EMBEDDED SYSTEMS AND SENSORS PROJECT REPORT

Introduction

This report outlines the successes and struggles faced when programming the Elisa -3 robot to perform a set of 4 challenges using Arduino. This is a great first step into the programming of embedded systems. Despite being a small robot, the fundamentals can be taken into many wider fields in robotics. Most automated systems will use similar methods to sense information of both internal and external environments whilst processing this information in such a way that it can be used to determine its state and in turn activate the required output. In this case the robot will be using infrared sensors to measure light levels in the environment to determine the colour of the ground and the proximity of obstacles. This information will be processed and used to ascertain the wheel motor controls needed to complete the given challenge.

Background / Key Functions

Analogue to Digital Converter (ADC)

An ADC converts an analogue voltage to a digital number. Converting from the analogue world to the digital world, allows the information of varying values (between 0 and 3.3V) to be processed by the controller. The Arduino board contains a multichannel 10-bit ADC, meaning the input voltages between 0 and 3.3V is mapped to an integer value between 0 and 1023 ($2^{10} = 1024$). Therefore providing a resolution ≈ 0.0032.

analogRead() is the built in Arduino function designed to read the converted value from a specified pin.

Pulse Width Modulation (PWM)

PWM is a way to control analogue devices with a digital signal. While the signal can only be high (3.3V) or low (ground), at any time, the proportion of time the signal is high compared to when it is low over a consistent time interval can be changed, to achieve an average output. The duty cycle is the percentage of time a digital signal is high, over a period of time equal to the inverse of the PWM frequency (Arduino default = 500Hz).

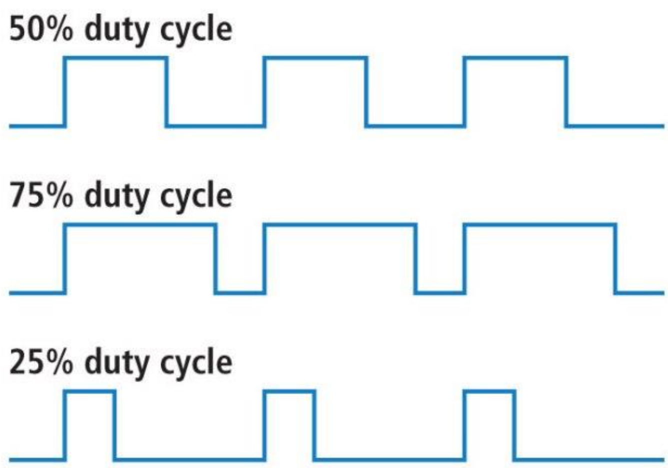


Figure 1: Duty Cycle

analogWrite() is the built-in Arduino function to perform this. It takes two arguments, the digital pin number, and the desired duty cycle. Since the duty cycle has an 8-bit resolution, this value is between 0 (always off) and 255 (always on) ($2^{8} = 256$). This can be mapped to 0 – 100%, should this be easier to work with. The function can be used to light LEDs at various brightness or drive the motors at various speeds.

Active Ground Sensing

The Elisa robot has four ground infrared sensors, situated at its base. Two at the front, (front left / front right), and two situated further back, just in front of the wheels (back left / back right). These sensors work by transmitting infrared light through LEDs. When the light hits the ground surface it bounces back to the receiver returning an analogue input voltage. The ADC converts this into an integer value between 0 and 1023. This can be used to detect changes in in surface such as colour, where darker colours, i.e. black, absorb more of the light, thus returning a lower light level. Therefore, the robot can distinguish between the two based on the returned value. In this case, the more reflected light received the lower value read on the pin.

First the pins mapped to the four ground sensors must be initialised as outputs (pins 0 to 3 on Port J). To achieve this the data direction register for port lines PJ0 – PJ3 must be set to 1: $DDRJ \mid = 00001111$ (Line 7). The *readGroundSensor()* function (line 504 – 518) reads the value at a given sensor, returning a value for black and white. The function takes an integer input to allow for the reading of each individual sensor (0 – 3).

The LEDs must be activated for a short period of time to allow for a sensor response, by setting the output data register for port J to one. Here the *groundLEDon()* function is called to activate the LED of the inputted pin. A delay of 5ms was chosen for the sensor response. Then the *analogRead()* function was used to read the value from the pin of the receiver associated with the LED that's been called. *groundLEDoff()* is used to turn the LED off to conserve battery. The function is also used in conjunction with *thresh* – an array containing the tuned integer values for the threshold between black and white lines. If the returned value from sensor is greater than the given threshold, the function returns 0 (black), otherwise 1 (white).

One problem with the ground sensors is that they are extremely sensitive to the light levels of the surrounding environment. Therefore, the thresholding values need to be tuned and updated from room to room, and even throughout the day as level of light changes. The function *readGroundSensors()* (line 521 – 546) is used to read every sensor returning the value each one reads, and the determination of black or white based on the current threshold values. These values can then be updated based on the results.

Active Proximity Sensing

The Elisa robot has 8 infrared sensors situated around the circumference of the robot, each placed $45°$ away from each other and capable of sensing up to 6cm. They can be used in both passive and active mode. Passive, to measure the light levels emitted from the environment (no LED light emitted). Active, to measure the proximity of objects. To measure the proximity of objects, the active sensing works in the same way as the ground sensing above. Infrared light is transmitted through the LEDs situated at each of sensors, which will reflect off any nearby objects and return to the receiver which measures the light as a voltage (0 – 3.3V). Once again, the ADC converts this to an integer between 0 and 1023. The level of light received can then be mapped to the distance, with more light returning from closer objects. In this case the closer the object, the closer this value is to 0. However, from our results it is clear that this value is not a linear reflection of the distance, but instead resembles an exponential curve. This had to be considered when tuning the threshold values. Much like the ground sensors, these sensors are sensitive to the level of light in the environment, once again requiring constant monitoring and tuning.

The pins for the LEDs are located at the respective pins on Port A and must be initialised as outputs (line 6). The *readProximitySensor()* function (line 549 – 556), takes an integer as an input for the desired LED/sensor, and returns the integer value converted by the ADC. Since the 8 IR sensors are located at digital pins 0 – 7 respectively, no conversion is required between the input used to activate and deactivate the LEDs, and that used read the values at each sensor.

Movement

The pins used to control the left motor are digital pins 6 and 7 for forwards and backwards respectively. The corresponding pins for the right motor are digital pins 5 and 6. For forwards motion the backwards pin should be set to 0, and PWM control using *analogWrite()* used to control the forwards speed (vice-versa for backwards motion).

Line 570 – 671 are the functions used to control the movement of the robot. Due to varying inertia in each of the wheels, the left wheel rotated at slower speed to the right wheel for the same duty cycle. Therefore, the left wheel had to be tuned to achieve the same speed as the right wheel, with a single duty input ($duty \times 1.1$ for forwards motion (line 608)). This value also had to be tuned for turning control, to achieve the correct angular transformation based on the input.

Rotary Selector

The rotary selector has 16 positions corresponding to the value of the sel0, sel1, sel2, sel3 signals, located at pins 0 – 3 on port C ($2^4 = 16$). The pin values corresponding to the selector position is defined below:

Sel3	Sel2	Sel1	Sel0	Current Selector Position	Sel3	Sel2	Sel1	Sel0	Current Selector Position
0	0	0	0	0	1	0	0	0	8
0	0	0	1	1	1	0	0	1	9
0	0	1	0	2	1	0	1	0	A
0	0	1	1	3	1	0	1	1	B
0	1	0	0	4	1	1	0	0	C
0	1	0	1	5	1	1	0	1	D
0	1	1	0	6	1	1	1	0	E
0	1	1	1	7	1	1	1	1	F

Figure 2: Selector Switch pin values

The pins PC0 – PC3 must be initialised as inputs in the data direction register (line 5). The function *selectorPosition()* (line 58 – 61) uses the input register (PINC) to read the values of the for pins, and return the current selector position.

This was implemented in the main loop of the program using an if-else if structure to determine which functions to run based on the selector position. This allowed for the various challenges to be completed using the same code.

Challenge 1

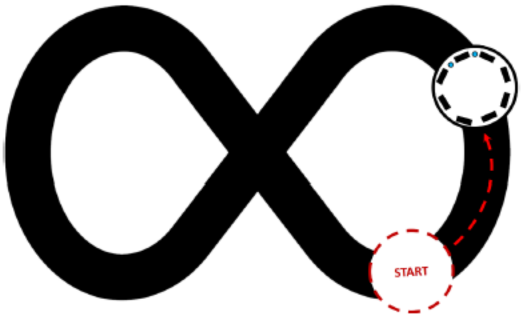


Figure 3: Challenge 1 - line follower 1

In challenge 1 the robot is required to follow the figure of 8 circuit as in Figure 3, in both the clockwise and anti-clockwise direction. The track is wide enough so 3 sensors can detect it at once meaning at least one sensor will read black at any given time. The four active ground infrared sensors had to be used to be able to distinguish between the black line to follow and the white paper. Varying wheel motor inputs can implemented based upon the results of *readGroundSensor()* at each sensor, to ensure the black line is followed. To activate the controls required for this challenge, the rotary selector is turned to 0.

