtodeg();* function. While the power is fairly high it will continue to move the motor and uses a P controller to move the arm and ensure that u is being saturated. Once our robot's absolute value of the error is greater than 10 it will exit the loop and then the motor power in the arm will be zero. (See appendix A figure 4 for flowchart)

The function *driveStraight();* drives a specified distance in mm and then stops. It initialises values pwr : power, error2 : equalises power between the motors, u1 : the output of the p controller for the distance as integers and sets the starting point as a reference point using the left and right encoders.After it also initialises pos : power, u2 : The output of the p controller for the motors as integers, error1 : difference of target distance and position e and kps as float type. Uses a while loop for when motor power is still fairly high, it calls function *wheeltomilli();* to help obtain error1. If the power output is not saturated ,the pi controller integrates and if it is outside the bounds it will be saturated. The error for the wheel power imbalance is used by finding the difference in the sensor for the right and the left which will be needed in obtaining u2. Then it will set the motor power using u2 and will finally exit the loop once the error is below 10. Finally the left and right motors will be turned off. (See appendix A figure 5 for flowchart)

The function *turn();* rotates the robot to an angle inputed and then stops, a positive value results in a counter-clockwise turn. We set initial placement as reference using the right encoder and initialise variables, error : the difference between target degree and current degree, position : current position and e : integrated error as float variables. In addition, initialised pwr: power of motor and angleValue : desirred arc length to be swept as integers. One of the keys for this function is ensuring we convert the angle to arc length next. It will loop and call the function *wheeltomilli();* to help obtain position and set error as difference between distance travelled by wheel and the desired arc length. It will then set u as the product of kp and the error added to the product of ki and the error. We then check to see if the absolute value of u is smaller than 50 and if it is we continue to add the errors. After saturating the power it turns the robot counterclockwise for positive values and clockwise for negative values. It will exit the loop when error is within tolerance and turn the motor power off. (See appendix A figure 6 for flowchart)

Line-Following Navigation

The function *linefollow();* allows the robot to continue moving along the brown line and stop when it senses the colours black. To start with initialises the variable i as 1 and error which ensures the robot is driving straight, both as integers. The value of the sensor is all set to zero all initialised as integers. The output of the P controller(u), the k value for the *drivestraight();* both initialised as float variables. We use data logging to find the lower and upper limit of brown and the lower value of black and initialise those three to variables as integers. After this we saturate the power and set the motor power, we delay for half a second then set the motor power to zero.

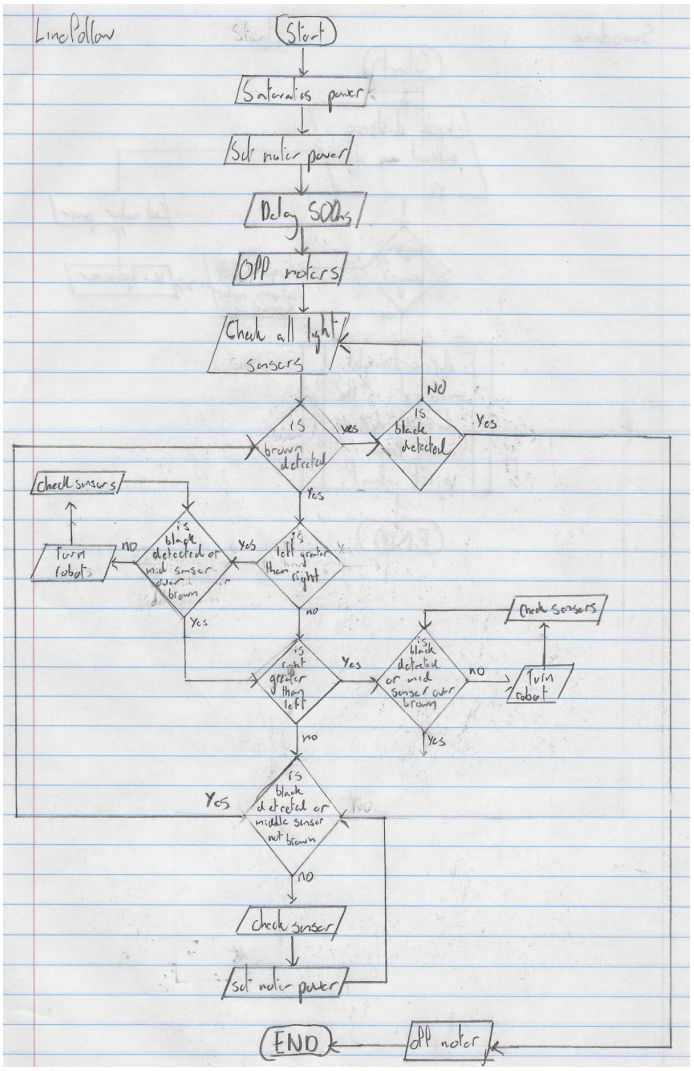


Figure 2 : linefollow(); flowchart

We have a loop while black is not detected and the light sensors are checked. Now it checks to see if brown is detected, if it is then checks to see if the left is greater than the right, if not will check if black is detected and if it will end the loop. After seeing left greater than right, it then checks to see if black is detected or the mid sensor is larger and if this is not true will turn the robot and check sensors, however if left is not greater than right then it checks if right is greater than the left. If right is greater than left it does the same process as before checking to see if the mid sensor is larger or if black is detected and if not will turn the robot and check sensors. Finally if right is not greater than black, it will check to see if black is detected or the if the middle sensor is not brown, if it's not then it will check the sensor and set the motor power, if so then it will loop back to the start checking to see if brown is detected and if not will exit the loop, turning b the motors off. (See appendix B figure 1 for zoomed flowchart)

Additional Functions

The function *waitForStart();* enables the user to make the robot start only when the button is pressed. The variable is initialised as an integer zero before the start button is pressed. The while function checks if a button is pressed. checking to see if the start button is pressed, it will check if it is true and sets value to one.It then rechecks sensor after 50 milliseconds and if start button is pressed the initialised value of 1 delays 0.5 secs and proceeds. (See appendix B figure 3 for flowchart)

