BACHELOR OF INFORMATION COMMUNICATION TECHNOLOGY
MAJORING IN SOFTWARE ENGINEERING

ICT 2104 - Embedded Systems Programming

TRIMESTER 1, ACADEMIC YEAR 2022/2023

Final Report

Team A2 (IR & Wheel Encoder) Submission

	Name	Student ID
1	Michael Ronny Chandiary	2101491
2	Gan Jia Xin	2101882
3	Tan Ai Xin	2102468

Table of Content

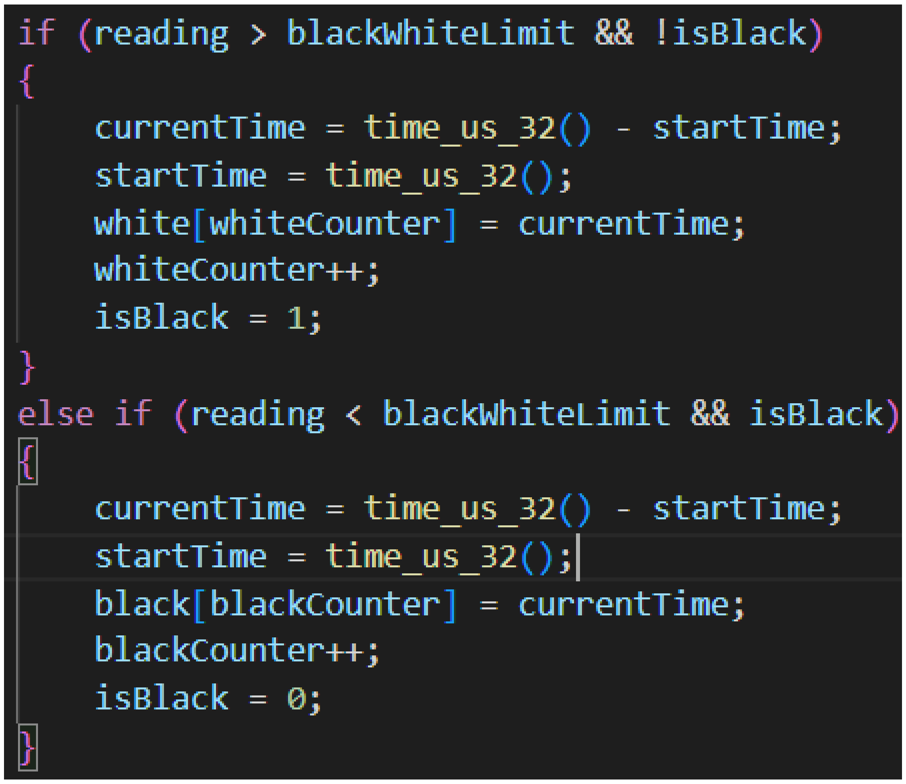
1. Barcode Logic **3**
1.1 Reading 3
1.2 Decoding 5
1.3 Filters Algorithm 7
1.3.1 Moving Average (MA) 7
1.3.2 Exponential Weighted Moving Average (EWMA) 8
1.3.3 ButterWorth 9
2. Encoder Logic **11**
3. Flow Chart **13**
3.1 Flow Chart For IR 13
3.2 Flow Chart for Wheel Encoder 14
4. State Diagram **15**
5. Black Box Testing for IR **16**
6. White Box Testing for Wheel Encoder **17**
6.1 get_current_speed() 17
6.2 wheel_encoder() 19
7. Video Links **21**

1. Barcode Logic

For the interpretation of the barcode, we have divided it into 2 main parts, Reading and Decoding. After reading the raw data and applying the filter to get better readings, we pass the data into the decoding part of the code.

1.1 Reading

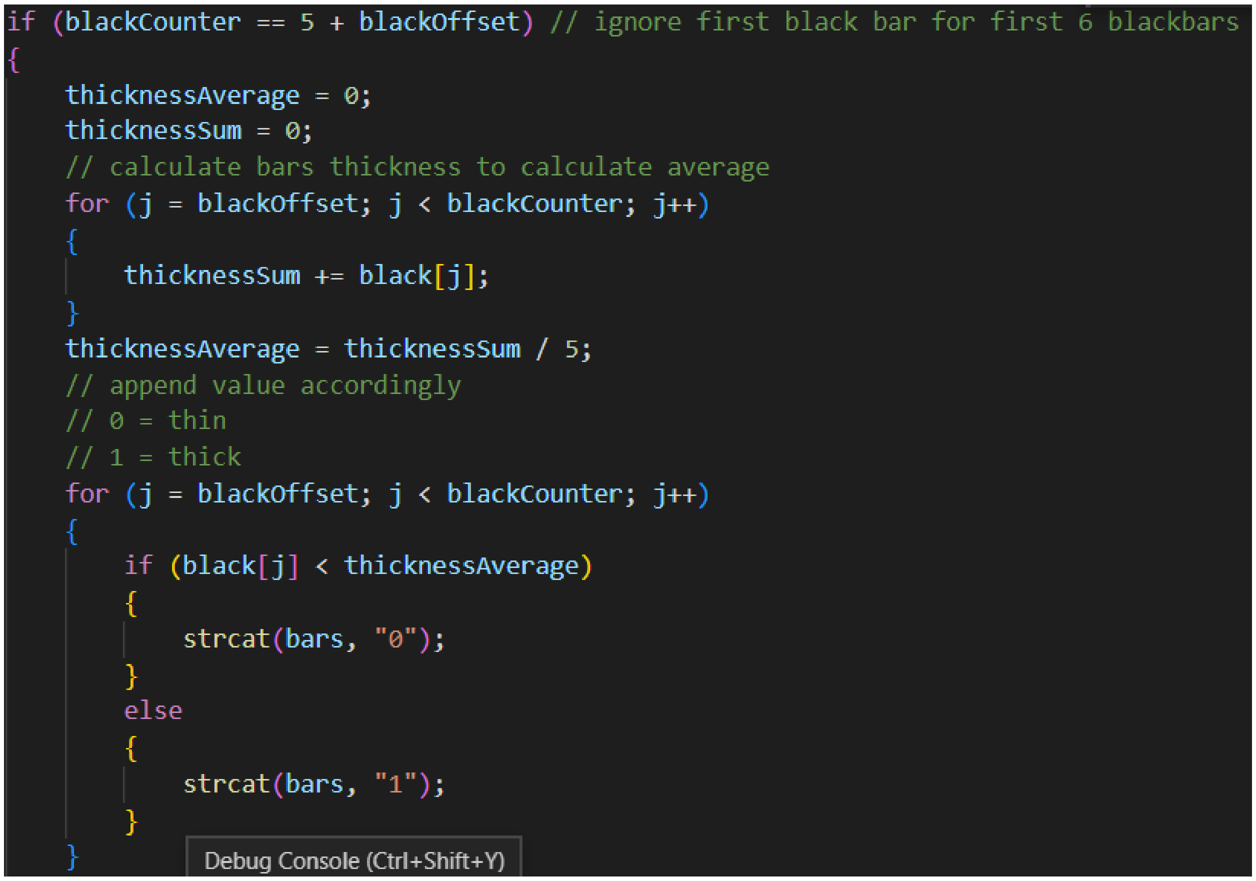
After triggering the interrupt for the ADC to convert the raw reading in the barcode_interrupt_exponential_weighted_filter function, an average of 100 readings to calculate the weighted reading is used. Calling the get_barcode would then start the understanding of the raw data.



```c
if (reading > blackWhiteLimit && !isBlack)
{
    currentTime = time_us_32() - startTime;
    startTime = time_us_32();
    white[whiteCounter] = currentTime;
    whiteCounter++;
    isBlack = 1;
}
else if (reading < blackWhiteLimit && isBlack)
{
    currentTime = time_us_32() - startTime;
    startTime = time_us_32();
    black[blackCounter] = currentTime;
    blackCounter++;
    isBlack = 0;
}
```