To follow the desired path, the function, *LineMovement()* (line 65 – 87) was created. This calls on the *readGroundSensor()* function to determine the colour each ground sensor is reading and follows a simple if-else if structure to dictate the motion of the robot based on inputs from the sensors.

```
65  void lineMovement() // movement algorithm for challange 1 and challenge 2
66  {
67    if(!readGroundSensor(1)&&readGroundSensor(2))  //if front left sensor reads black and front right reads white
68      {
69        turnLeft(duty);
70      }
71    else if(readGroundSensor(1)&&!readGroundSensor(2))  //if front left sensor reads white and front right reads black
72      {
73        turnRight(duty);
74      }
75    else if(!readGroundSensor(3)&&readGroundSensor(0))  //if back left senso reads white and back right reads black //constrained with && because of part of track where 0 and 3 read black
76      {
77        turnRightFast(duty);                        //sharp right turn
78      }
79    else if(!readGroundSensor(0)&&readGroundSensor(3))  //if back left sensor reads blcak and back right reads white //constrained with && because of part of track where 0 and 3 read black
80      {
81        turnLeftFast(duty);                         //sharp left turn
82      }
83    else
84      {
85        forwards(duty);
86      }
87  }
```

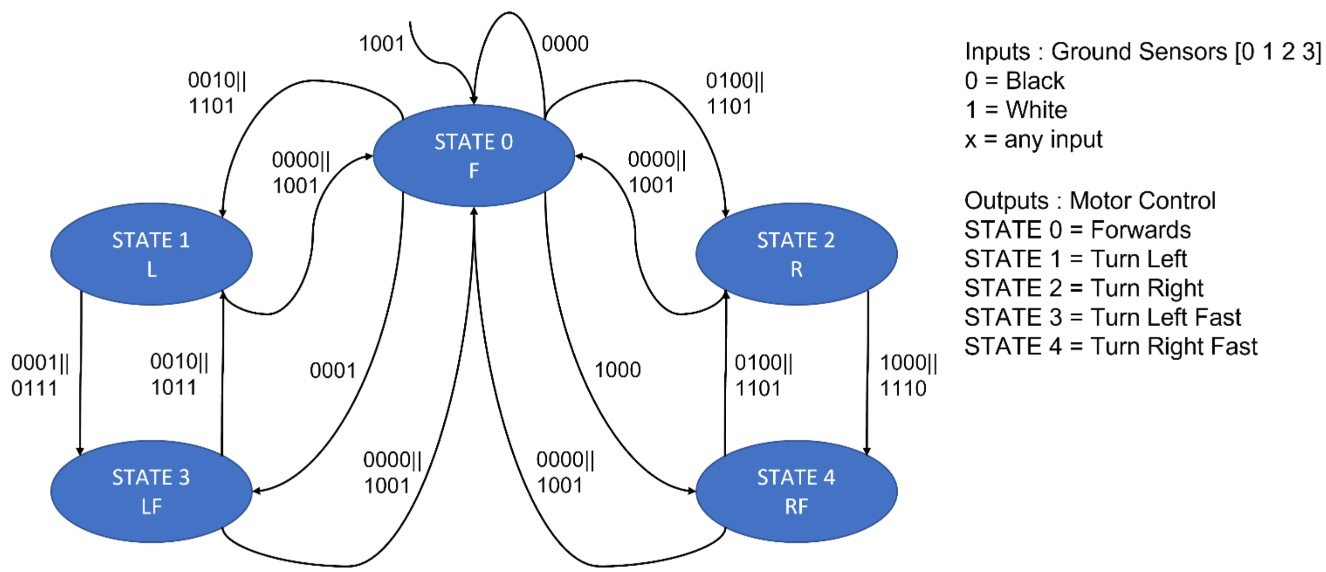


Figure 4: Finite State Machine for Challenge 1

Figure 4 represents the finite state machine for challenge 1. It models the relationship between the inputs from the ground sensors and the outputs at the wheel motors, reflecting the if-else if nature of the function. The turning functions are constrained so that they only occur in conditions that require a turn.

One problem faced with this task was when the robot crossed the cross section of the track. In addition, since this section follows the exit of a turn, the robot is often entering this part at an angle. During testing it was often the case for the robot to turn at this section instead of following straight. To solve this, the forward motion is not constrained to particular values from the ground sensors (e.g. front sensors == black), instead, if none of the turning statements are satisfied the robot moves forward. Therefore, despite being at an angle, the robot continues straight through this section. Although after this the robot is close to leaving the track, it can catch itself using the turn fast functions to regain the track.

This method is much more robust than our previous method, with far more constraints associated to each of the movements. Due to the number of constraints, some of the statements contradicted each other, resulting in an undesirable performance. Whilst the robot would successfully stay on the track, it would often make a turn at the cross section.

There are additional methods to complete this task such as a switch/case logical structure or implementing a PID to control the motion based upon the integer value read by the sensors, as opposed to the binary black/white. However, given the simple nature of the challenge, adding additional complexity is not necessary.

Using the updated *LineMovement()* and ensuring the thresholds at each of the sensors are tuned correctly to the current light levels, the robot completes the task in both the clockwise and anticlockwise direction.

Challenge 2

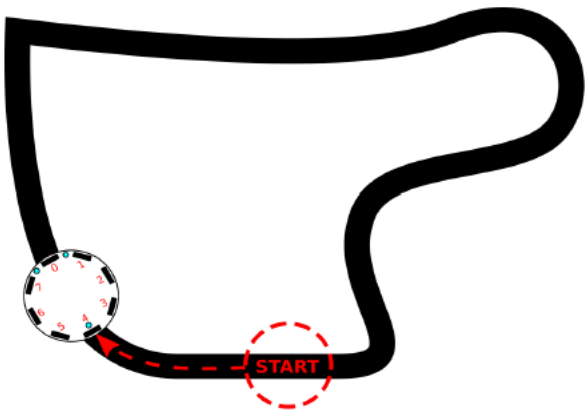


Figure 5: Challenge 2 - line follower 2

Challenge 2 requires the robot to follow the circuit shown in Figure 5, both clockwise and anti-clockwise. The track here is narrower than the previous challenge such that it fits between the front two sensors. Once again, the four ground sensors are used to distinguish the black track from the white paper. To activate the controls required for this challenge, the rotary selector is turned to 1.

The logic actually remains the same as the previous challenge, so the function *lineMovement()* is used once more. The differences come from the possible inputs from the ground sensors, which is illustrated in Figure 6 below.

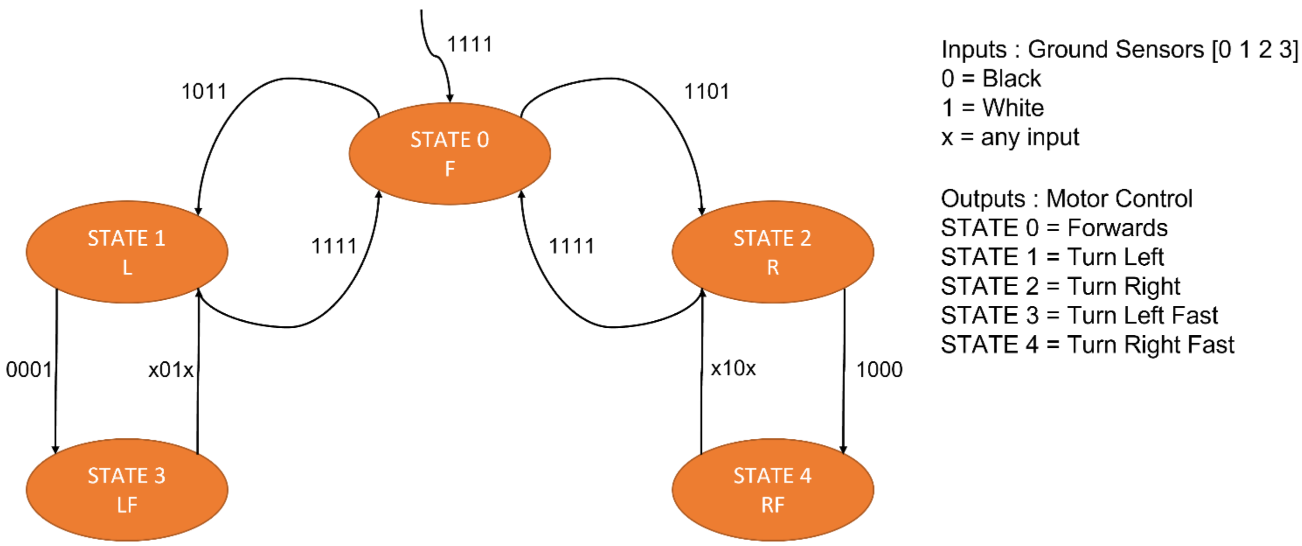


Figure 6: Finite State Machine for Challenge 2

Figure 6 represents the finite state machine for challenge 2. As you can see, there are fewer possible inputs resulting in more constrained movement between states.

The main problem associated with this challenge are the tight corners on the track. This is where the turn fast functions perform really well (line 75 – 82). For example, if the robot approaches a right turn on the track, and the rICC (radius of the Instantaneous Centre of Curvature) of the *turnRight()* function is too large to make the tight corner, the back right sensor (3) will soon read black. Once this happens, the *turnRightFast()* statement is satisfied. This function increases the speed of the left wheel whilst decreasing the speed of the right wheel, thus performing a tighter turn. This function and the equivalent *turnLeftFast()* allowed the robot to negotiate, the sharp corners of the track.