Overall Structure For Solution

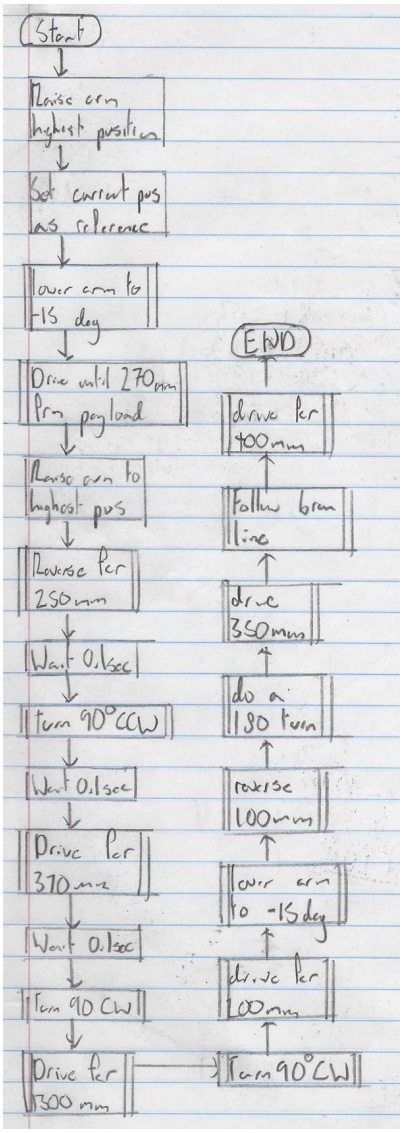


Figure 3 : Task flowchart

Using all the functions stated above it was necessary to arrange them in such a way we would be able to complete the task in hand for the dairy industry cooperative. First, wait for the user to click the start button. Ensures arm is at the highest position when starting in the charging zone. Set the current arm position to zero. Lower the arm to the height of the gap of the payload. Drives arms length away from the payload using sonar-based navigation. Then raises the arm with the payload on it to secure it. Drives backwards for 250mm, to ensure it doesn't hit any stock shelves and warehouse walls. Delay for 100m and then turn counterclockwise 90 degrees. It will delay for 100ms and then drive straight for 370mm. Again it will delay for100ms. turn clockwise 90 degrees, drive straight for 1300mm. And then turn clockwise 90 degrees to face the drop off zone. Then it proceeds to drive straight for 200mm. It will then lower the arm to drop the payload off in the yellow circular zone. Drive backwards for 100mm. Turn counterclockwise 180 degrees. Drives straight for 350mm. Follow the brown line till it reaches the black line and finally will drive straight till the finish zone. (See appendix B figure 2 for zoomed flowchart)

Reflection

For our reflection it was for the project it was necessary to break it into two subcategories being performance(how our robot performed on demonstration day) and recommendations(how our robot solution could be improved).

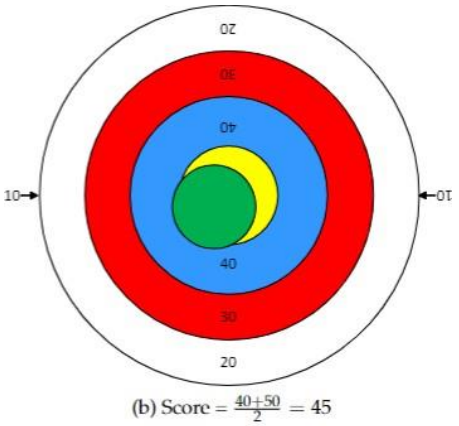
Performance Reflection

When it came to demonstration day we felt as though we were ready as we tested all our codes pre hand on RVW builder and our modified codes were working well on the virtual world. With the overall performance of the robot itself we were able to complete the start button task, reaching and lifting can task, line following, encoder based navigation and placement of the can. We knew that the two 5 minute blocks would run really quick in the heat of the moment and so we planned prehand how we would spend each of the five minutes which we did execute on the day to our credit. We thought our team dynamic was also really good as we were able to communicate throughout the project ensuring we were keeping up to date and at pace even when one person couldn't make the tutorial. We unfortunately ran out of time to complete a full run even though we had all the functions working at the end of the five minutes. We felt a bit upset when the previous group cut into about a minute of our time leaving us with four minutes instead of five for one block, which could have been the difference between getting a full run in but we understood it was the nature of the day. For our code itself we felt we used robust and efficient programmes to ensure we didnt have alot of calibration, we contained internal checks and

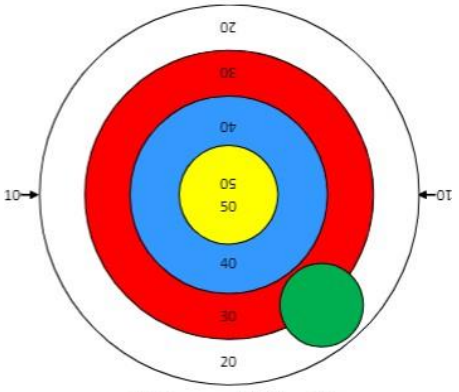
took care of the physical uncertainties in the robot. We achieved efficient codes with a good layout and variable name choice to ensure the demonstration was made easier.

Recommendations Reflection

We thought that although we did plan well in advance for the final demonstration day we should have planned the five minute demonstration better. When we did our run through for the first time round through we only received a placement mark of 15 and our line following failed. But what we did during that period between the first demonstration and the second demonstration was the majority spent on fixing our line following with a minor adjustment to the *driveStraight();* function to try to get a placement mark. We ended up completing our line following function the second time through but only managed to increase the placement mark by 10 obtaining 25 in the second demonstration. We felt that even if we focused purely on the placement mark we could have obtained even more marks than just fixing our line following. In saying this if we had time to complete our full run as we got our line following to work we may have potentially been able to obtain the time bonus and return bonus in this case.



