ESS REPORT

Candidate Number: 251984

University of Sussex

Contents

Introduction and Problem Outline

The problem that the group had to solve was to have an Elisa3 robot navigate a series of 4 challenges each requiring different techniques and the use of different sensors in different combinations. These challenges each issued their own issues and objectives that needed to be overcome.

For example, challenge 1 was a thick black track in the shape of a figure 8, due to its thick shape just two sensors could be used at the front of the robot, with one slightly offset to the left and one the right. However, there was an issue when it came to the paths crossing over as it wouldn't know whether to go straight or turn if the perpendicular path hit one of the sensors first. This was the opposite in challenge 2 where the line was thing and consisted of lots of turns and curves for the robot to follow so the inverse was required when it came to programming the robot. This is as the robot must use the sensors to detect white and keep itself detecting white instead of detecting black like the previous.

Challenge 3 was very different from the previous 2 with the robot having to not only be detecting the black line but also its proximity to the blocks. When one of these were detected, it had to turn by a random amount. The difficulties here were implementing the random turns, detecting the black and detecting the objects in time.

The final challenge, Challenge 4, was a culmination of all the above challenges. Initially Elisa3 must navigate a grid of black lines whilst detecting objects at large distances to avoid them with the goal of reaching the top left or top right corner of the grid. Here the grid transitions into a tunnel that the robot must navigate, that has no black line to follow, until it reaches a black box where it must stop. This introduces a variety of challenges such as Elisa3 having to be able to detect objects at large distances, being able to navigate without using proximity sensors and implementing an algorithm to get the robot to a right or left corner.

Another objective was to have the selector switch have all the different challenge modes, as well as have some utility functions, available to be accessed when changing the position, the switch is at. For example, position 1 will access challenge 1 and position 2, challenge 2.

These challenges and issues and how they were overcome is what this report details and analyses throughout its contents.

Code Description and Analysis

Starting off all the required variables were initialised such as the duty cycle, line sensors, sensors sensitivities, turn counters and an LED pin number. Following this the required pins for the selector were initialised as inputs, line 16-19, as well as the serial; monitor being initialised, line 15. This code works as intended and would be difficult to condense further.

To implement the selector the main loop, line 22, was used with if-else loops, that each call a function called “SelectorPosition()” that returns the current position the selector is in, that iterate through all the possible states for the selector. At positions 1-4 the functions for the challenge of the same number are called whilst 0, line 23, is used for debugging the ground sensors whilst an LED flashes and position 5, line 62, is used for debugging the proximity sensors. Selector 3 also has an if-else statement inside it that is used to stop the robot after a certain amount of turns which was implemented due to a misunderstanding in the requirements of challenge 3. To avoid code repetition, such as when moving the robot, code that would be used multiple times was moved to separate functions such as forward, line 24. There are some more examples where this could have been used, like the debugging outputs and LED sequences, but these are used rarely.

Lines 77 to 100 contains the code for the “SelectorPosition()” function by first reading the digital pins, that were previously initialised on lines 16-19, with each pin referencing a bit of a 4-bit number. Following this an if-else statement is et up to check for the first five possible outcomes of these 4 bits, lines 85-98, being 0000, 0001, 0010, 0011 and 0100. At each of these outcomes the corresponding number is returned from 0-4 allowing for the main loop to then execute corresponding challenge or debug function. If none of these outcomes are selected then 5 is returned, line 100. This function is quite long in its execution but other than initialising the digital reads outside of the function there is not much that can be done to improve it.

Following this are the challenge functions 1 to 4, with challenge 1 running from line 104 to 116 whose aim was to follow the thick black figure of eight described previously. To do so the ground sensors are first read before 3 if statements are used with the first, at line 107, checking that the value of the ground sensors is above the average of the black and white sensitivity of the same sensor. This therefore is checking that both sensors are detecting they are on the black line allowing for it to execute the “forward” function. The other if statements, at line 111 and 115, detect when one of the sensors has come off the line by seeing if the detected value is less than the average of the sensitivities meaning it has detected white. This then allows Elisa3 to compensate accordingly, for instance sensors one is on the left so if it detects white it must turn right therefore “rightTurn” is executed. The reason for the averages of these sensitivities being used is that the sensors are very sensitive and there were issues in development where white and black weren’t being detected as the value obtained was slightly off what as expected. Using the average of these values along with more than and less than symbols avoided this issue for the most part if the original sensitivities are correct and not too close to the opposing value.