Using far less constrained logic than initially designed greatly improved the performance of this task. In addition, allowing the forward motion to occur whenever the turning functions are not satisfied created a much smoother motion compared to when it was constrained to certain inputs.

As was the case with challenge 1, methods such as switch/case logic or PID control could be used to complete this challenge. However once again, such complexity is not required. Using *LineMovement()* and ensuring the thresholds at each of the sensors are tuned correctly to the current light levels, the robot completes the task in both the clockwise and anticlockwise direction.

Challenge 3

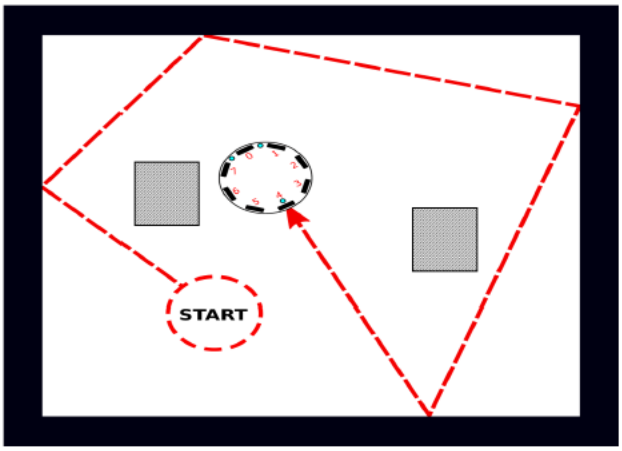


Figure 7: Challenge 3 - Stay inside the box and avoid obstacles

Challenge 3 requires the robot to stay within the black box whilst avoiding wooden blocks (40x40x40mm), as can be seen in Figure 7. The robot must be able to turn clockwise and anti-clockwise, and at a varying angle each time, to reach all sides of the box. Here, both the ground sensors (to detect the box edge), and the proximity sensors (to detect the obstacles) are used. *Challenge3()* (line 89 – 158) is the function used to determine the movement required based on the sensor readings. To activate the controls required for this challenge, the rotary selector is turned to 2.

```
89   void challenge3()
90   {
91       int rDelay = random(0,400); //randomise the time for a turn to vary turn angle
92       if(!readGroundSensor(1))
93       {
94         spinClockwise(duty-1);     //rotate clockwise for a random time between 0-400ms
95         delay(rDelay);
96       }
97       else if(!readGroundSensor(2))
98       {
99         spinAntiClockwise(duty-1);//rotate anti-clockwise for a random time between 0-400ms
100        delay(rDelay);
101      }
```

```
    else if(!readGroundSensor(1)&&!readGroundSensor(2))
    {
      backwards(duty);
      delay(500);
      int LR = random(0,1);     //randomiser to deterimine direction of rotation
      if(LR = 0)
      {
        spinClockwise(duty);
        delay(rDelay);
      }
      else
      {
        spinAntiClockwise(duty);
        delay(rDelay);
      }
    }
    else if(readProximitySensor(0) < 830) //tuned value for the front sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      backwards(duty);
      delay(400);
      int LR = random(0,1);
      if(LR = 0)
      {
        spinClockwise(duty);
        delay(350);
      }
      else
      {
        spinAntiClockwise(duty);
        delay(350);
      }
    }
    else if(readProximitySensor(1) < 810) //tuned value for the front right sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      spinAntiClockwise(duty);
      delay(350);
    }

    else if(readProximitySensor(7) < 800) //tuned value for the front left sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      spinClockwise(duty);
      delay(350);
    }
    else if(readProximitySensor(2) < 855) //tuned value for the right sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      turnLeft(duty);
    }

    else if(readProximitySensor(6) < 850) ////tuned value for the left sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      turnRight(duty);
    }
    else
    {
      forwards(duty);
    }
}
```

readGroundSensor() is once again used to detect the black edges as above, and *readProximityFunction()* is used to detect obstacles. To stay within the bounds of the box and avoid the obstacles, the functions *spinClockwise()* (line 624 – 628) and *spinAntiClockwise* (line 630 – 634) are used to rotate the robot by setting one wheel to rotate forwards and the other backwards depending on the direction required. The direction of rotation determined by what side of the robot the line or obstacle is detected (the robot will always spin away). For example if proximity sensor 7 (front left) detects an obstacle, the robot will rotate clockwise away from the obstacle. The built in Arduino function *random()*, is used to vary the angular rotation at each spin. Taking a random value between 0 and 400, this value is inputted into the delay used to determine the duration of each spin, therefore randomising the resultant angle. This, and the ability to rotate in both directions allowed the robot to reach each side of the box over time.

Inputs :
Ground Sensors [0 1 2 3]
0 = Black
1 = White
x = any input

Proximity Sensors [0 1 2 3 4 5 6 7]
0 = Obstacle detected
1 = No Obstacle detected
x = any input

Outputs : Motor Control
STATE 0 = Forwards
STATE 1 = Turn Left
STATE 2 = Turn Right
STATE 3 = Backwards then Spin (CW or ACW)
STATE 4 = Spin Anti-Clockwise
STATE 5 = Spin Clockwise

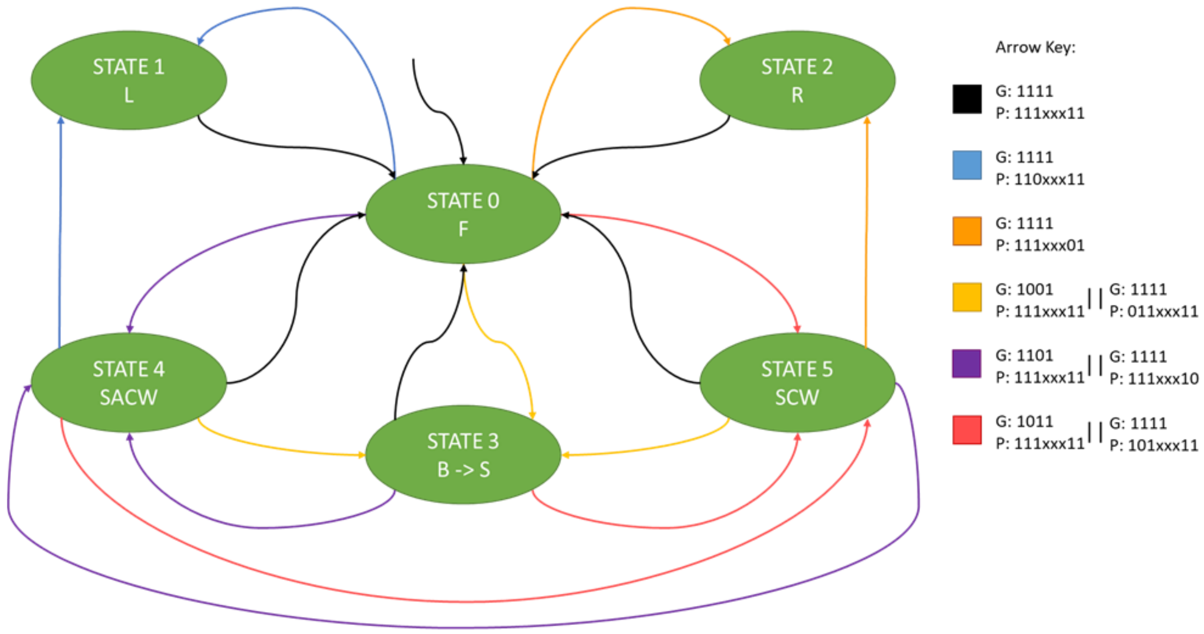


Figure 8: Finite State Machine for Challenge 3

Figure 8 represents the relationship between the various inputs, and outputs resulting from the wheel motors. Given the increased number of inputs, the state diagram appears much more complex. However, the logical structure remains the same as previous functions (if-else if), with the movement between states becoming less constrained due to the random angle at each spin.

One problem faced with this challenge occurred when the robot travelled head on into a corner (i.e. readGroundSensor(1) && readGroundSensor(2) == black). Due to both sensors reading black, resulting in numerous spins in each direction of an angle < 90°, the robot can wiggle its way out of the box. To counter this, backwards movement is utilised when this occurs (line 102 – 117), allowing the robot to move away from the edge before rotating. The same was implemented when the robot is travelling head on to an obstacle (line 118 – 127). State 3 in Figure 8 represents this movement and the inputs required to implement it.

Using the logic described, the robot successfully completes challenge 3 with no issues.