Figure 1.1.1: Store Data Into Array

In the get_barcode function, each bar/space is determined by having a currentTime - startTime to determine how long each bar/space would take. We set the startTime when the first ADC conversion occurs. Whenever there is a change in reading between black and white, the time for the previous black/white will be taken down and stored in the corresponding array. The startTime would then be resetted.



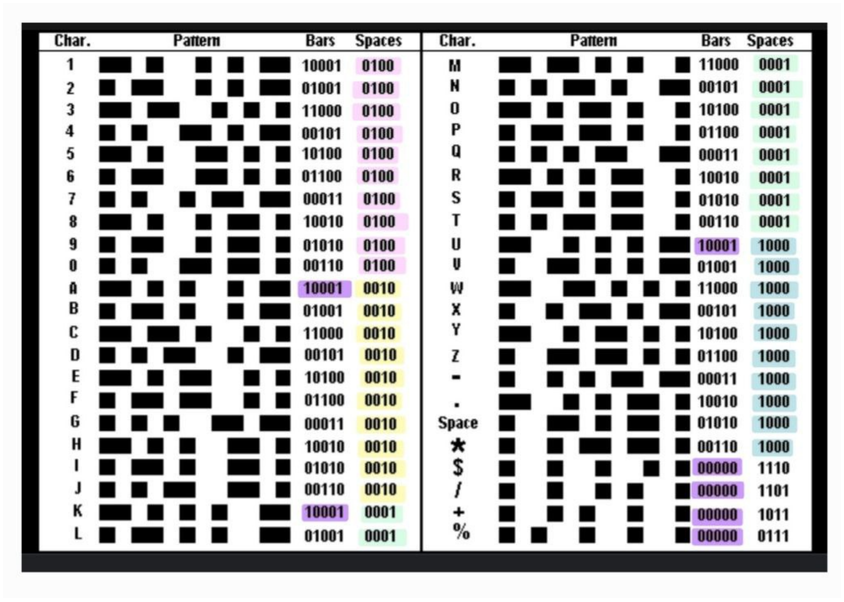
```
if (blackCounter == 5 + blackOffset) // ignore first black bar for first 6 blackbars
{
    thicknessAverage = 0;
    thicknessSum = 0;
    // calculate bars thickness to calculate average
    for (j = blackOffset; j < blackCounter; j++)
    {
        thicknessSum += black[j];
    }
    thicknessAverage = thicknessSum / 5;
    // append value accordingly
    // 0 = thin
    // 1 = thick
    for (j = blackOffset; j < blackCounter; j++)
    {
        if (black[j] < thicknessAverage)
        {
            strcat(bars, "0");
        }
        else
        {
            strcat(bars, "1");
        }
    }
```

Figure 1.1.2: Outputting Binary

Since the sequence of reading the barcode would be black -> white -> barcode -> white -> barcode -> white -> barcode -> end, the first reading of black and white would be ignored, thus the offset of black and white is used. The total time taken for each set of 5 bars and 4 spaces would be taken to find the average time. This average time would be to determine if a bar is thick or thin. (below average = thin, above average = thick) After outputting at least 5 bars and 4 spaces, these 2 strings will be passed into the decode function.

1.2 Decoding

For the decoding, we first figure out the pattern of Code 39. Below is a screenshot of how the pattern works.



Char.	Pattern	Bars	Spaces	Char.	Pattern	Bars	Spaces
1		10001	0100	M		11000	0001
2		01001	0100	N		00101	0001
3		11000	0100	O		10100	0001
4		00101	0100	P		01100	0001
5		10100	0100	Q		00011	0001
6		01100	0100	R		10010	0001
7		00011	0100	S		01010	0001
8		10010	0100	T		00110	0001
9		01010	0100	U		10001	1000
0		00110	0100	V		01001	1000
A		10001	0010	W		11000	1000
B		01001	0010	X		00101	1000
C		11000	0010	Y		10100	1000
D		00101	0010	Z		01100	1000
E		10100	0010	-		00011	1000
F		01100	0010	.		10010	1000
G		00011	0010	Space		01010	1000
H		10010	0010	*		00110	1000
I		01010	0010	$		00000	1110
J		00110	0010	/		00000	1101
K		10001	0001	+		00000	1011
L		01001	0001	%		00000	0111

Figure 1.2.1: Code 39

From figure 1, we can see that each of the barcodes is split into bars and spaces. Thus, for the decoding function, we will parse in the values for bars and spaces.