(b) Score = $\frac{40+50}{2} = 45$



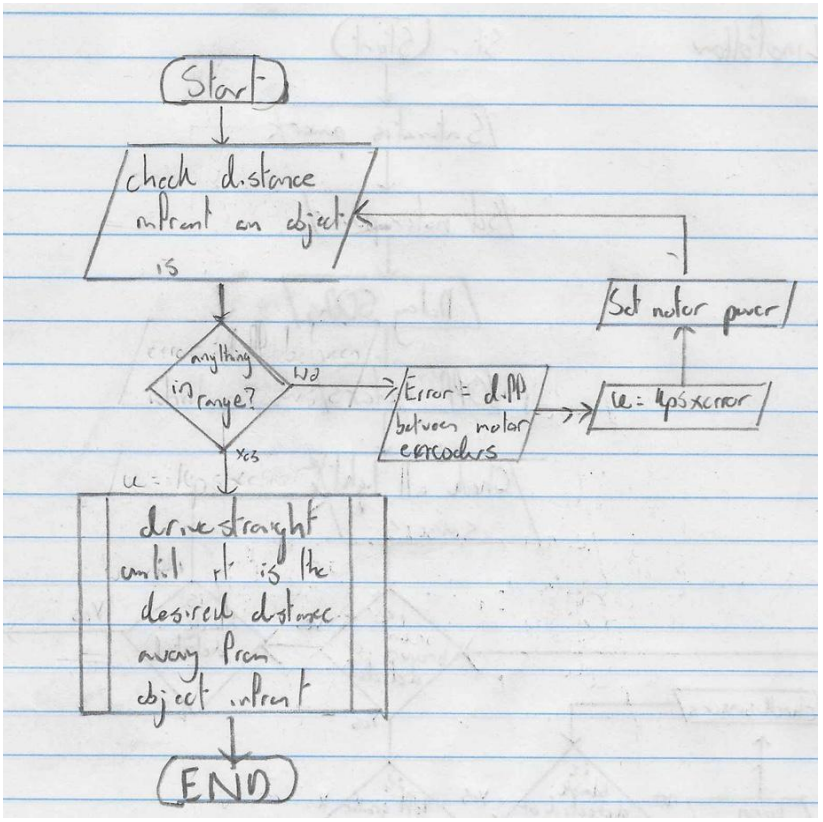
(d) Score = $\frac{30+20}{2} = 25$

Figure 4 : Improving Placement Values

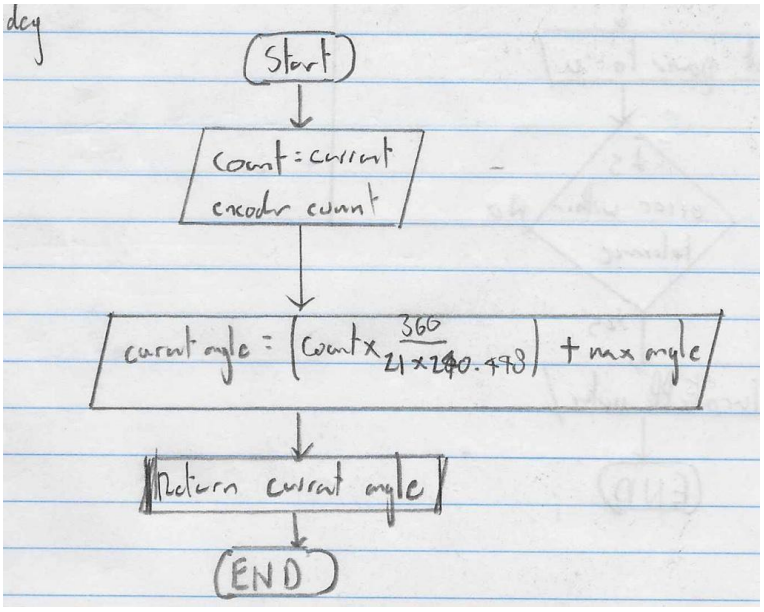
For both of us this was the first time we dived into the scope of anything related to robotics so not only was this a very fun experience with new technology it was also a great learning experience for both of us. I think the biggest takeaway from this is seeing our robot complete a task after we had spent long periods of time designing codes for it to complete and once we saw it perform the way we wanted it to, it was very rewarding. Due to our great autonomous solution in the small storage room it gave us the tools to help the dairy industry cooperative design another autonomos solution for the large storage room.

Appendix A:

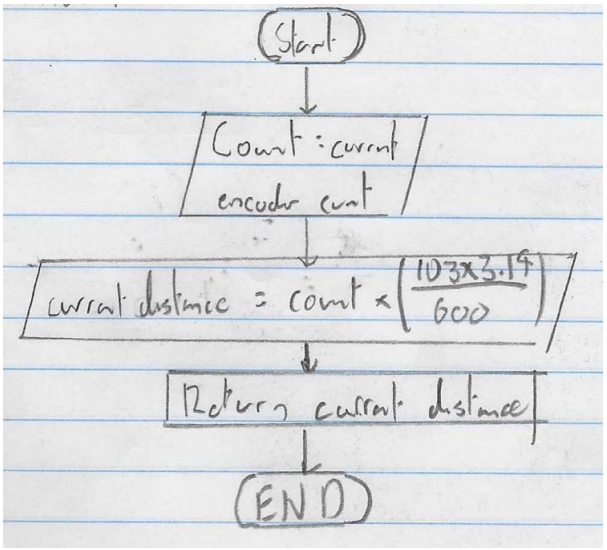
Appendix A Figure 1. ***sonicDrive();*** **function flowchart :** drive straight using an ultrasonic sensor and a p controller to a specified distance in mm away from the obstacle



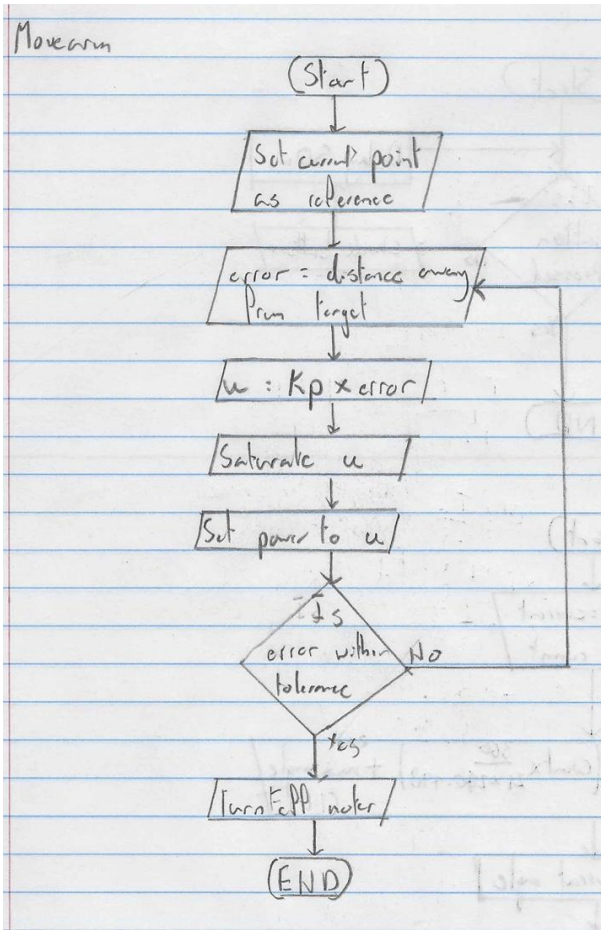
Appendix A Figure 2. ***armtodeg();*** **function flowchart :** takes the arm encoder count and then changes it to a degree of the current position.