Challenge 4

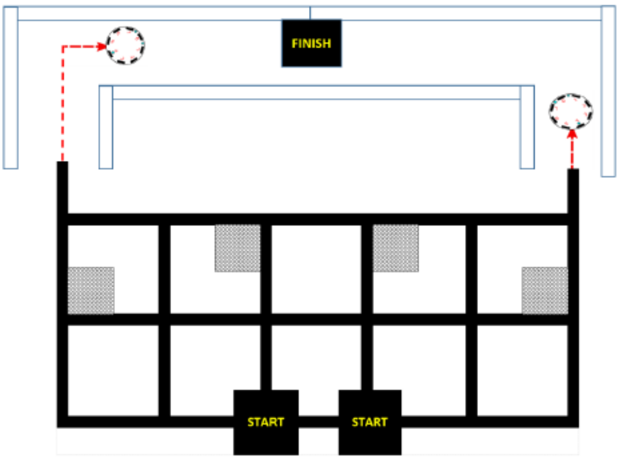


Figure 9: Challenge 4 - line and corridor follower

Challenge 4 is the most difficult task to complete. Such that there was not enough time to implement the desired solution. Nonetheless, the fundamental idea will be detailed here. Illustrated in Figure 9, the robot is required to start at either of the start positions and begin movement through the gridded track using the ground sensors, until it reaches the corridor where it must follow the wall until it reaches the finish.

Since there are at least two possible paths to take requiring differing outputs based on the same/similar inputs, the rotary selector is switched to either 3 or 4, for starting on the left or right respectively. The logic required to complete the challenge can be split into two parts: the grid and the corridor. This is represented in the code where two functions are used for each starting position.

The grid track proved a great challenge due to the multiple directions the robot could theoretically take whilst following the black line. Instead the correct path through the grid must be programmed to reach the corridor.

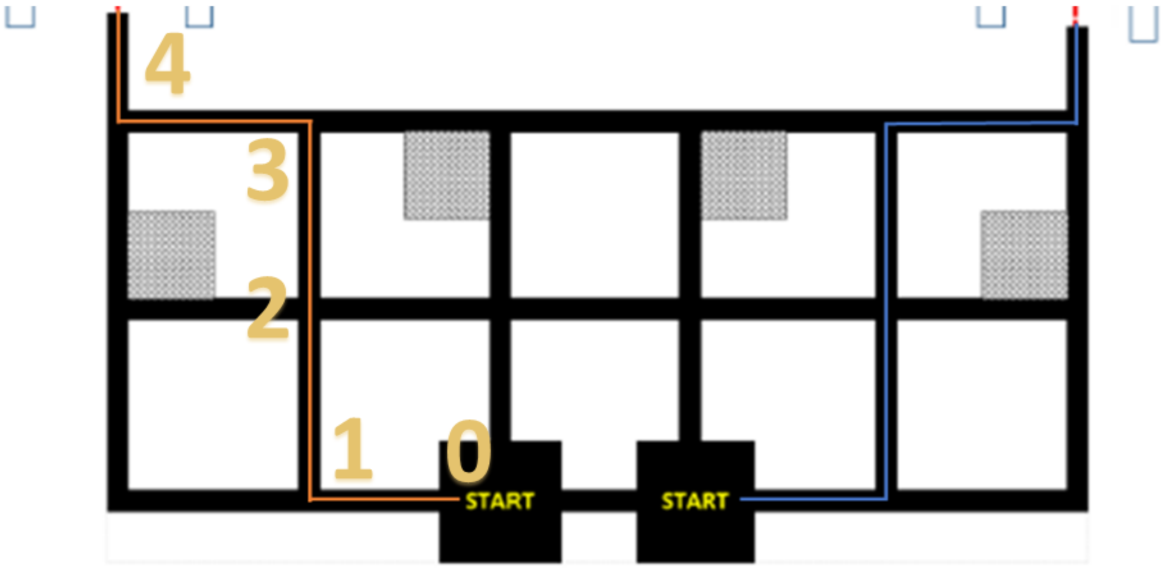


Figure 10: Challenge 4 - desired path through the grid

Figure 10 shows the desired path through the grid, depending on the starting position. Because of the varying movements, two functions are required *challenge4GridLeft()* (below) (line 184 – 258) and *challenge4GridRight()* (line 260 – 334).

```
184 void challenge4GridLeft()
185 {
186
187   if(count == 0)                         //count is initialised to zero
188   {
189     forwards(duty);                          //release from the starting square
//facing left
190     delay(1000);                           //delay to ensure robot is beyond the
junction - back sensors no longer read black
191     count++;                             ////increase turn counter by 1 //repeats
after each turn
192   }
193   else if(count == 1 && !readGroundSensor(3) && readGroundSensor(0))  //when
first corner is approached turn right
194   {
195     Stop();
196     delay(500);
197     while(readGroundSensor(0))                                  //while
back left sensor reads white to ensure full turn is completed
198     {
199       spinClockwise(duty);
200       forwards(duty);
201       delay(500);                                               //delay
to ensure robot is beyond the junction - back sensors no longer read black
202       count++;
203     }
204   }
```

```
  else if(count == 2 && !readGroundSensor(1) && !readGroundSensor(2))
//when second junction (count == 2) is approached go straight ahead
  {
    forwards(duty);
    delay(500);
//delay to ensure robot is beyond the junction - back sensors no longer read black
    count++;
  }
  else if(count == 3 && !readGroundSensor(3) && !readGroundSensor(0)) //when third junction (count == 3) is approached turn left
  {
    Stop();
    delay(500);
    if(!readGroundSensor(0)&&readGroundSensor(3))
//spin until back left reads black and back right reads white simultaneously
    {
      Stop();
      delay(500);
      forwards(duty);
      delay(500);
//delay to ensure robot is beyond the junction - back sensors no longer read black
      count++;
    }
    else
    {
      spinAntiClockwise(duty);
    }
  }
  else if(count == 4 && !readGroundSensor(3) && !readGroundSensor(0)) //when forth junction (count == 4) is approached turn right
  {
    Stop();
    delay(500);
    if(readGroundSensor(0)&&!readGroundSensor(3))
//spin until back right reads black and back left reads white simultaneously
    {
      Stop();
      delay(500);
      forwards(duty);
      delay(1500);
//longer delay to allow the robot to enter the corridor
      count++;
    }
    else
    {
      spinClockwise(duty);
    }
  }

  else if(!readGroundSensor(1)&&readGroundSensor(2))
//general line following code
  {
    turnLeft(duty);
  }
  else if(readGroundSensor(1)&&!readGroundSensor(2))
  {
    turnRight(duty);
  }
  else
  {
    forwards(duty);
  }
}
```

To work out the required movement at each point, a counter (*count)* is used to keep track of the number of turns and thus the robot's location on the track. The numbers at each junction indicate the value of the counter as the robot enters the junction. This is increased by one once the 'turn' is completed. The front sensors are used to follow the line (line 246 – 253), as in challenge 2. The back sensors can be used to determine the correct orientation of the robot. For example, if the robot is starting from the left, as the robot enters turn 1 the back right sensor will read black whilst the back left will continue to read white, prompting the robot to stop. It will then rotate clockwise until the back left sensor reads black, therefore completing a 90° rotation. Moving forward the count is increased by one to 2 as the second junction is approached. Here the robot continues straight, and the counter is increased by one once more. Entering junction 3 both the back sensors will read black, and the robot will come to a stop. Here an anticlockwise rotation is required, indicted by the count being equal to 3. The if-else statement here rotates the robot anti-clockwise until the back left sensor reads black, *and* the back right sensor reads black. Then the robot will move forward and increase the counter. Turn four requires a clockwise rotation and follows the same formula as turn 3 but flipped for the clockwise turn. The forwards movement following the rotation is also increased to allow the robot to enter the corridor before the counter is increased and the wall following algorithm begins.

Figure 11 below shows the finite state machine for this section. It indicates the motion (state) implemented based on the readings from the ground sensors and the value of the counter, as well as the possible movement between the states.

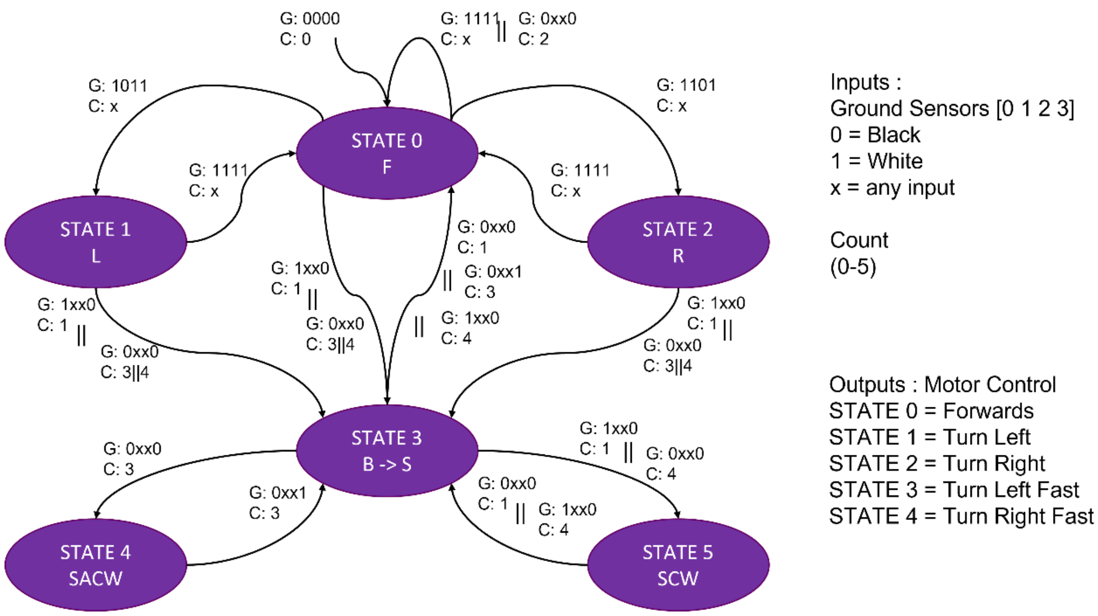


Figure 11: Finite State Machine for Challenge 4 - Grid Section, Method 1 LEFT

The method if the robot starts on the right-hand side is the equivalent, but flipped since process is the same, just the direction of each rotation changes.