There are a few patterns in Code 39 that I have spotted. One of the more obvious patterns is in the space section. The 4 main spaces category will be {0100, 0010, 0001, 1000}. Another pattern that I have spotted is in the bar section. In each of the 4 main categories of spaces, the bar value will always repeat from 10001(Number 1) to 00110 (Number 10). The last pattern will be when the bar value is 0000. There are 4 space values in that bar. However, we will not add this pattern to the array for this project.

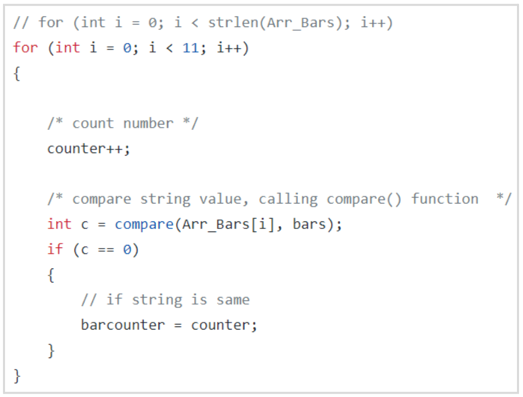
In summary, this is how the arrays will look like:

```
char Arr_Characters[NUMBER_OF_STRING][MAX_STRING_SIZE] = {
    "1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "A", "B", "C", "D", "E",
    "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T",
    "U", "V", "W", "X", "Y", "Z", "-", ".", "_", "*", "$", "/", "+", "%"};
```

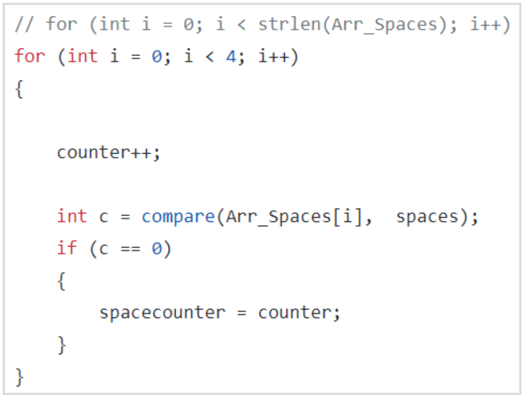
```
char Arr_Spaces[NUMBER_OF_STRING][MAX_STRING_SIZE] =
{"0100", "0010", "0001", "1000"};
```

```
char Arr_Bars[NUMBER_OF_STRING][MAX_STRING_SIZE] =
{"10001", "01001", "11000", "00101", "10100", "01100", "00011", "10010", "01010", "00110", "00000"};
```

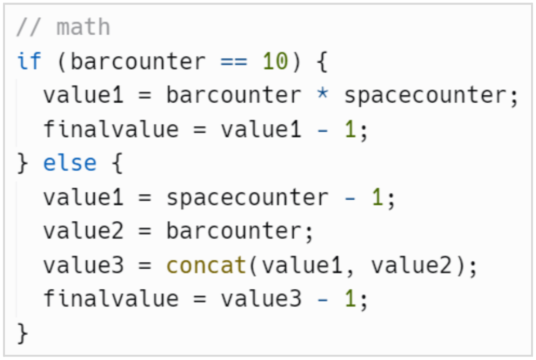
In the decode_barcode function, we first check the array position of the bar. We will only check the part where the barcounter falls in 1 to 10, and ignore when it is 11. We will then store the bar counter value and also run another for loop to check the value of the space counter and store this value. Once we know the position of the bars and spaces array, we can then calculate the Arr_Characters value.



```
// for (int i = 0; i < strlen(Arr_Bars); i++)
for (int i = 0; i < 11; i++)
{

    /* count number */
    counter++;

    /* compare string value, calling compare() function  */
    int c = compare(Arr_Bars[i], bars);
    if (c == 0)
    {
        // if string is same
        barcounter = counter;
    }
}
```



```
// for (int i = 0; i < strlen(Arr_Spaces); i++)
for (int i = 0; i < 4; i++)
{

    counter++;

    int c = compare(Arr_Spaces[i],  spaces);
    if (c == 0)
    {
        spacecounter = counter;
    }
}
```



```
// math
if (barcounter == 10) {
  value1 = barcounter * spacecounter;
  finalvalue = value1 - 1;
} else {
  value1 = spacecounter - 1;
  value2 = barcounter;
  value3 = concat(value1, value2);
  finalvalue = value3 - 1;
}
```

Math Logic

To find the value of the Arr_Characters array position, there will be two different methods to use. The first method is when the bar counter is 10 (the last value of bar). This will then run the IF part of the code.
Formular: (barcounter * spacecounter) - 1

Next method is when the bar counter is from 1 to 9. This will run the ELSE part of the code.
Formular:
First Digit: space counter - 1
Second Digit: barcounter
Final Value (Array Value):
(First Digit)(Second Digit) - 1

Once the Calculation is completed, we can then use Arr_Characters[finalvalue] to get the actual value from that array.