Challenge 2's function ran from line 121 to 142 and is similar in operation to challenge 1 but utilising another two sensors for detecting when sharper turns were needed and detecting the inverse of what the sensors did in challenge 1. For example, after the ground sensors are read with the "readGroundSensors" function the if statements are iterated through. The fist checks sensors 1 and 2 are detecting white and then executes "forward" to move Elisa3 forwards. The next two if statements use sensors 1 and 2 detecting black to turn left and right accordingly to keep the robot on the thin path. However, the robot now was turning the same side that the sensor was situated on unlike in challenge 1. Sensors 0 and 3 are then used in the same way, but having the turn be faster by multiplying the duty cycle by 1.1, with sensor 0 being on the left so therefore having to turn left to keep Elisa3 on track. Sensor 3 was situated on the right and therefore executed "turnRight". Challenge 1 and 2's functions are simple and elegant in their design using simple if statements and executing movement functions as not to repeat code.

Challenge 3 runs from line 145-176 and, as with the previous challenges, starts by reading the ground sensors current state before executing a series of if-else statements. The first if-else statement was concerned with staying in the box where if sensors 1 and 2 detect the black border the vehicle reverse by executing the "backward" function before a random turn amount is chosen. This is done by having the "tightrightTurn" function be executed for a time between 300ms and 1000ms giving a range of angles between 45 to 270 degrees allowed to be taken before the turn counter is iterated on by 1. Meanwhile if no border is found the robot shall keep moving forwards before the next if-else statement is run. These next statements are focused on detecting the blocks inside of the box using the proximity sensors located around Elissa3's circumference, the ones used here are sensor 0 looking straight ahead, 1 to the right and 7 to the left. If any of these detect an object within 15mm a random turn is executed with "spin()" and the turn count iterated though if none are detected the vehicle moves forwards. One improvement that could be made here is that the else of the first if-else statement was not needed as it is executed again at the end of the second.

Challenge 4 runs from 178 to 233 and initially gets the current state of the ground sensors before executing challenge 2 which is utilised as a line following function. The first if statements focus on the solution for getting to the tunnel along the grid. This is by alternating between left and right turns until the tunnel is reached. This is done using the variable "turn" where when equal to 1 it will turn left and 0 will equal right. These turns happen when an object is detected to the left or right of the vehicle accordingly. Here "tightleftTurn" and "tightrightTurn" are used which are like the normal turn functions but the wheels are at the same speed in different directions instead of at differing speeds allowing the robot to turn on the spot. As soon as a wall is detected on either side of the bot the variable tunnel is made to equal 1 making the robot enter "tunnel mode". Once in the tunnel the robot moves forwards whilst checking for the walls to its left and right, line 214 and 223, and if they are not present then it will turn in that direction, 90 degrees. The robot will then move forwards until the black box is found. This function is one of the longest in this coursework and it could have been condensed as there is some code repetition when it comes to the navigating the grid part of the algorithm. However, utilising the challenge 2 code allowed for some loss of repetition. Lines 194 to 233 may have been over complicated slightly when it comes to navigating the tunnel, but it is difficult to find a simpler solution.

One improvement that can be made to all the challenge functions is moving the "readGroundSensors" function to the start of the main loop, so it's only called once and not in each challenge function.

Lines 236 to 258 are the functions created to read and output the values of the proximity sensors with line 137, “readProximitySensor” takes the sensors index and reads the value of the sensor at that index. Lines 240 and 243 have opposite functions where the LED of the proximity sensors of the given index is turn on or off using direct manipulation of that sensor using “PORTA”. These three functions are combined in “getDistance” at line 248 where the LED is turned om, the sensor is read and the LED is turned off for each of the eight proximity, sensors 0 to 7. The value obtained is then run through the equation obtained from MATLAB to give a distance in mm.

The ground sensor functions are found at line 260 to 274 with the first being the integral “readGroundSensors” function that turns on the ground sensor LED, reads the sensor to the “line” array and then turns the LED off. The LED being turned on and off are handled by the functions at line 269 and 272 that work like LED functions from the proximity sensors but using “PORTJ” instead of “PORTA”. The functions at 275 and 278 are used to turn on a green LED of a given LED number by digitally writing it to LOW or HIGH accordingly.