Another method that should allow the robot to move faster through this section but is perhaps less robust has also been created. Functions *challenge4GridLeftFast()* (below) (line 336 – 388) and *challenge4GridRightFast()* line(390 – 443). Once again, the counter is used to determine the direction of the turn at each junction. The front sensors are used to follow the line, whilst the back sensors are used to sense the location of each junction. In this method, instead of spinning on the spot, the turn fast functions are used to turn sharply round each corner. In addition, in the previous method, the back sensors are used to determine the orientation of the robot. Here the robot performs a sharp turn whilst the back sensor on the inside of the turn reads black. Whilst this certainly increases the speed through this section, its reliability is diminished as it has no prediction of its orientation.

```
void challenge4GridLeftFast()    //if starting on the left
{

  if(count == 0)                          //count is initialised to zero
  {
    forwards(duty);                          //release from the starting square //facing left
    delay(1000);
    count++;                          ////increase turn counter by 1 //repeats after each turn
  }
  else if(count == 1 && !readGroundSensor(3) && readGroundSensor(0))  //when first corner is approached turn right
  {
    while(!readGroundSensor(3)) //while back right sensor reads black to ensure full turn is completed
    {
      turnRightFast(duty);
      count++;
    }
  }
  else if(count == 2 && !readGroundSensor(1) && !readGroundSensor(2)) //when second junction (count == 2) is approached go straight ahead
  {
    forwards(duty);
    delay(500);
    count++;
  }
  else if(count == 3 && !readGroundSensor(3) && !readGroundSensor(0)) //when third junction (count == 3) is approached turn left
  {
    while(!readGroundSensor(0)) //while back left sensor reads black to ensure full turn is completed
    {
      turnLeftFast(duty);
      count++;
    }
  }
  else if(count == 4 && !readGroundSensor(3) && !readGroundSensor(0)) //when forth junction (count == 4) is approached turn right
  {
    while(!readGroundSensor(3)) ////while back right sensor reads black to ensure full turn is completed
    {
      turnRightFast(duty);
      count++;
    }
  }

```

```
    else if(!readGroundSensor(1)&&readGroundSensor(2)) //general line following code
    {
      turnLeft(duty);
    }
    else if(readGroundSensor(1)&&!readGroundSensor(2))
    {
      turnRight(duty);
    }
    else
    {
      forwards(duty);
    }
}
```

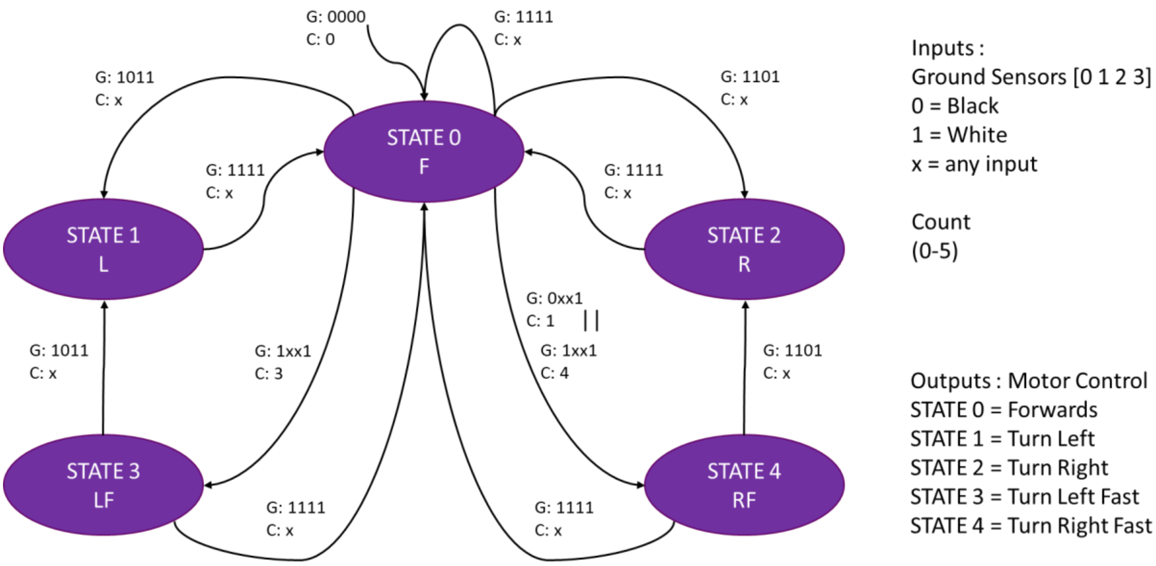


Figure 12: Finite State Machine for Challenge 4 - Grid Section Method 2 LEFT

Figure 12 is the finite state machine for this method once again starting on the left, indicating the differing motions (states) implemented depending on the various sensor readings and counter value using this algorithm. This method is activated when rotator switch equals 5 and 6 for left and right starting positions respectively.

To negotiate the corridor section, a wall following algorithm is used based upon the proximity sensor reading of the robot's distance from the wall. As with the grid section, the algorithm depends on side the robot started so two functions are used: *wallFollowingLeft()* (below)(line 446 – 467) and *wallFollowingRight()* (line 469 – 490). Firstly, this is only active once the turn count reaches 5 (i.e. when the grid section is completed). Continuing on the left side of the track, the robot will continue straight until the robot is a given distance away from the wall on the right-hand side as indicated by proximity sensor 3. Once it passes this threshold the control algorithm kicks in, and it will begin to turn right and continue straight until it gets too close to the wall. When this occurs the controller switches on once more to prompt the robot to begin a left turn until the robot is at a safe distance from the wall. This iterates throughout the course, until the ground sensors sense black, where the robot will stop at the finish.

```
void wallFollowingLeft() // if starting on the left of the track
{
  if(count == 5)            // if grid is completed
  {
    if(readProximitySensor(2) > 825)  //hug the right wall
    {
      turnRight(duty);
    }
    else if(readProximitySensor(2) < 750) //give enough room from the wall
    {
      turnLeft(duty);
    }
    else if(!readGroundSensor(0) || !readGroundSensor(3)) //stop when robot has passed through the grid and reaches black finish square
    {
      Stop();
    }
    else
    {
      forwards(duty);
    }
  }
}
```

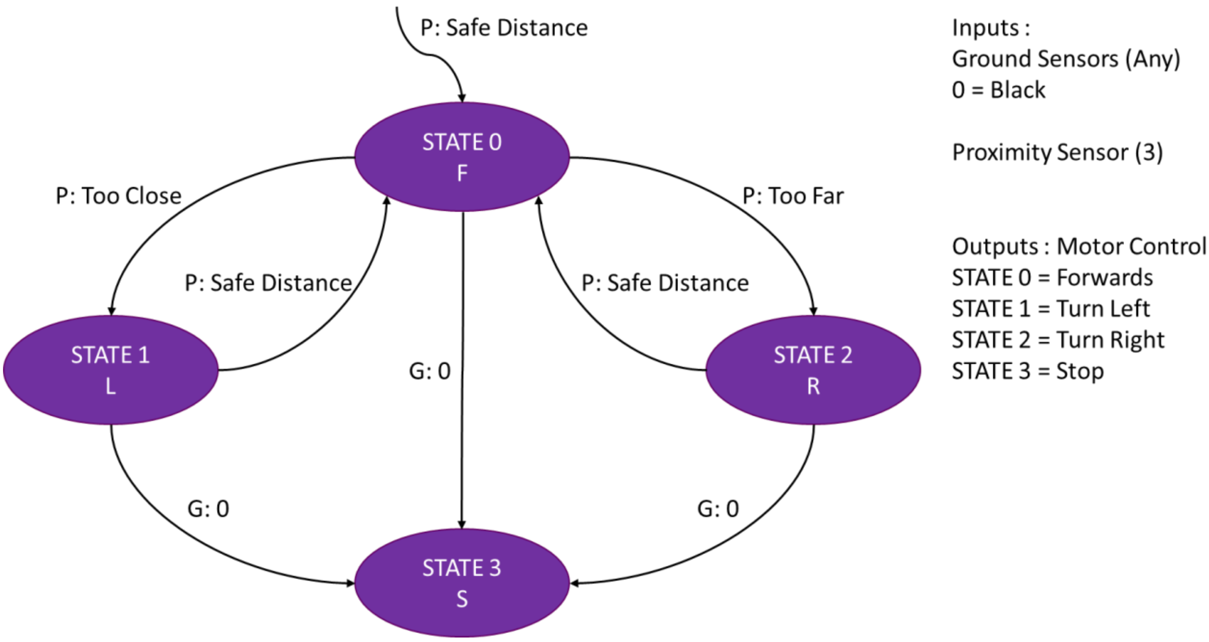


Figure 13: Finite State Machine for Challenge 4 - Corridor Section LEFT

To determine the movement (state) at any given time in the corridor, the finite state diagram (Figure 13) showing the required motor outputs for the current sensor input can be referenced.