1.3 Filters Algorithm

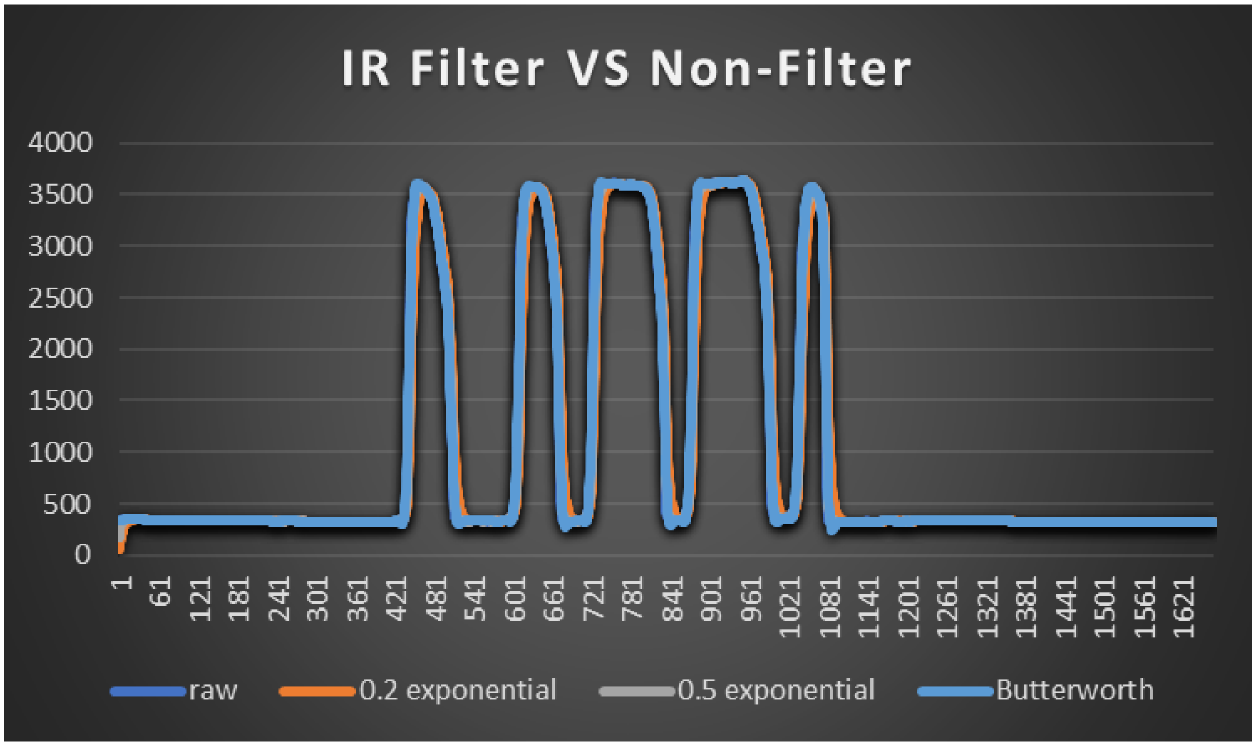
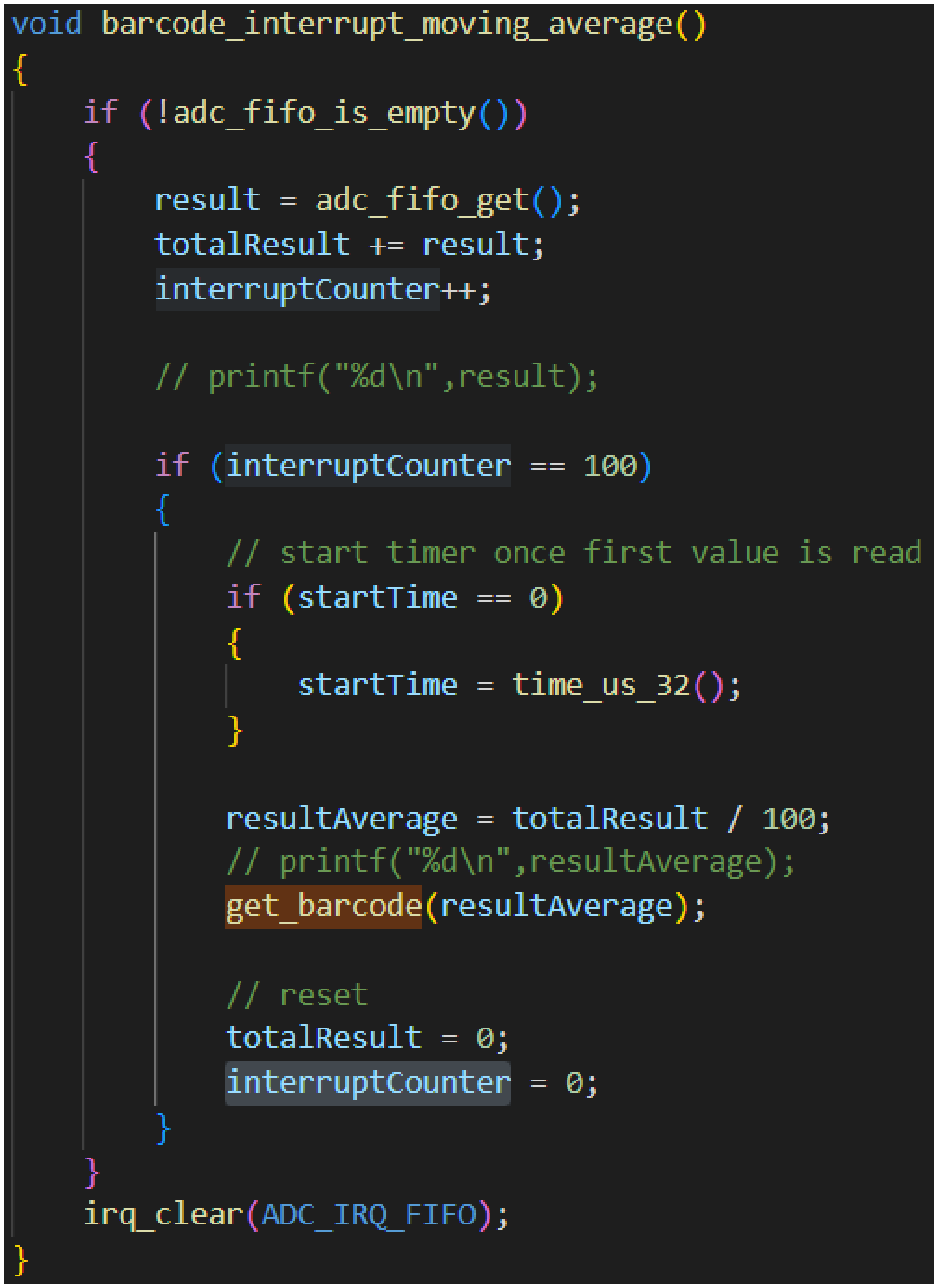


Figure 1.3.1: Filter Algorithm for Barcode

For the filtering of raw data for the barcode, we have tried Moving Average, Weighted Moving Average, and ButterWorth. Through the filters that we tried, we decided to use the EWMA as it helps to smoothen the graph by a bit with a 0.2 weight.

1.3.1 Moving Average (MA)

The moving average filter helps to smoothen the curve by taking a few values and finding the average value of 100 readings. Doing this, even if there is a sudden change in readings/noise, the interpretation of the input will still be accurate.