From lines 283 to 340 the different movement options are displayed for the most part they consist of a function for the left motor and a function for the right either ion the forwards or backwards direction. For example, moving forwards have both left and right moving forwards at the same speed and vice versa when moving backwards. For the turning functions the wheels would move in opposite directions, at different speeds for a normal turn and the same for turning on the spot, with left forwards and right backwards turning right. For the motor forward functions the motors were written to initially to stop any backwards movement they may have, 310 and 315, before being written to again with duty cycle, initialised previously as “s”. Due to the left wheel spinning slower the left had to be slowed by 24% by multiplying the duty cycle by 0.76 when writing to it. With the backward functions the same method was done but the positions the index used were flipped so forward momentum was stopped and backward momentum was applied. The final function created was “spin” where the “backward” function was called with a delay to control its duration before “tightrightTurn” was called with a random delay between 300ms and 1000ms to give it a random turn between 45 and 270 degrees. This concludes the movement functions with the final function of the code being the average function that takes two values, usually the black and white sensitivity, and returns the average of them.

From lines 236 to 345, by utilising direct manipulation and separating this code into its own separate functions the size of the file was condensed, which is important for uploading the code to the small space of the Elisa3 and its execution. Also, if any changes had to be made it only had to be done once here and not every place where these functions were required.

Analysis and Conclusion

Both the selector and Challenge 1 were executed extremely well with no issues with Elisa3 managing to get the right number, stick to the line and not being confused by the paths crossing over. Challenge 2 went well counter-clockwise but when going clockwise the curve with the tight turn wasn't registered meaning that the bot lost the path and drove off. However, we were able to lower the duty cycle slightly which resolved this as the issue seemed to be in that direction the bot moved to fast to pick up the black line.

For challenge 3 Elisa 3 performed it well managing to complete 15 random turns when detecting the border but some of the turns put it back into the black or the object so multiple were then executed in a row. Therefore 15 rotations weren't always seen, also 15mm was too low a detection range and should have been higher as it is turning too close to the objects made it hit the object occasionally. Therefore, increasing this means the bot can turn earlier and not hit the blocks.

In challenge 4 the bot barely operated as expected as it was able to follow the lines of the grid but was never able to make it to the tunnel. The reason for this is probably the use of challenge 2 as a line following function that was probably not needed in retrospect as the turning algorithm could have just used parts of it instead.

In Conclusion, aside from challenge 4, Elisa3 operated as expected with some improvements to be made tweaking the duty cycles, light sensitivity and the distances proximity sensors detected at. Challenge 4 however has some major issues that need to be addressed to get it to the tunnel where it should operate as intended but due to it never making it there it is an unknown. The layout of the code is mostly excellent in how condensed it is and its layout with only small discrepancies of repetition and over complicated solutions. Overall though I believe it was a well-executed study with more positives than negatives that completed the large majority of the original objectives.