As mentioned above, given the time constraints there was not enough time to test these algorithms in order to tune the sensors and the turning velocities. However, with testing it is believed that they should work correctly.

Conclusion

Throughout this project I have learnt how to successfully program an embedded system to perform a task given there is certain data readings of the external environment. I have developed an understanding of how the system can convert an analogue value, such as light level, into a digital value that can be processed with the ADC. As well as the reverse where a digital signal (on/off) can be programmed into an analogue output such as motor speed using the PWM. I have also further enhanced my ability in logical C/Arduino programming, creating successor functions based on the current state of the robot and its sensor readings. Whilst some of the logic (challenge 4) could not be completed in time for testing and demonstration, with the additional time I believe I have completed it.

Appendix

Full Code

```
float duty;     //variable to hold the duty cycle value (0 - 255)
int count = 0; //variable to count number of 'turns' in challenge 4
void setup()
{
  DDRC &= B11110000; //set pins 0-3 (sel0, sel1, sel2, sel3) as inputs for rotary selector switch // &= ensures pins 0-3 = 0, whilst maintaining any previously set values for pins 4-7
  DDRA |= B11111111; //set all pins as outputs // initialise proximity LEDs
  DDRJ |= B00001111; //set pins 0-3 as outputs // initialise ground LEDs // |= ensures pins 0-3 = 1, whilst maintaining any previously set values for pins 4-7

    Serial.begin(2000000);  //baud rate set perhaps too fast but this setting improved the performance of the challenges
//    Serial.begin(300);
}

void loop()
{
  if(selectorPosition() == 0)
  {
    duty = 20;
    lineMovement(); // challenge 1
  }
  else if(selectorPosition() == 1)
  {
    duty = 21;
    lineMovement(); // challenge 2
  }
  else if(selectorPosition() == 2)
  {
    duty = 20;
    challenge3();
    // challenge 3
  }
  else if(selectorPosition() == 3)
  {
    duty = 18;
    challenge4Left();
  }
  else if(selectorPosition() == 4)
  {
    duty = 18;
    challenge4Right();
  }
  else if(selectorPosition() == 5)
  {
    duty = 18;
    challenge4LeftFast();
  }
  else if(selectorPosition() == 6)
  {
    duty = 18;
    challenge4RightFast();
  }
  else
  {
```

```
    Stop();
    readGroundSensors();
  }
}

unsigned char selectorPosition() //read rotary selector position
{
  return PINC & 0x0F; //read pins 0-3 at port C and return value
}

//CHALLENGE FUNCTIONS

void lineMovement() // movement algorithm for challange 1 and challenge 2
{
  if(!readGroundSensor(1)&&readGroundSensor(2))  //if front left sensor reads black and front right reads white
    {
      turnLeft(duty);
    }
  else if(readGroundSensor(1)&&!readGroundSensor(2))  //if front left sensor reads white and front right reads black
    {
      turnRight(duty);
    }
  else if(!readGroundSensor(3)&&readGroundSensor(0))  //if back left senso reads white and back right reads black //constrained with && because of part of track where 0 and 3 read black
    {
      turnRightFast(duty);                          //sharp right turn
    }
  else if(!readGroundSensor(0)&&readGroundSensor(3))  //if back left sensor reads blcak and back right reads white //constrained with && because of part of track where 0 and 3 read black
    {
      turnLeftFast(duty);                           //sharp left turn
    }
  else
    {
      forwards(duty);
    }
}

void challenge3()
{
    int rDelay = random(0,400); //randomise the time for a turn to vary turn angle
    if(!readGroundSensor(1))
    {
      spinClockwise(duty-1);    //rotate clockwise for a random time between 0-400ms
      delay(rDelay);
    }
    else if(!readGroundSensor(2))
    {
      spinAntiClockwise(duty-1);//rotate anti-clockwise for a random time between 0-400ms
      delay(rDelay);
    }
    else if(!readGroundSensor(1)&&!readGroundSensor(2))
```

```
    {
      backwards(duty);
      delay(500);
      int LR = random(0,1);     //randomiser to deterimine direction of rotation
      if(LR = 0)
      {
        spinClockwise(duty);
        delay(rDelay);
      }
      else
      {
        spinAntiClockwise(duty);
        delay(rDelay);
      }
    }
    else if(readProximitySensor(0) < 830) //tuned value for the front sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      backwards(duty);
      delay(400);
      int LR = random(0,1);
      if(LR = 0)
      {
        spinClockwise(duty);
        delay(350);
      }
      else
      {
        spinAntiClockwise(duty);
        delay(350);
      }
    }
    else if(readProximitySensor(1) < 810) //tuned value for the front right sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      spinAntiClockwise(duty);
      delay(350);
    }

    else if(readProximitySensor(7) < 800) //tuned value for the front left sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      spinClockwise(duty);
      delay(350);
    }
    else if(readProximitySensor(2) < 855) //tuned value for the right sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      turnLeft(duty);
    }

    else if(readProximitySensor(6) < 850) ////tuned value for the left sensor - close enough to allow for movement throughout the box, far enough to avoid contact with block
    {
      turnRight(duty);
```

```
    }
    else
    {
      forwards(duty);
    }
}

void challenge4Left()
{
  challenge4GridLeft();
  wallFollowingLeft;
}

void challenge4Right()
{
  challenge4GridRight();
  wallFollowingRight;
}

void challenge4LeftFast()
{
  challenge4GridLeftFast();
  wallFollowingLeft;
}

void challenge4RightFast()
{
  challenge4GridRightFast();
  wallFollowingRight;
}

void challenge4GridLeft()
{

  if(count == 0)                      //count is initialised to zero
  {
    forwards(duty);                       //release from the starting square //facing left
    delay(1000);                        //delay to ensure robot is beyond the junction - back sensors no longer read black
    count++;                          ////increase turn counter by 1 //repeats after each turn
  }
  else if(count == 1 && !readGroundSensor(3) && readGroundSensor(0)) //when first corner is approached turn right
  {
    Stop();
    delay(500);
    while(readGroundSensor(0)) //while back left sensor reads white to ensure full turn is completed
    {
      spinClockwise(duty);
      forwards(duty);
      delay(500); //delay to ensure robot is beyond the junction - back sensors no longer read black
      count++;
    }
  }
  else if(count == 2 && !readGroundSensor(1) && !readGroundSensor(2)) //when second junction (count == 2) is approached go straight ahead
```

```
{
  forwards(duty);
  delay(500); //delay to ensure robot is beyond the junction - back sensors no longer read black
  count++;
}
else if(count == 3 && !readGroundSensor(3) && !readGroundSensor(0)) //when third junction (count == 3) is approached turn left
{
  Stop();
  delay(500);
  if(!readGroundSensor(0)&&readGroundSensor(3)) //spin until back left reads black and back right reads white simultaneously
  {
    Stop();
    delay(500);
    forwards(duty);
    delay(500); //delay to ensure robot is beyond the junction - back sensors no longer read black
    count++;
  }
  else
  {
    spinAntiClockwise(duty);
  }
}
else if(count == 4 && !readGroundSensor(3) && !readGroundSensor(0)) //when forth junction (count == 4) is approached turn right
{
  Stop();
  delay(500);
  if(readGroundSensor(0)&&!readGroundSensor(3)) //spin until back right reads black and back left reads white simultaneously
  {
    Stop();
    delay(500);
    forwards(duty);
    delay(1500); //longer delay to allow the robot to enter the corridor
    count++;
  }
  else
  {
    spinClockwise(duty);
  }
}

else if(!readGroundSensor(1)&&readGroundSensor(2)) //general line following code
{
  turnLeft(duty);
}
else if(readGroundSensor(1)&&!readGroundSensor(2))
{
  turnRight(duty);
}
else
```

```
  {
    forwards(duty);
  }
}

void challenge4GridRight()
{

  if(count == 0)                          //count is initialised to zero
  {
    forwards(duty);                           //release from the starting square //facing right
    delay(1000);
    count++;                              ////increase turn counter by 1 //repeats after each turn
  }
  else if(count == 1 && readGroundSensor(3) && !readGroundSensor(0)) //when first corner is approached turn left
  {
    Stop();
    delay(500);
    while(readGroundSensor(3)) //while back right sensor reads white to ensure full turn is completed
    {
      spinAntiClockwise(duty);
      forwards(duty);
      delay(500);
      count++;
    }
  }
  else if(count == 2 && !readGroundSensor(1) && !readGroundSensor(2)) //when second junction (count == 2) is approached go straight ahead
  {
    forwards(duty);
    delay(500);
    count++;
  }
  else if(count == 3 && !readGroundSensor(3) && !readGroundSensor(0)) //when third junction (count == 3) is approached turn right
  {
    Stop();
    delay(500);
    if(readGroundSensor(0)&&!readGroundSensor(3)) //spin until back right reads black and back left reads white simultaneously
    {
      Stop();
      delay(500);
      forwards(duty);
      delay(500);
      count++;
    }
    else
    {
      spinClockwise(duty);
    }
  }
  else if(count == 4 && !readGroundSensor(3) && !readGroundSensor(0)) //when forth junction (count == 4) is approached turn left
  {
    Stop();
```

```
    delay(500);
    if(!readGroundSensor(0)&&readGroundSensor(3))
//spin until back left reads black and back right reads white simultaneously
    {
      Stop();
      delay(500);
      forwards(duty);
      delay(1500);
//longer delay to allow the robot to enter the corridor
      count++;
    }
    else
    {
      spinAntiClockwise(duty);
    }
  }

  else if(!readGroundSensor(1)&&readGroundSensor(2))
//general line following code
  {
    turnLeft(duty);
  }
  else if(readGroundSensor(1)&&!readGroundSensor(2))
  {
    turnRight(duty);
  }
  else
  {
    forwards(duty);
  }
}

void challenge4GridLeftFast()    //if starting on the left
{

  if(count == 0)                        //count is initialised to zero
  {
    forwards(duty);                          //release from the starting square //facing left
    delay(1000);
    count++;                            ////increase turn counter by 1
//repeats after each turn
  }
  else if(count == 1 && !readGroundSensor(3) && readGroundSensor(0))
//when first corner is approached turn right
  {
    while(!readGroundSensor(3))
//while back right sensor reads black to ensure full turn is completed
    {
      turnRightFast(duty);
      count++;
    }
  }
  else if(count == 2 && !readGroundSensor(1) && !readGroundSensor(2))
//when second junction (count == 2) is approached go straight ahead
  {
    forwards(duty);
    delay(500);
    count++;
  }
```

```
  else if(count == 3 && !readGroundSensor(3) && !readGroundSensor(0)) //when third junction (count == 3) is approached turn left
  {
    while(!readGroundSensor(0)) //while back left sensor reads black to ensure full turn is completed
    {
      turnLeftFast(duty);
      count++;
    }
  }
  else if(count == 4 && !readGroundSensor(3) && !readGroundSensor(0)) //when forth junction (count == 4) is approached turn right
  {
    while(!readGroundSensor(3)) ////while back right sensor reads black to ensure full turn is completed
    {
      turnRightFast(duty);
      count++;
    }
  }

  else if(!readGroundSensor(1)&&readGroundSensor(2)) //general line following code
  {
    turnLeft(duty);
  }
  else if(readGroundSensor(1)&&!readGroundSensor(2))
  {
    turnRight(duty);
  }
  else
  {
    forwards(duty);
  }
}

void challenge4GridRightFast() //equivalent to above but reversed
{
  if(count == 0)
  {
    forwards(duty);
    delay(1000);
    count++;
  }
  else if(count == 1 && readGroundSensor(3) && !readGroundSensor(0))
  {
    while(!readGroundSensor(0))
    {
      turnLeftFast(duty);
      count++;
    }
  }
  else if(count == 2 && !readGroundSensor(1) && !readGroundSensor(2))
  {
    delay(500);
    count++;
  }
  else if(count == 3 && !readGroundSensor(3) && !readGroundSensor(0))
  {
    while(!readGroundSensor(3))
    {
```

```
      turnRightFast(duty);
      count++;
    }
  }
  else if(count == 4 && !readGroundSensor(3) && !readGroundSensor(0))
  {
    while(!readGroundSensor(0))
    {
      turnLeftFast(duty);
      count++;
    }
  }
  else if(count == 5 && !readGroundSensor(1) && !readGroundSensor(2))
  {
    Stop();
  }
  else if(!readGroundSensor(1)&&readGroundSensor(2))
  {
    turnLeft(duty);
  }
  else if(readGroundSensor(1)&&!readGroundSensor(2))
  {
    turnRight(duty);
  }
  else
  {
    forwards(duty);
  }
}


void wallFollowingLeft() // if starting on the left of the track
{
  if(count == 5)             // if grid is completed
  {
    if(readProximitySensor(2) > 825)  //hug the right wall
    {
      turnRight(duty);
    }
    else if(readProximitySensor(2) < 750) //give enough room from the wall
    {
      turnLeft(duty);
    }
    else if(!readGroundSensor(0) || !readGroundSensor(3)) //stop when robot has passed through the grid and reaches black finish square
    {
      Stop();
    }
    else
    {
      forwards(duty);
    }
  }
}

void wallFollowingRight() // if starting on the right of the track
{
  if(count == 5);              //if grid is completed
  {
    if(readProximitySensor(6) > 825) //hug the left wall
```