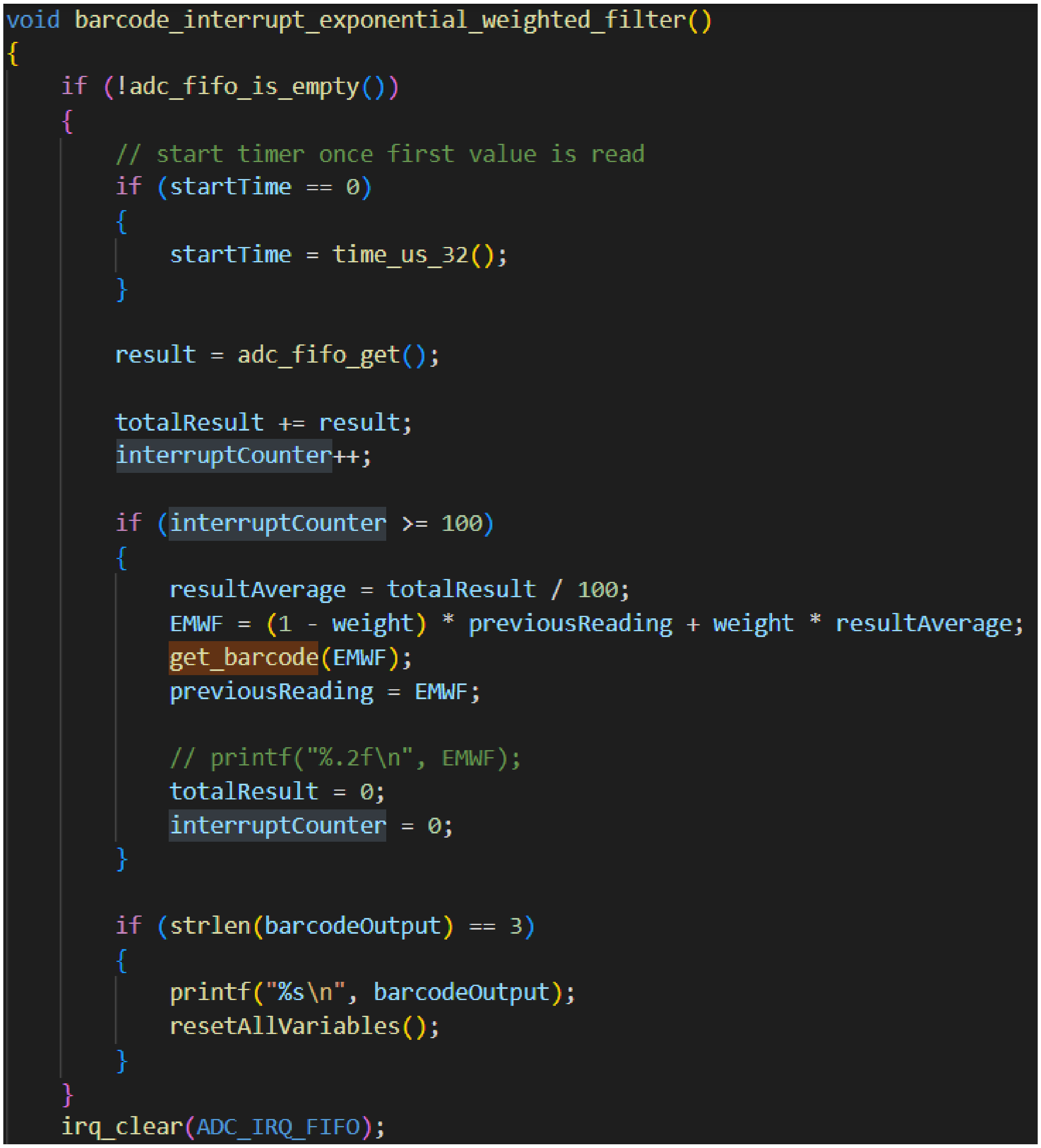


```
void barcode_interrupt_moving_average()
{
    if (!adc_fifo_is_empty())
    {
        result = adc_fifo_get();
        totalResult += result;
        interruptCounter++;

        // printf("%d\n",result);

        if (interruptCounter == 100)
        {
            // start timer once first value is read
            if (startTime == 0)
            {
                startTime = time_us_32();
            }

            resultAverage = totalResult / 100;
            // printf("%d\n",resultAverage);
            get_barcode(resultAverage);

            // reset
            totalResult = 0;
            interruptCounter = 0;
        }
    }
    irq_clear(ADC_IRQ_FIFO);
}
```

Figure 1.3.1.1: Moving Average Filter

1.3.2 Exponential Weighted Moving Average (EWMA)

For the EWMA filter that was implemented, we decided to use a weight of 0.2 that helps to smoothen out the reading better than the moving average.

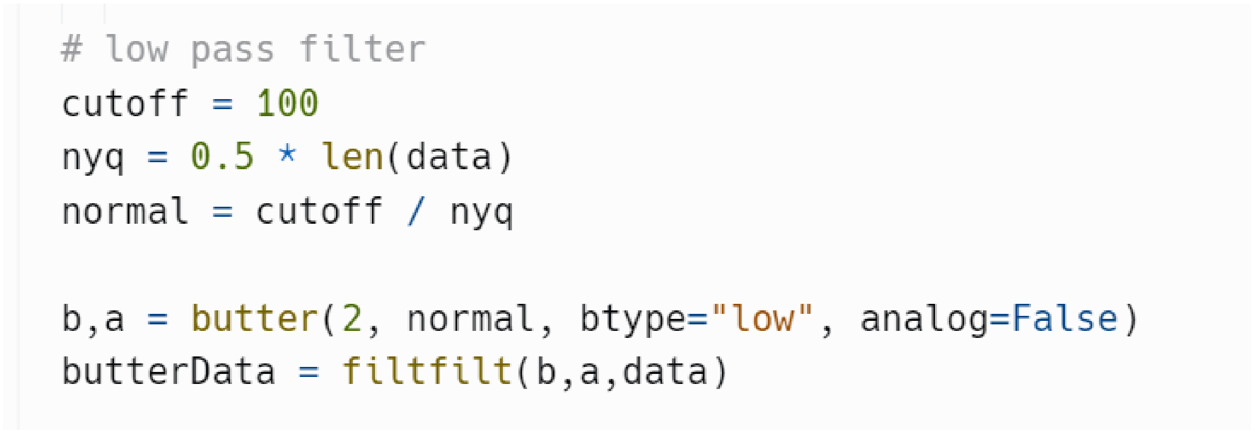


```c
void barcode_interrupt_exponential_weighted_filter()
{
    if (!adc_fifo_is_empty())
    {
        // start timer once first value is read
        if (startTime == 0)
        {
            startTime = time_us_32();
        }

        result = adc_fifo_get();

        totalResult += result;
        interruptCounter++;

        if (interruptCounter >= 100)
        {
            resultAverage = totalResult / 100;
            EMWF = (1 - weight) * previousReading + weight * resultAverage;
            get_barcode(EMWF);
            previousReading = EMWF;

            // printf("%.2f\n", EMWF);
            totalResult = 0;
            interruptCounter = 0;
        }

        if (strlen(barcodeOutput) == 3)
        {
            printf("%s\n", barcodeOutput);
            resetAllVariables();
        }
    }
    irq_clear(ADC_IRQ_FIFO);
```