Appendix

Appendix Word Count: 1595

Code

```
1. int line[4];//store black light numbers
2. float s;//speed or duty cycle
3. int black0=1016;//Values for black on sensors 0-3
4. int black1=1010;
5. int black2=1010;
6. int black3=1016;
7. int white0=1000;//Values for white on sensors 0-3
8. int white1=980;
9. int white2=980;
10. int white3=1000;
11. int turncount=0;//Count number of turns in challenge 3
12. int turn = 1;//Decide turn type in challenge 4
13. unsigned char LEDno=48;
14. void setup() {//Open the serial monitors for
15.   Serial.begin(9600);
16.   pinMode(37,INPUT);
17.   pinMode(36,INPUT);
18.   pinMode(35,INPUT);
19.   pinMode(34,INPUT);
20. }
21.
22. void loop() {// Main Loop Function
23.   if (SelectorPosition()==0){ //Check selector is at position 0
24.     forward(0);//Set wheel speeds to 0
25.     greenLEDon(LEDno);//Make LED blink
26.     delay(100);
27.     greenLEDoff(LEDno);
28.     delay(100);
29.     readGroundSensors();//Update ground sensor readings
```

```
30.     for(int i=0;i<4;i++){//For loop to output updated readings to allow for debugging and sensitivity adjustments
31.       Serial.print("Light sensor: ");
32.       Serial.print(i);
33.       Serial.print(" light level ");
34.       Serial.println(line[i]);
35.     }
36.   }
37.   else if(SelectorPosition()==1){//Check selector is at position 1
38.     s=30;//Set Custom speed for challenge 1 and execute challenge 1
39.     Challenge_1();
40.   }
41.   else if(SelectorPosition()==2){//Check selector is at position 2
42.     s=28;//Set Custom speed for challenge 2 and execute challenge 2
43.     Challenge_2();
44.   }
45.   else if(SelectorPosition()==3){//Check selector is at position 3
46.     if (turncount<15){//Check maximum turns haven't been taken for challenge 3
47.       s=30;//Set Custom speed for challenge 1 and execute challenge 3
48.       Challenge_3();
49.     }
50.     else if(turncount>=15){//If maximum turns have been taken for challenge 3 stop robot
51.       forward(s);
52.       delay(1000);
53.       s=0;
54.     }
55.
56.   }
57.   else if(SelectorPosition()==4){//Check selector is at position 4
58.     s=30;//Set Custom speed for challenge 1 and execute challenge 4
59.     Challenge_4();
60.   }
61.
```

```
62.  else if (SelectorPosition()==5){//Check selector is at position 5
63.    //Get distance for distance sensors 0-2 & 6-7 and output them for debugging
64.    Serial.print("Distance of sensor 0 is: ");
65.    Serial.println(getDistance(0));
66.    Serial.print("Distance of sensor 6 is: ");
67.    Serial.println(getDistance(6));
68.    Serial.print("Distance of sensor 2 is: ");
69.    Serial.println(getDistance(2));
70.    Serial.print("Distance of sensor 7 is: ");
71.    Serial.println(getDistance(7));
72.    Serial.print("Distance of sensor 1 is: ");
73.    Serial.println(getDistance(1));
74.  }
75. }
76.
77. //Selector Function
78. unsigned char SelectorPosition(){
79.   //Initialise Each bit of the selector
80.   int sel0=digitalRead(37);
81.   int sel1=digitalRead(36);
82.   int sel2=digitalRead(35);
83.   int sel3=digitalRead(34);
84.   //iterate through the possible options
85.   if(sel0==LOW && sel1==LOW && sel2==LOW && sel3==LOW){
86.     return 0;
87.   }
88.   else if(sel0==HIGH && sel1==LOW && sel2==LOW && sel3==LOW){
89.     return 1;
90.   }
91.   else if(sel0==LOW && sel1==HIGH && sel2==LOW && sel3==LOW){
92.     return 2;
93.   }
```

```
94.   else if(sel0==HIGH && sel1==HIGH && sel2==LOW && sel3==LOW){
95.     return 3;
96.   }
97.   else if(sel0==LOW && sel1==LOW && sel2==HIGH && sel3==LOW){
98.     return 4;
99.   }
100.   return 5;
101. }
102.
103. //Challenge functions
104. void Challenge_1(){
105.   readGroundSensors();//Update Ground Sensors
106.   //ground sensors 1&2> average of black and white values; must be on black line so go forwards
107.   if ((line[1]>avr(black1,white1))&&(line[2]>avr(black2,white2))){
108.     forward(s);
109.   }
110.   //ground sensor 1<average of black and white values sensor 1 is off the line so turn right to compensate
111.   if (line[1]<avr(black1,white1)){
112.     rightTurn(s);
113.   }