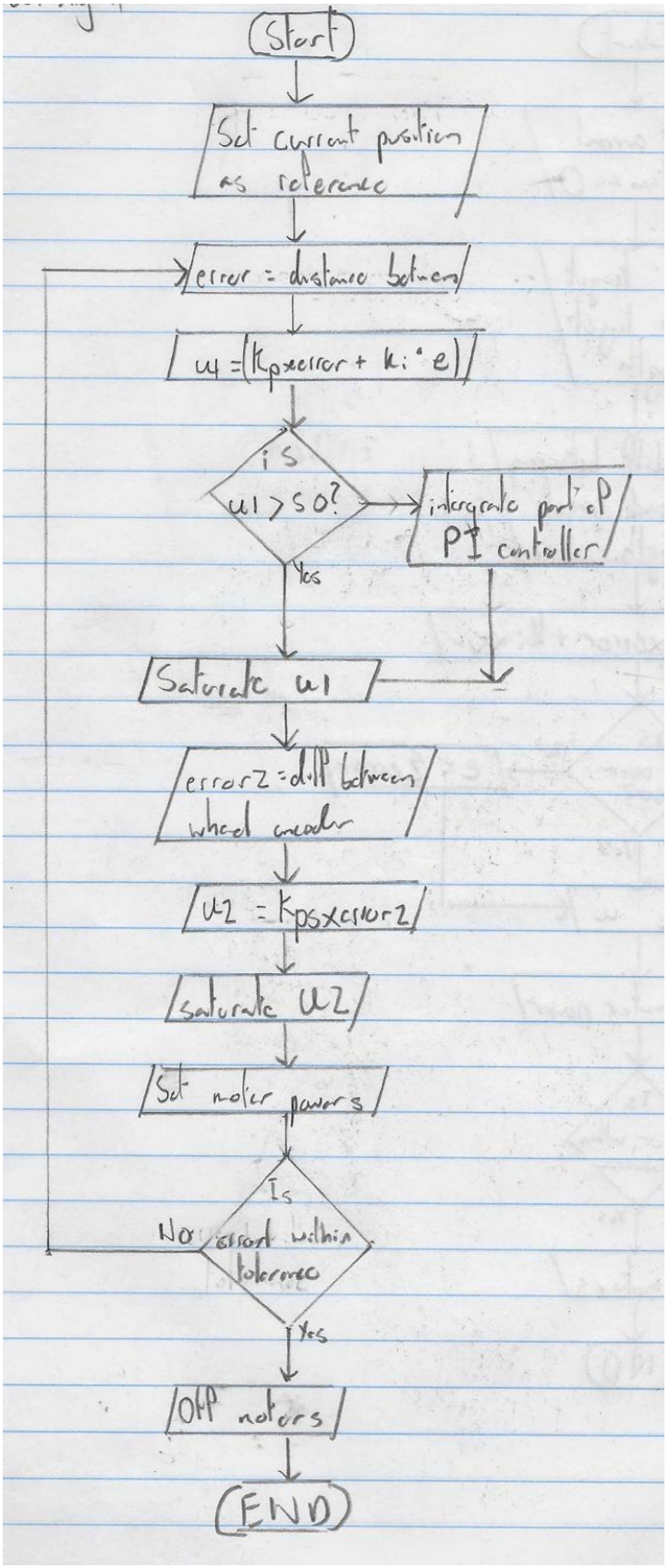
Appendix A Figure 3. *wheeltomilli();* function flowchart : takes the wheel encoder count and converts them to travelled distance in mm.



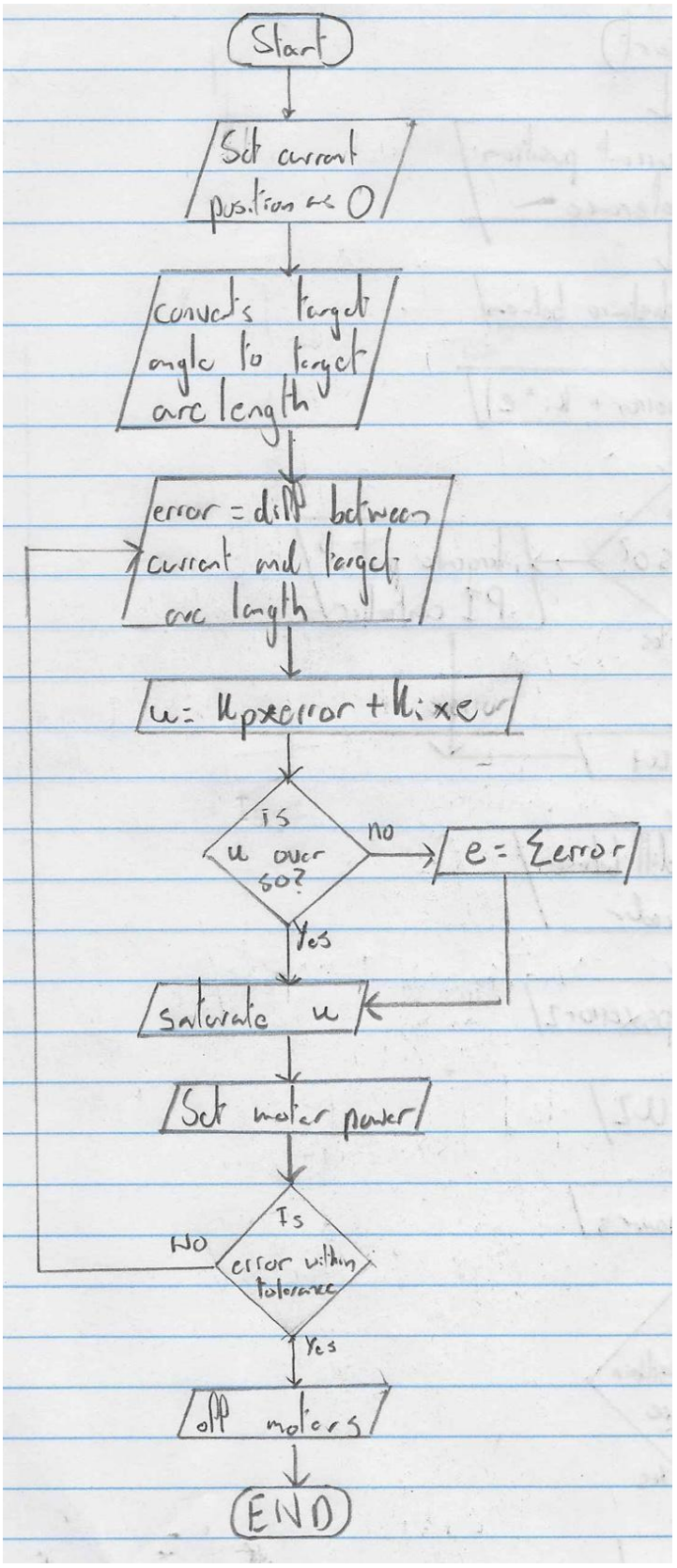
Appendix A Figure 4. *movearm();* function flowchart : rotates the arm to a specified angle in degrees using a P controller



Appendix A Figure 5. *driveStraight();* function flowchart : drives a specified distance in mm and then stops.