Figure 1.3.2.1: Weighted Moving Average Filter

1.3.3 ButterWorth

Our team tried to apply a butterworth filter to eliminate peaks of each pass band for the analogue reading. We use a sampingling rate of 100, then proceed to calculate the nyquist frequency and the normalized frequency. This filter is done in python to test if it is better.



```python
# low pass filter
cutoff = 100
nyq = 0.5 * len(data)
normal = cutoff / nyq

b,a = butter(2, normal, btype="low", analog=False)
butterData = filtfilt(b,a,data)
```

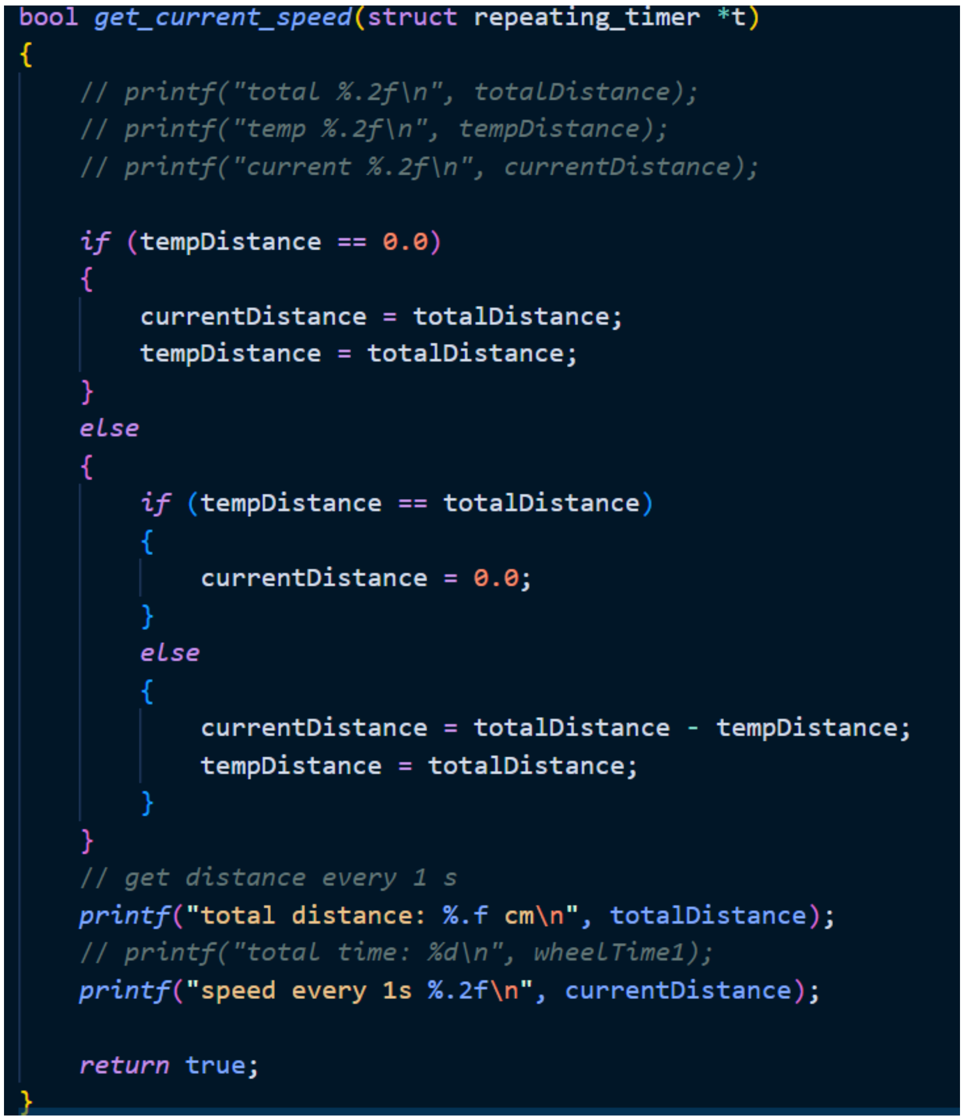
Figure 1.3.3.1: Weighted Moving Average Filter

2. Encoder Logic

```
if (gpio == 17)
    {
        wheelRotation1++; // right wheel
        wheelDistance1 = wheelRotation1 * 0.165 * PI;
    }
    if (gpio == 14)
    {
        wheelRotation2++;                            // left wheel
        wheelDistance2 = wheelRotation2 * 0.165 * PI; // d = 6.6cm, distance =
notches * 6.6pi / 40
    }

totalDistance = ((wheelDistance2 + wheelDistance1) / 2.0);
```

Firstly, we divide the total distance for both wheels separately. Then, we add both distances together and divide by 2 to get the average distance. By calculating the average, we can get the actual distance of the car even if one of the wheels turns faster than the other.