114.   //ground sensor 2<average of black and white values sensor 1 is off the line so turn left to compensate
115.   if(line[2]<avr(black2,white2)){
116.     leftTurn(s);
117.   }
118.
119. }
120.
121. void Challenge_2(){
122.     readGroundSensors();//Update Ground Sensors
123.     //ground sensors 1&2< average of black and white values; must be either side of black line so go forwards
```

```
124.    if (line[1]<avr(black1,white1) && line[2]<avr(black2,white2)){
125.      forward(s);
126.    }
127.    //ground sensor 1>average of black and white values sensor 1 is on the line so turn left to compensate
128.    if (line[1]>avr(black1,white1)){
129.      leftTurn(s);
130.    }
131.    //ground sensor 2>average of black and white values sensor 2 is on the line so turn right to compensate
132.    if (line[2]>avr(black2,white2)){
133.      rightTurn(s);
134.    }
135.    //ground sensor 0>average of black and white values sensor 0 is on the line so turn right alot to compensate
136.    if (line[0]>avr(black0,white0)){
137.      leftTurn(s*1.1);
138.    }
139.    //ground sensor 3>average of black and white values sensor 3 is on the line so turn right alot to compensate
140.    if (line[3]>avr(black3,white3)){
141.      rightTurn(s*1.1);
142.    }
143. }
144.
145. void Challenge_3(){
146.   readGroundSensors();//Update Ground Sensors
147.   //ground sensor 1 or 2> average of black and white values; a sensor either sensor must be on the black line so is at the border
148.   if(line[1]>avr(black1,white1) || line[2]>avr(white2,black2)){
149.    //Reverse from edge, spin randomly and iterate turncount
150.    backward(s);
151.    delay(200);
152.    tightrightTurn(s);
```

```
153.    delay(random(300,1000));
154.    turncount++;
155.  }
156.  else{
157.    //if not at edge keep going forwards
158.    forward(s);
159.  }
160.  if (getDistance(0)<15){//If sensor 0 is within 15mm of an object
161.    spin(s);//Execute random spin function
162.    turncount++;//iterate turncount
163.  }
164.  else if (getDistance(7)<15){
165.    spin(s);//Execute random spin function
166.    turncount++;//iterate turncount
167.  }
168.  else if (getDistance(1)<15){
169.    spin(s);//Execute random spin function
170.    turncount++;//iterate turncount
171.  }
172.  else{//No object so keep moving forward
173.    forward(s);
174.  }
175.
176. }
177.
178. void Challenge_4(){
179.    readGroundSensors();//Update ground sensors
180.    Challenge_2();//Use challenge 2's line following so not to repeat code
181.    if (turn == 1){//Left Turn Next
182.      if(getDistance(0)<25 || getDistance(1)<30 ||getDistance(7)<30){//if sensors 0, 1 or 7 are
within 30mm of an object
183.        //Stop, turn left tightly, set turn to 0 and start moving forward
184.        forward(0);
```

```
185.        delay(900);
186.        tightleftTurn(s);
187.        delay(600);
188.        turn = 0;
189.        forward(s);
190.        delay(100);
191.      }
192.     }
193.     if (turn == 0){//Right Turn Next
194.      if(getDistance(0)<25 || getDistance(1)<30 ||getDistance(7)<30){//if sensors 0, 1 or 7 are within 30mm of an object
195.        //Stop, turn right tightly, set turn to 1 and start moving forward
196.        forward(0);
197.        delay(900);
198.        tightrightTurn(s);
199.        delay(500);
200.        turn = 1;
201.        forward(s);
202.        delay(100);
203.      }
204.     }
205.     // use side sensors to detect walls and ground sensors to check line is gone
206.     if (getDistance(6)<20 && getDistance(2)<20 && line[1]<avr(black1,white1) && line[2]<avr(white2,black2)){
207.      int tunnel=1;
208.      while(tunnel==1){//put bot in tunnel mode (loop)
209.        if(line[1]<avr(black1,white1) || line[2]<avr(white2,black2)){//If ground sensors detect the box turn off tunnel mode and stop
210.          tunnel=0;
211.          forward(0);
212.          delay(1000);
213.        }
214.        else if(getDistance(6)>100 && getDistance(2)<20){//If left sensors detect no tunnel but right does turn left
```

```
215.          leftTurn(s);
216.          forward(s);
217.        }
218.        else if(getDistance(6)<20 && getDistance(2)>100){//If right sensors detect no tunnel but left does turn right