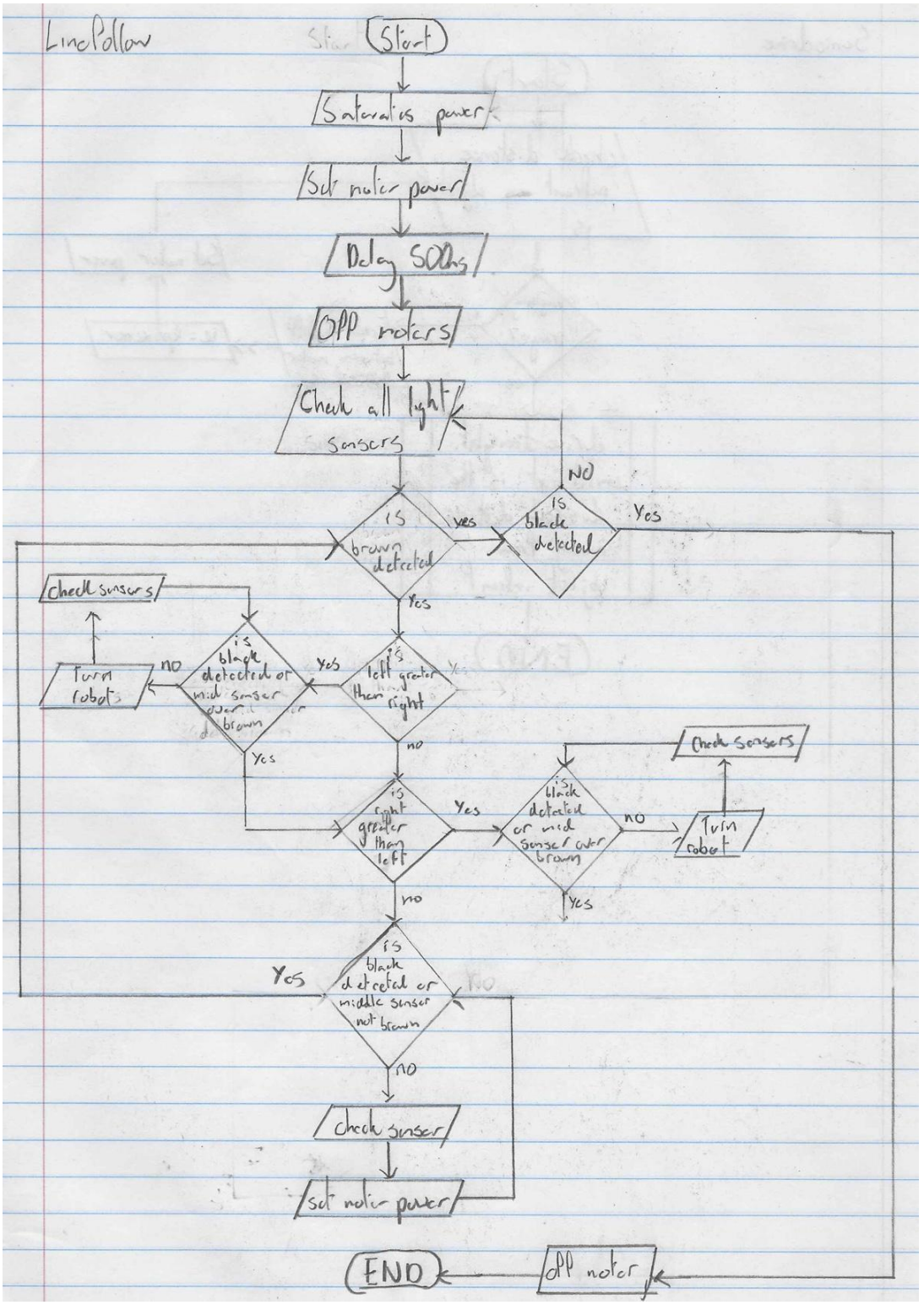


Appendix A Figure 6. *turn();* function flowchart : rotates the robot to an angle inputed and then stops, a positive value results in a counter-clockwise turn.

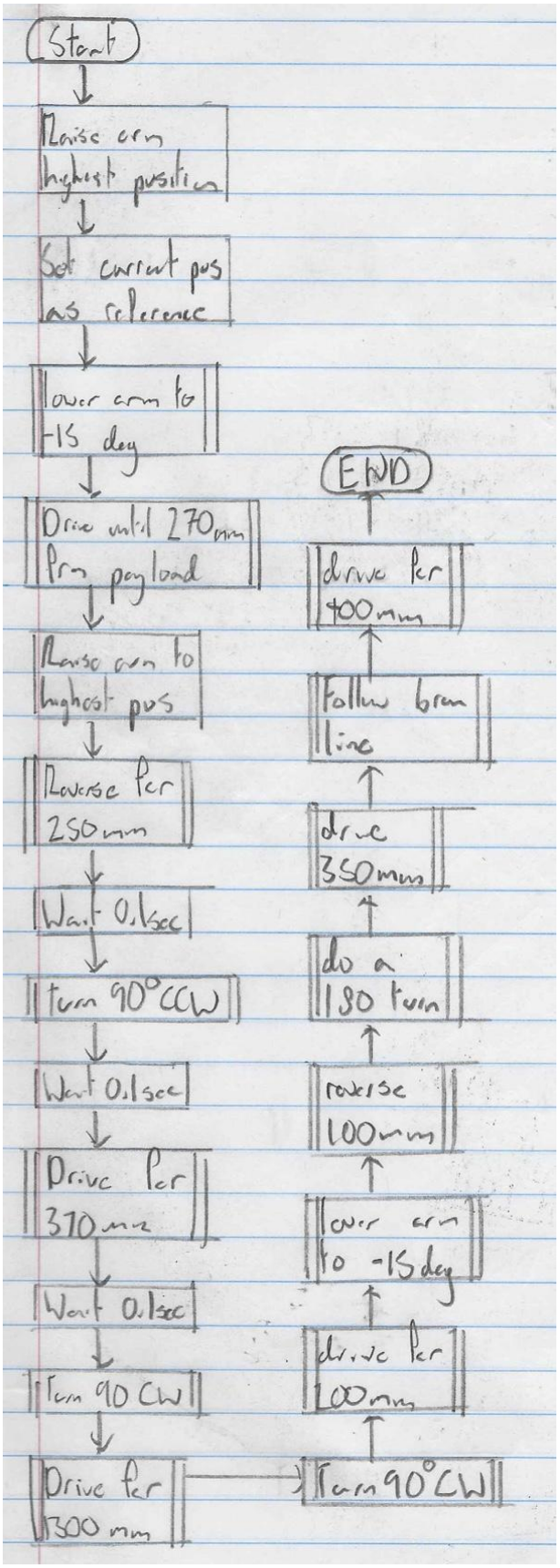


Appendix B:

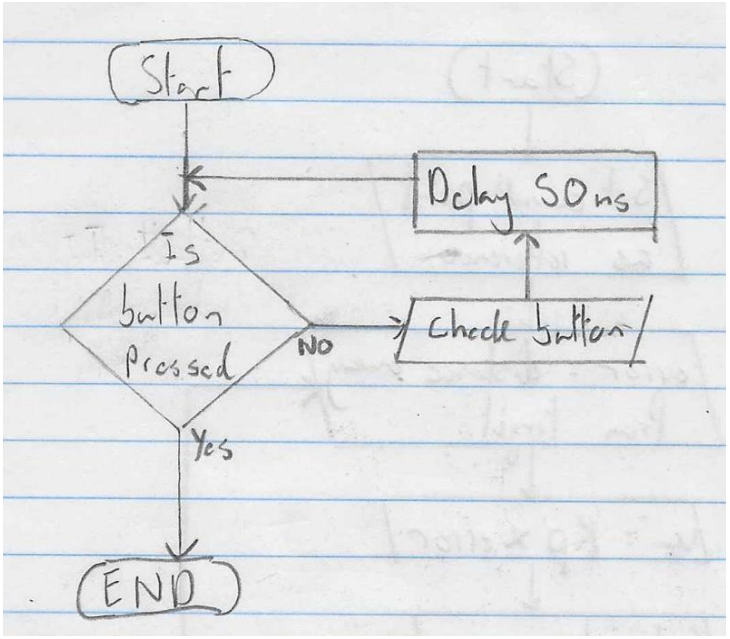
Appendix B Figure 1. *linefollow();* function flowchart : allows the robot to continue moving along the brown line and stop when it senses the colours black. (Zoomed version)



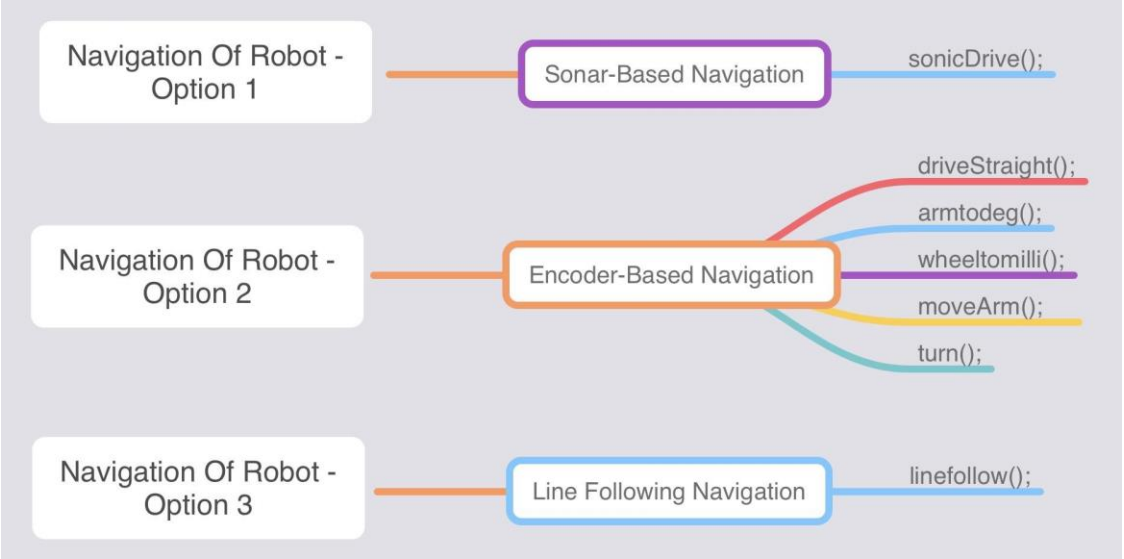
Appendix B Figure 2. Full solution flowchart : allows the robot to use all the different functions to get from the chagrin zone, pick up the package, drop the package whilst not hitting any shelves and then return back to the charging zone (zoomed version).



Appendix B Figure 3. *WaitforStart();* additional function flowchart : enables the user to make the robot start only when the button is pressed.



Appendix B Figure 4. Navigation of robot : divides into encoder-based navigation, sonar-based navigation and line following navigation.