As we are reading the pulse every interrupt, the formula to get the total distance for each wheel is number of rotation * circumference of the car per pulse.



```c
bool get_current_speed(struct repeating_timer *t)
{
    // printf("total %.2f\n", totalDistance);
    // printf("temp %.2f\n", tempDistance);
    // printf("current %.2f\n", currentDistance);

    if (tempDistance == 0.0)
    {
        currentDistance = totalDistance;
        tempDistance = totalDistance;
    }
    else
    {
        if (tempDistance == totalDistance)
        {
            currentDistance = 0.0;
        }
        else
        {
            currentDistance = totalDistance - tempDistance;
            tempDistance = totalDistance;
        }
    }
    // get distance every 1 s
    printf("total distance: %.f cm\n", totalDistance);
    // printf("total time: %d\n", wheelTime1);
    printf("speed every 1s %.2f\n", currentDistance);

    return true;
}
```

Figure 2.1: get_current_speed()

In the get_current_speed() function, the speed will be calculated every second and stored into the currentDistance variable. When the car moves, the distance increases and the current distance will be updated to the total distance the car traveled minus the previous distance (tempDistance). When the car stops moving, the distance is not increasing / decreasing and the system checks if the total distance matches the previous distance. If it matches, the current distance would be clear to 0 and the current speed would be updated as 0.

3. Flow Chart

3.1 Flow Chart For IR

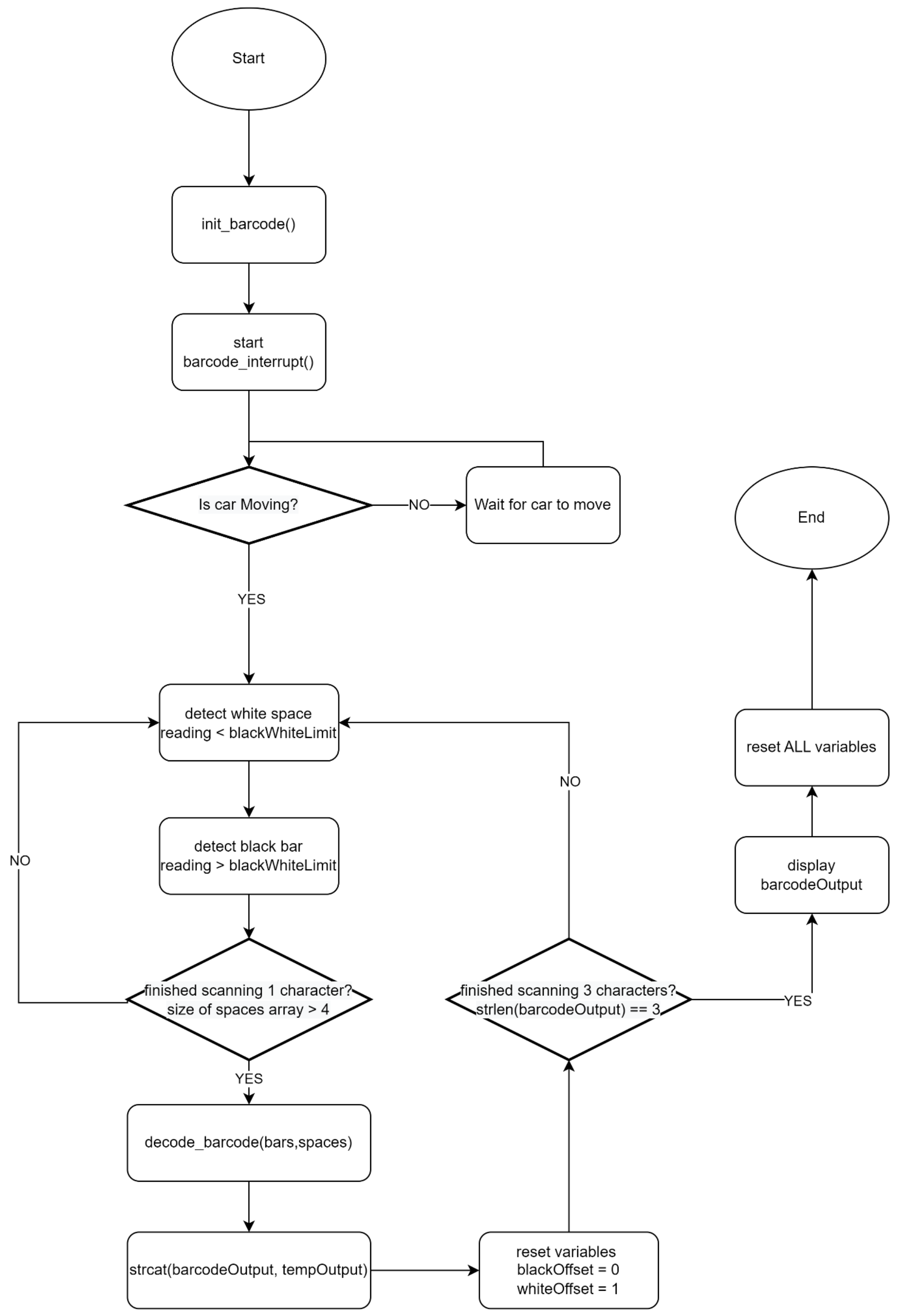

Start
init_barcode()
start barcode_interrupt()
Is car Moving?
NO
Wait for car to move
YES
End
detect white space reading < blackWhiteLimit
NO
reset ALL variables
detect black bar reading > blackWhiteLimit
NO
display barcodeOutput
finished scanning 1 character? size of spaces array > 4
finished scanning 3 characters? strlen(barcodeOutput) == 3
YES
YES
decode_barcode(bars,spaces)
strcat(barcodeOutput, tempOutput)
reset variables blackOffset = 0 whiteOffset = 1