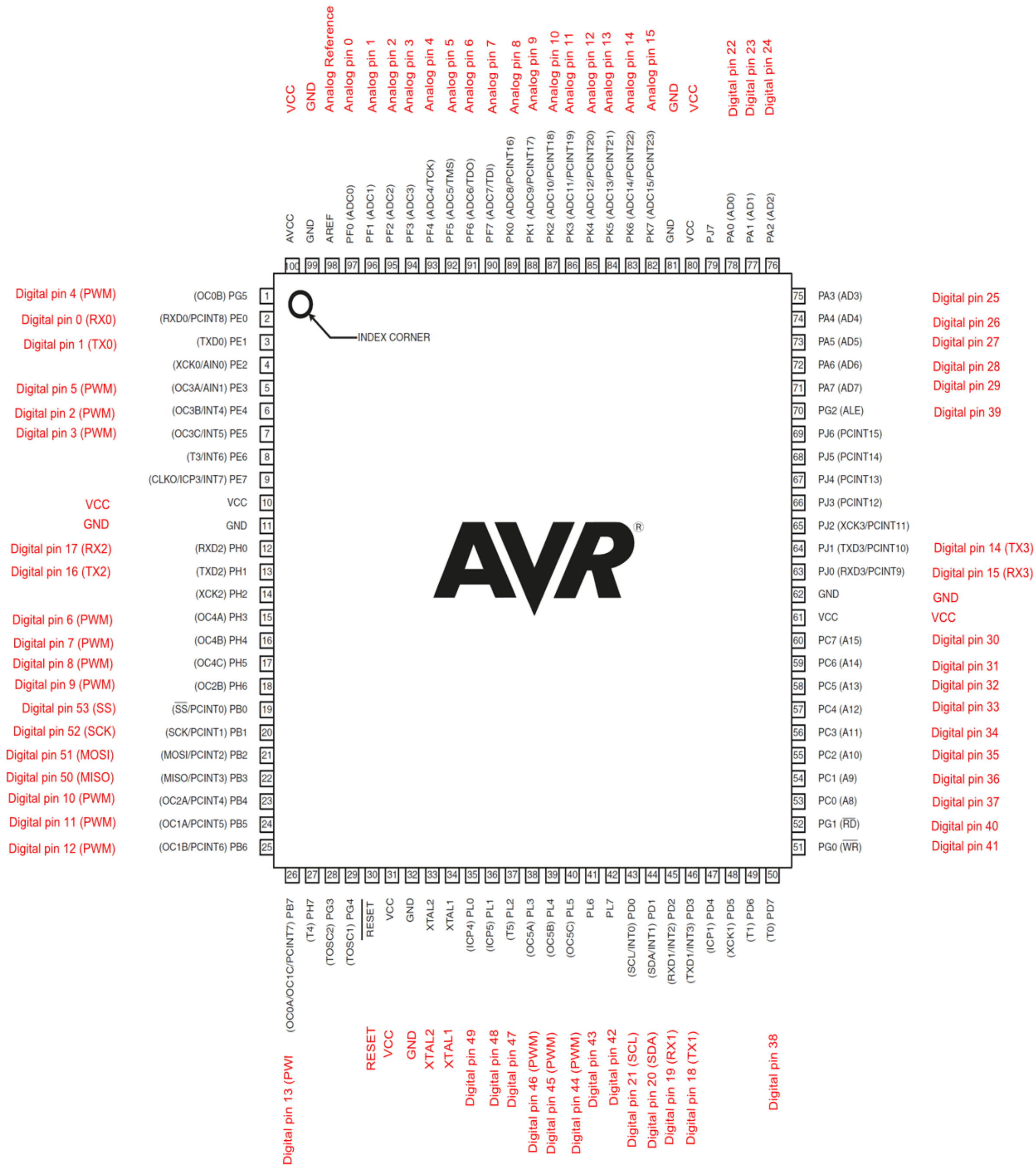
```
    {
      turnLeft(duty);
    }
    else if(readProximitySensor(6) < 750) //give enough room from the wall
    {
      turnRight(duty);
    }
    else
    {
      forwards(duty);
    }
  }
}

//SENSOR FUNCTIONS
//Ground sensor functions
int thresh[] = {981, 966, 968, 983}; // black/white thresholds for sensors {0, 1, 2, 3}
int thresh1[] = {968, 940, 939, 955}; //used to keep track of previous values when tuning


void groundLEDon(unsigned char lineindex)
{
  PORTJ |= (1<<lineindex); //set inputted LED pin to 1 (turn on)
}

void groundLEDoff(unsigned char lineindex)
{
  PORTJ &= (0<<lineindex); //set inputted LED pin to 0 (turn off)
}

int readGroundSensor(unsigned char lineindex) //read value at an inputted ground sensor - return black / white determination
{
    groundLEDon(lineindex);
    delay(5);                            //5 ms delay
    int val = analogRead(lineindex+8); //infrared receivers mapped to ADC channels 8-11 for ground LEDs 0-3 respectively //variable to store read values
    groundLEDoff(lineindex);
    if(val > thresh[lineindex])          //comparison with tuned threshold values
    {
      return 0; //black
    }
    else
    {
      return 1; //white
    }
}

int groundArr[4];          //array to store sensor readings
void readGroundSensors()  //used for tuning thresholds // when rotary selector value > 4
{

  for(int i = 0; i < 4; i ++)       //iteration through each sensor
  {
    groundLEDon(i);
```

```
    delay(5);
    groundArr[i] = analogRead(i+8);
    groundLEDoff(i);
    Serial.print("Ground sensor ");        //process same as above except Serial.print to print values to monitor
    Serial.print(i);
    Serial.print(": ");
    Serial.println(groundArr[i]);
    Serial.print("Ground sensor ");
    Serial.print(i);
    Serial.print(": ");
    if(groundArr[i] > thresh[i])
    {
      Serial.println("Black");
    }
    else
    {
      Serial.println("white");
    }

  }
}

// prox sensor functions
int readProximitySensor(unsigned char proxindex)    // read value at an inputted proximity sensor value
{
    proxLEDon(proxindex);
    delay(5);                                        //5ms delay
    int val = analogRead(proxindex);                 //variable to store read value //sensor pins mapped to digital pins 0-7
    proxLEDoff(proxindex);
    return val;                                      //return read value
}

void proxLEDon(unsigned char proxindex)
{
  PORTA |= (1<<proxindex); //set inputted LED pin to 1 (turn on)
}

void proxLEDoff(unsigned char proxindex)
{
  PORTA &= (0<<proxindex); //set inputted LED pin to 1 (turn off)
}

//MOVEMENT FUNCTIONS

void turnRight(float duty)
{
  rightMotorForward(duty*0.5);
  leftMotorForward(duty*1.1);
}

void turnRightFast(float duty)
{
  rightMotorForward(duty*0);
  leftMotorForward(duty);
}

void turnRightSlow(float duty)
{
```

```
  rightMotorForward(duty*0.8);
  leftMotorForward(duty*1.1);
}

void turnLeft(float duty)
{
  leftMotorForward(duty*0.7);
  rightMotorForward(duty*1.1);
}

void turnLeftFast(float duty)
{
  leftMotorForward(duty*0);
  rightMotorForward(duty);
}

void turnLeftSlow(float duty)
{
  leftMotorForward(duty);
  rightMotorForward(duty*1.1);
}

void forwards(float duty)
{
  leftMotorForward(duty*1.1);  //wheels turned at different speeds for the same duty cycle - left wheel had to be tuned to acheive a straight line
  rightMotorForward(duty);
}

void backwards (float duty)
{
  leftMotorBackward(duty);
  rightMotorBackward(duty);
}

void Stop()
{
  rightMotorStop();
  leftMotorStop();
}

void spinClockwise(float duty)
{
  leftMotorForward(duty);
  rightMotorBackward(duty);
}

void spinAntiClockwise(float duty)
{
  leftMotorBackward(duty);
  rightMotorForward(duty);
}

void leftMotorForward(float duty)
{
  analogWrite(7, 0); //left motor backwards pin set to duty cycle of 0
  analogWrite(6, duty); //left motor forwards pin set to an inputted duty cycle

}