219.          rightTurn(s);
220.          forward(s);
221.        }
222.        else{//otherwise go forwards
223.          forward(s);
224.        }
225.      }
226.     }
227.     if(line[1]>avr(black1,white1)&&line[0]>avr(black0,white0)){//If ground sensors detect a line go forwards
228.       s = 25;
229.       forward(s);
230.       delay(500);
231.       s = 30;
232.     }
233. }
234.
235.
236. //Prox Sensor Functions
237. int readProximitySensor(unsigned char proxIndex){//return value of proximity sensor at given pin
238.   return analogRead(proxIndex);
239. }
240. void proxLEDon(unsigned char proxIndex){//Turn on the LED of a proximity sensor at given pin
241.   PORTA |= (1<<proxIndex);
242. }
243. void proxLEDoff(unsigned char proxIndex){//Turn off the LED of a proximity sensor at given pin
244.    PORTA &= ~(1<<proxIndex);
245. }
```

```
246.
247. //Pin Values(6 left, 2 right, 0 straight ahead)
248. int getDistance(unsigned char sensor){
249.   //Iterate through Przimity Sensor indexes, turn on their LEDs, read the value and turn the LEDs off again
250.   for (unsigned char i=0;i<8;i++){
251.     proxLEDon(sensor);
252.     delay(10);
253.     int x=(readProximitySensor(sensor));
254.     proxLEDoff(sensor);
255.     delay(10);
256.     return (4.158e-10*exp(0.02773*x)+3.568*exp(0.002083*x));//Substitute value in Matlab equation to get distance in mm
257.   }
258. }
259.
260. //Ground Sensor Functions
261. void readGroundSensors(){//Read ground sensors 0,1,2,3 by turing their LEDs on, reading values and turning them off again
262.   for (int i=0; i<4;i++){
263.     groundLEDon(i);
264.     delay(1);
265.     line[i] = analogRead(8+i);
266.     groundLEDoff(i);
267.     }
268. }
269. void groundLEDon(unsigned char lineIndex){//Turn a ground sensor LED on with pin
270.   PORTJ |=(1<<lineIndex);//Set index to ON
271. }
272. void groundLEDoff(unsigned char lineIndex){//Turn a ground sensor LED on with pin
273.   PORTJ &= ~(1<<lineIndex);//Set index to OFF
274. }
275. void greenLEDon(unsigned char LEDno){//Turn a green LED on with pin
276.   digitalWrite(LEDno, LOW);
```

```
277. }
278. void greenLEDoff(unsigned char LEDno){//Turn a green LED on with pin
279.   digitalWrite(LEDno, HIGH);
280. }
281.
282.
283. //Movement functions
284. void tightrightTurn(float s){//Left Motor Forward & Right Motor Backward at same speed; turn on spot
285.   rightMotorBackward(s);
286.   leftMotorForward(s);
287.
288. }
289. void tightleftTurn(float s){//Right Motor Forward & Left Motor Backward at same speed; turn on spot
290.   rightMotorForward(s);
291.   leftMotorBackward(s);
293. }
295. void leftTurn(float s){//Right Motor Forward & Left Motor Forward at a reduced speed
296.   rightMotorForward(s);
297.   leftMotorForward(s*0.6);
298. }
299. void rightTurn(float s){//Left Motor Forward & Right Motor Forward at a reduced speed
300.   rightMotorForward(s*0.6);
301.   leftMotorForward(s);
302. }
304. void forward(float s){//Right Motor Forward & Left Motor Forward
305.   rightMotorForward(s);
306.   leftMotorForward(s);
307. }
309. void rightMotorForward(float s){
310.   analogWrite(2,0);//Stop any backwards rotation
311.   analogWrite(5,s*0.76);//Rotate Forwards with a compenstaion for broken motor
```

```
312. }
314. void leftMotorForward(float s){
315.   analogWrite(7,0);//Stop any backwards rotation
316.   analogWrite(6,s);//Rotate Forwards
317. }
318.
319. void backward(float s){//Right Motor Backwards & Left Motor Backwards
320.   rightMotorBackward(s);
321.   leftMotorBackward(s);
322. }
323.
324. void leftMotorBackward(float s){
325.   analogWrite(6,0);//Stop any Forwards rotation
326.   analogWrite(7,s);//Rotate Backwards
327. }
328.
329. void rightMotorBackward(float s){
330.   analogWrite(5,0);//Stop any Forwards rotation
331.   analogWrite(2,s);//Rotate Backwards
332. }
333.
334. void spin(float s){
335.   //Reverse from object and spin a random amount
336.   backward(s);
337.   delay(200);
338.   tightrightTurn(s);
339.   delay(random(300,1000));
340. }
342. //Other functions
343. int avr(int B, int W){//Fucntion to take to values and give the average
344.   return (B+W)/2;
345. }
```