Appendix B continued : list of figures in main report

By Joseph Kaw

```
#pragma config(I2C_Usage, I2C1, i2cSensors)
#pragma config(Sensor, in1,    _lightLeft,       sensorReflection)
#pragma config(Sensor, in2,    _lightMid,        sensorReflection)
#pragma config(Sensor, in3,    _lightRight,      sensorReflection)
#pragma config(Sensor, dgtl1,  _btnStop,         sensorTouch)
#pragma config(Sensor, dgtl2,  _btnStart,        sensorTouch)
#pragma config(Sensor, dgtl3,  _sonar,           sensorSONAR_mm)
#pragma config(Sensor, dgtl5,  _encRight,        sensorQuadEncoder)
#pragma config(Sensor, dgtl7,  _encLeft,         sensorQuadEncoder)
#pragma config(Sensor, dgtl9,  _LED_Right,       sensorDigitalOut)
#pragma config(Sensor, dgtl10, _LED_Left,        sensorDigitalOut)
#pragma config(Sensor, dgtl11, _armLimit_low,    sensorTouch)
#pragma config(Sensor, dgtl12, _armLimit_high,   sensorTouch)
#pragma config(Sensor, I2C_1,  _armEncoder,      sensorQuadEncoderOnI2CPort,    , AutoAssign )
#pragma config(Motor,  port2,           _motorArm,     tmotorVex269_MC29, openLoop, reversed, encoderPort, I2C_1)
#pragma config(Motor,  port7,           _motorRight,   tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port8,           _motorLeft,    tmotorVex393_MC29, openLoop, reversed)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

// -----------  DO NOT MODIFY anything ABOVE this line! ---------- //

//Authors : Joseph Kaw
#include "ProjectBackgroundProcesses2022.c"

/* Timers:
T_1 Free
T_2 Free
T_3 Free
T_4 Used by checkButtons() -- DO NOT USE
*/

// ---------------------- Defining physical robot parameters ----------------------------------------
// Update these numbers to match the physical robot (information found in the lab manual)
int drivingWheelDiameter = 103;     // diameter of the driving wheels [mm]
int robotWidth = 250;                           // width of the robot including the wheel thickness [mm]
int wheelWidth = 22;                            // width of the driving wheel [mm]
float drivingWheelRatio = 0.0;  // ratio of wheel shaft rotations to wheel encoder shaft rotations
float armRatio = 0.0;                       // ratio of arm shaft rotations to arm encoder shaft rotations
float wheelEncoderCounts = 360; // number of encoder ticks per 1 revolution of wheel encoder
float armEncoderCounts = 240.448;       // number of encoder ticks per 1 revolution of arm encoder
//
------------------------------------------------------------------------------------------------------------------------
-------------
```

```
42
43  //  ------------------------------------ Function Prototypes  ----------------------------------------
44  //Below listed are all our function prototypes that we used.
45  void waitForStart();
46  float armtodeg();
47  float wheeltomilli();
48  void moveArm(float deg,float kp);
49  void driveStraight(float distance,float kp, float ki);
50  void turn(float angle, float Kp, float Ki);
51  void sonicDrive(float distance,float kp, float ki, float kps);
52  void linefollow(int power);
53  // ------------------------------------------------------------------------------------------------------
54
55  // ************************** Main Task                                                                ➲
      ************************************
56  task main() {
57      //Background Tasks
58      startTask(checkArm);        // DO NOT DELETE THIS LINE
59      startTask(checkButtons);    // DO NOT DELETE THIS LINE
60      startTask(filterSonar);       // DO NOT DELETE THIS LINE
61      // DO NOT PUT YOUR CODE BEFORE THIS LINE!!!
62      //Waits for user to click the start button
63      waitForStart();
64      //Ensures arm is at the highest position when starting
65      armUp(30);
66      //Sets the current arm postion to zero
67      setSensor(EncoderArm,0);
68      //lowers it to the gap of the payload
69      moveArm(-15,6);
70      //drives arms length away from the payload
71      sonicDrive(270,2.5,2.5,0.2);
72      //raises arm with the payload on it
73      armUp(30);
74      //drives backwards for 250mm
75      driveStraight(-250,2.5,2.5);
76      //delays for 100ms
77      delay(100);
78      //turns counterclockwise 90 degrees
79      turn(90,2.5,2.5);
80      //delays by 100ms
81      delay(100);
82      //drives straight for 370mm
83      driveStraight(370,2.5,2.5);
84      //delays by 100ms
85      delay(100);
86      //turns clockwise 90 degrees
87      turn(-90,2.5,2.5);
88      //drives straight for 1300mm
89      driveStraight(1300,2.5,2.5);
90      //turns clockwise 90 degrees
91      turn(-90,2.5,2.5);
92      //drives straight for 200mm
93      driveStraight(200,2.5,2.5);
94      //lower arm to drop the payload off
95      moveArm(-15,6);
96      ////drives backwards for 100mm
```

```
 97      driveStraight(-100,2.5,2.5);
 98      //turns counterclockwise 90 degrees
 99      turn(180,2.5,2.5);
100      //drives straight for 350mm
101      driveStraight(350, 2.5,2.5);
102      //Follows the brown line
103      linefollow(30);
104      //Drive straight till the finish zone
105      driveStraight(400,2.5,2.5);
106      stopAllTasks(); // end of program - stop everything
107  }
108  //                                                                          ↵
       ***************************************************************************   ↵
       *
109  //   ---------------------------------- Function Definitions ----------------------------------------
110
111  //The function waitForStart enables to user to make the robot
112  //start only when the button is pressed.
113  void waitForStart(){
114      //The variable is initialised as an integer zero before start button is    ↵
         pressed
115      int Initialise = 0;
116      //The while function checks if button is pressed
117      while(Initialise ==0){
118          //checking to see if the start button is pressed, it will check if it  ↵
             is true
119          //sets button value to 1
120          Initialise = readSensor(StartButton);
121          //rechecks sensor after 50 milliseconds
122          delay(50);
123          }
124      //if start button is pressed the initialised value of 1 delays 0.5 secs    ↵
         and proceeds
125      delay(500);
126      }
127
128  //The function armtodeg takes the arm encoder count and then changes it to a   ↵
       degree of the current position
129  float armtodeg(){
130      //(21)(240.448)counts for 360 deg of the arm rotated or 360/(21 x 240.448) ↵
           degrees rotated per count
131      //highest postion is at 47 deg
132      float maxangle = 47;
133      int count =  readSensor(EncoderArm) ;
134      float deg =(float)( (360/(21*240.448)) *(float)count + maxangle);
135      //retunrs the current position
136      return (deg);
137      }
138
139  //The function wheeltomilli takes the wheel encoder count and converts them to ↵
       travelled distance in mm
140  float wheeltomilli(){
141      //1800 counts for 3(103)(pi)mm travelled or (103/600)(pi) mm travelled per ↵
           count
142      //takes average of wheel distance
143      float distance = 0.0;
```

```
144      int count = readSensor(RightEnc);
145      distance = (float) ((float)count *  (103*3.14)/(600) );
146      //returns the distance travelled by wheel
147      return distance;
148      }
149
150  //The function moveArm rotates the arm to a specified angle in degrees using a p controller
151  void moveArm(float deg,float kp){
152      //sets inital arm position as reference point
153      setSensor(EncoderArm,0);
154      //initialises values pos : postion, pwr : power, u : the output of p controller and error as integers
155      int pos,pwr,u,error;
156      //while the power level is fairly high continues to move motor
157      do{
158          //uses p controller to move arm
159          pos = armtodeg();
160          error = deg - pos;
161          u = (int)kp * error;
162          //ensures power is between -50 and 50
163          pwr = saturate(u,-50,50);
164          motorPower(ArmMotor, pwr);
165      }while(abs(error) > 10);
166      //turns motor power off to 0
167      motorPower(ArmMotor,0);
168  }
169
170  //The function driveStraight drivesd a specified distance in mm and then stops
171  void driveStraight(float distance,float kp, float ki){
172      //sets starting point as refernce point
173      setSensor(LeftEnc,0);
174      setSensor(RightEnc,0);
175      //initialises values pwr : power, error2 : equalises power between the motors, u1 : the output of the p controller for the distance as integers
176      int pwr,error2,u1;
177      //initialises pos : power, u2 : he output of the p controller for the motors as integers, error1 : difference of target distance and position
178      //e and kps as float type.
179      float pos,u2, error1;
180      float e = 0;
181      float kps = 0.5;
182
183      //while loop for when motor power is still fairly high
184      do{
185          //error for the distance(PI control) and call function wheeltomilli