```

```
void leftMotorBackward(float duty)
{
  analogWrite(6, 0); //left motor forwards pin set to duty cycle of 0
  analogWrite(7, duty); //left motor backwards pin set to an inputted duty cycle
}

void leftMotorStop()
{
  analogWrite(6, 0); //left motor forwards pin set to duty cycle of 0
  analogWrite(7, 0); //left motor backwards pin set to duty cycle of 0
}
void rightMotorForward(float duty)
{
  analogWrite(2, 0);    //right motor backwards pin set to duty cycle of 0
  analogWrite(5, duty); //right motor forwards pin set to an inputted duty cycle
}

void rightMotorBackward(float duty)
{
  analogWrite(5, 0);    //right motor forwards pin set to duty cycle of 0
  analogWrite(2, duty); //right motor backwards pin set to an inputted duty cycle

}

void rightMotorStop()
{
  analogWrite(5, 0);  //right motor forwards pin set to duty cycle of 0
  analogWrite(2, 0);  //right motor backwards pin set to duty cycle of 0
}


```

Pin Map


AVR
INDEX CORNER
Digital pin 4 (PWM) (OC0B) PG5 1
Digital pin 0 (RX0) (RXD0/PCINT8) PE0 2
Digital pin 1 (TX0) (TXD0) PE1 3
(XCK0/AIN0) PE2 4
Digital pin 5 (PWM) (OC3A/AIN1) PE3 5
Digital pin 2 (PWM) (OC3B/INT4) PE4 6
Digital pin 3 (PWM) (OC3C/INT5) PE5 7
(T3/INT6) PE6 8
(CLKO/ICP3/INT7) PE7 9
VCC VCC 10
GND GND 11
Digital pin 17 (RX2) (RXD2) PH0 12
Digital pin 16 (TX2) (TXD2) PH1 13
(XCK2) PH2 14
Digital pin 6 (PWM) (OC4A) PH3 15
Digital pin 7 (PWM) (OC4B) PH4 16
Digital pin 8 (PWM) (OC4C) PH5 17
Digital pin 9 (PWM) (OC2B) PH6 18
Digital pin 53 (SS) (SS/PCINT0) PB0 19
Digital pin 52 (SCK) (SCK/PCINT1) PB1 20
Digital pin 51 (MOSI) (MOSI/PCINT2) PB2 21
Digital pin 50 (MISO) (MISO/PCINT3) PB3 22
Digital pin 10 (PWM) (OC2A/PCINT4) PB4 23
Digital pin 11 (PWM) (OC1A/PCINT5) PB5 24
Digital pin 12 (PWM) (OC1B/PCINT6) PB6 25
26 (OC0A/OC1C/PCINT7) PB7 Digital pin 13 (PWI
27 (T4) PH7
28 (TOSC2) PG3
29 (TOSC1) PG4
30 RESET RESET
31 VCC VCC
32 GND GND
33 XTAL2 XTAL2
34 XTAL1 XTAL1
35 (ICP4) PL0 Digital pin 49
36 (ICP5) PL1 Digital pin 48
37 (T5) PL2 Digital pin 47
38 (OC5A) PL3 Digital pin 46 (PWM)
39 (OC5B) PL4 Digital pin 45 (PWM)
40 (OC5C) PL5 Digital pin 44 (PWM)
41 PL6 Digital pin 43
42 PL7 Digital pin 42
43 (SCL/INT0) PD0 Digital pin 21 (SCL)
44 (SDA/INT1) PD1 Digital pin 20 (SDA)
45 (RXD1/INT2) PD2 Digital pin 19 (RX1)
46 (TXD1/INT3) PD3 Digital pin 18 (TX1)
47 (ICP1) PD4
48 (XCK1) PD5
49 (T1) PD6
50 (T0) PD7 Digital pin 38
51 PG0 (WR) Digital pin 41
52 PG1 (RD) Digital pin 40
53 PC0 (A8) Digital pin 37
54 PC1 (A9) Digital pin 36
55 PC2 (A10) Digital pin 35
56 PC3 (A11) Digital pin 34
57 PC4 (A12) Digital pin 33
58 PC5 (A13) Digital pin 32
59 PC6 (A14) Digital pin 31
60 PC7 (A15) Digital pin 30
61 VCC VCC
62 GND GND
63 PJ0 (RXD3/PCINT9) Digital pin 15 (RX3)
64 PJ1 (TXD3/PCINT10) Digital pin 14 (TX3)
65 PJ2 (XCK3/PCINT11)
66 PJ3 (PCINT12)
67 PJ4 (PCINT13)
68 PJ5 (PCINT14)
69 PJ6 (PCINT15)
70 PG2 (ALE) Digital pin 39
71 PA7 (AD7) Digital pin 29
72 PA6 (AD6) Digital pin 28
73 PA5 (AD5) Digital pin 27
74 PA4 (AD4) Digital pin 26
75 PA3 (AD3) Digital pin 25
76 PA2 (AD2) Digital pin 24
77 PA1 (AD1) Digital pin 23
78 PA0 (AD0) Digital pin 22
79 PJ7
80 VCC VCC
81 GND GND
82 PK7 (ADC15/PCINT23) Analog pin 15
83 PK6 (ADC14/PCINT22) Analog pin 14
84 PK5 (ADC13/PCINT21) Analog pin 13
85 PK4 (ADC12/PCINT20) Analog pin 12
86 PK3 (ADC11/PCINT19) Analog pin 11
87 PK2 (ADC10/PCINT18) Analog pin 10
88 PK1 (ADC9/PCINT17) Analog pin 9
89 PK0 (ADC8/PCINT16) Analog pin 8
90 PF7 (ADC7/TDI) Analog pin 7
91 PF6 (ADC6/TDO) Analog pin 6
92 PF5 (ADC5/TMS) Analog pin 5
93 PF4 (ADC4/TCK) Analog pin 4
94 PF3 (ADC3) Analog pin 3
95 PF2 (ADC2) Analog pin 2
96 PF1 (ADC1) Analog pin 1
97 PF0 (ADC0) Analog pin 0
98 AREF Analog Reference
99 GND GND
100 AVCC VCC