186          pos = wheeltomilli();
187          error1 = distance - pos;
188          u1 = (int)(kp * error1 + ki*e);
189          //if power output is not saturated ,the pi controller intergrates
190          if (abs(u1) < 50){
191              e = e + error1;
192              }
193          //This allows the power to be saturated if outside these bounds
194          pwr = saturate(u1,-50,50);
195          //error for the wheel power inbalance(P control)
```

```
196         error2 = readSensor(LeftEnc) - readSensor(RightEnc);
197         u2 = (float)(kps*error2);
198         //sets motor power
199         motorPower(RightMotor,pwr + u2);
200         motorPower(LeftMotor,pwr - u2);
201         //While loop exits the loop if the error is below 10.
202     }while(abs(error1) > 10);
203     //turns motor off
204     motorPower(RightMotor,0);
205     motorPower(LeftMotor,0);
206 }
207
208 //The function turn rotates the robot to an angle inputed and then stops
      (Positive
209 //values results in a ounter-clockwise turn
210 void turn(float angle, float Kp, float Ki) {
211     //sets initial placement as reference
212     setSensor(RightEnc,0);
213     //initialises variables, error : the difference between target degree and
        current degree, position : current position
214     //e : integrated error as float variables
215     float error,position,e;
216     float u = 0;
217     //initalised pwr: power of motor and angleValue : desirred arc length to
        be swept as integers
218     int pwr;
219     int angleValue;
220     //converts the angle to arc length((pi)(diametre)(angle/360))
221     angleValue = ((3.14)*(robotWidth+wheelWidth) * angle)/360;
222     //do while loop until the error is within tolerance
223     do{
224             //finds postion and sets error as difference between distance
                travelled by wheel and the desired arc length
225         position =wheeltomilli();
226         error = angleValue - position;
227         u = Kp * error + Ki*e;
228
229         //checks to see if the absolute value is smaller than 50
230         if (abs(u) < 50){
231                 e = e + error;
232                 }
233             //saturates the power if it exceeds these outer bounds
234         pwr = (int) saturate(u, -50, 50);
235         //turns counterclockwise for positive values and clockwise for
            negative values
236             motorPower(RightMotor,pwr );
237             motorPower(LeftMotor,-pwr );
238             //exits the looop when error is within tolerance
239      }
240     while(abs(error)> 8 );
241     //turns motor off
242     motorPower(RightMotor,0 );
243     motorPower(LeftMotor,0 );
244 }
245
246
```

```
247 //The function linefollow allows the robot to continue moving along the brown line and stop when it sensors the colours black.
248 void linefollow(int power){
249     //initialising variable i, error:  ensures robot is driving striaght, both as integers
250     int i = 1, error;
251     //values of sensor set to zero all initialised as variables
252     int left = 0;
253     int mid = 0;
254     int right = 0;
255     //initialise variable u: output of the p controller, kps : k value for the drivestraight, both as float type.
256     float u;
257     float kps = 0.2;
258     //sets limits of brown and if on edges of brownline(between)
259     //intiatialise variable BUp as an interger : upperlimit of brown
260     int BUp = 2300;
261     //initalise variable BLow as an integer : lower limit of brown
262     int BLow = 1000;
263     //initialise varaible BLLow as an integer : lower black limit value
264     int BLLow = 2301;
265     //saturates power and drives forward for 0.5 sec to get over the black line at the start
266     int pwr;
267     power = saturate(power,-100,100);
268     pwr = (int)(power*127)/100;
269     //Sets power to the motors
270     motorPower(LeftMotor, pwr);
271     motorPower(RightMotor, pwr);
272     //delays by 500ms
273     delay(500);
274     //sets motor power to zero
275     motorPower(LeftMotor, 0);
276     motorPower(RightMotor,0);
277     //while loop while black isn't detected
278     while(i == 1){
279       left = readSensor(LeftLight);
280         mid = readSensor(MidLight);
281         right = readSensor(RightLight);
282         //if detects black ,its at end of the line(assuming it starts on brown)
283         if (left >= BLLow || mid >= BLLow|| right >= BLLow) {
284             i = 0;
285         }
286         //if any of the sensors detect brown
287         if (((right < BUp) && (right > BLow)) ||  ((mid < BUp)  && (mid > BLow)) || ((left < BUp) && (left > BLow))){
288           //if left sensor detects brown and is greater than right, then will turn right
289           if ((left > right) && (left >= BLow)){
290                   while(1){
291                       //exits loop if black is detected or the middle sensor is on the brown line
292                       if (left >= BLLow || mid >= BLLow || right >= BLLow || (mid >= BLow && mid <= BUp)){
293                           break;
```

```
294                                 }
295                             //gets current sensor values for left and middle
                        sensor
296                             left = readSensor(LeftLight);
297                             mid = readSensor(MidLight);
298                             //turns the robot to the left
299                             motorPower(LeftMotor, -pwr);
300                             motorPower(RightMotor, pwr);
301                             }
302                     }
303                         //if right sensor detects brown and is greater than left,
                        then will turn left
304                     else if ((left < right) && (right >= BLow)){
305                         while(1){
306                             //exits loop if black is detected or the middle sensor
                         is on the brown line
307                             if (right >= BLLow || mid >= BLLow || left >= BLLow ||
                         (mid >= BLow && mid <= BUp)){
308                                 break;
309                                 }
310                                 //gets current sensor values for right and middle
                        sensor
311                             right = readSensor(RightLight);
312                             mid = readSensor(MidLight);
313                                 //turns the robot to the right
314                             motorPower(LeftMotor, pwr);
315                             motorPower(RightMotor,-pwr);
316                             }
317                         }
318                             //else will drive straight
319                             else {
320                                 while (1) {
321                                     // if black is detected or the middle sensor
                        isnt detecting brown
322                                     if (mid < BLow || left > BLLow || mid > BLLow
                        || right > BLLow) {
323                                     break;
324                                         }
325                                     //gets current sensor values for middle sensor
326                                     mid = readSensor(MidLight);
327                                     //uses p controller to drive straight
328                                     error = readSensor(LeftEnc) - readSensor
                        (RightEnc);
329                                     u = (float)(kps*error);
330                                     motorPower(LeftMotor, pwr - u);
331                                     motorPower(RightMotor, pwr + u);
332                                     }
333                                 }
334         }
335     }
336         //Turns both motors off
337         motorPower(LeftMotor, 0);
338         motorPower(RightMotor, 0);
339 }
340
341
```

```
342 //The function sonicDrive drives straight using an ultasonic sesnor and a p
      contoller
343 //to a specified distance in mm away from obstacle.
344 void sonicDrive(float distance,float kp, float ki, float kps){
345     //initialises pwr : power of motors at 50, error : equalises the motors,
          u: output of the p controller as
346     //integers
347     int pwr = 50;
348     int error,u;
349     //Sets value of range as distance away from obstavle
350     float range = readSensor(sonarSensor);
351     //while nothing is in its range, continues to drive forward
352     while(range == -1){
353         //error for the wheel power inbalance(P control)
354         error = readSensor(LeftEnc) - readSensor(RightEnc);
355         u = (int)(kps*error);
356         //sets motor power
357         motorPower(RightMotor,pwr + u);
358         motorPower(LeftMotor,pwr - u);
359         //updates if anthing in range
360         range = readSensor(sonarSensor);
361         }
362     //when in sonar range uses drivestraight function
363     driveStraight(range - distance,kp,ki);
364 }
365
```