Figure 3.1.1: IR Flow Chart

3.2 Flow Chart for Wheel Encoder

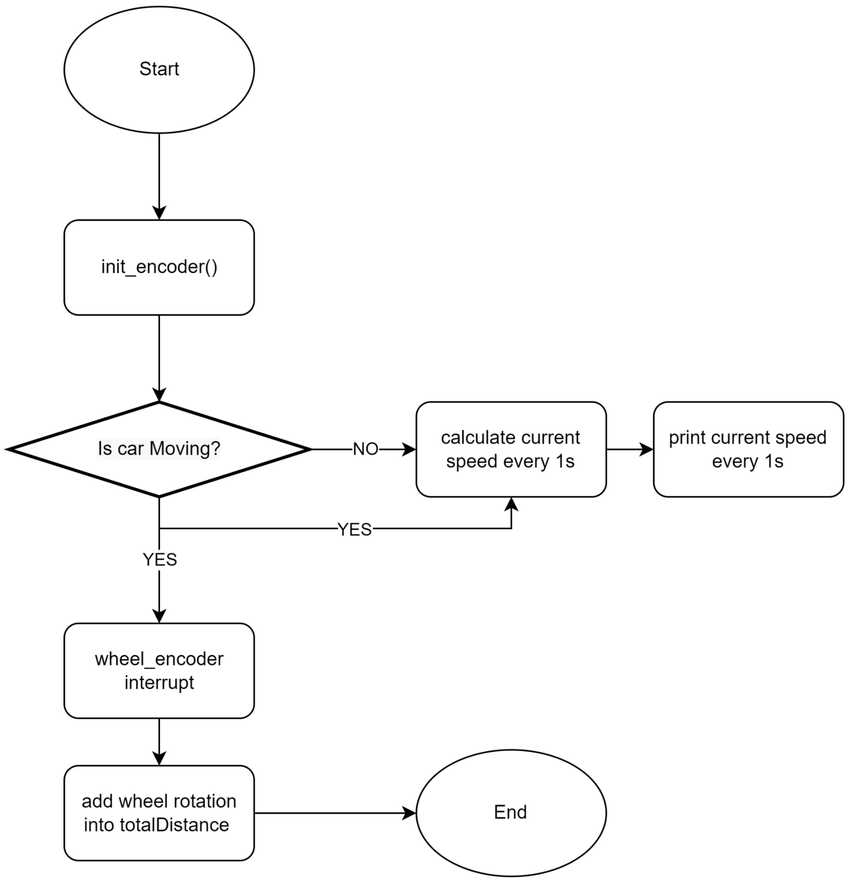

Start
init_encoder()
Is car Moving?
NO
calculate current speed every 1s
print current speed every 1s
YES
YES
wheel_encoder interrupt
add wheel rotation into totalDistance
End

Figure 3.2.1: Wheel Encoder Flow Chart

4. State Diagram

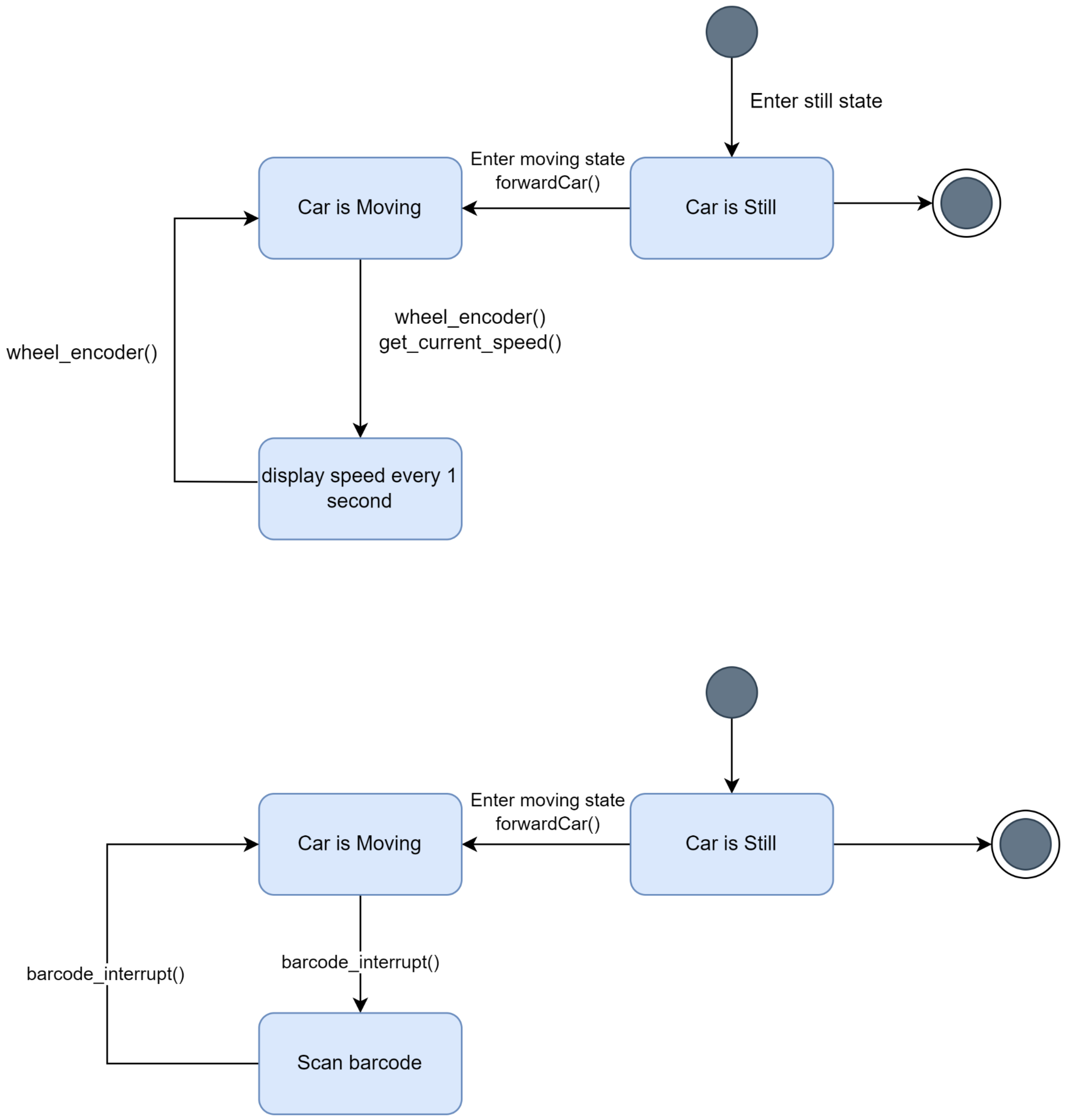
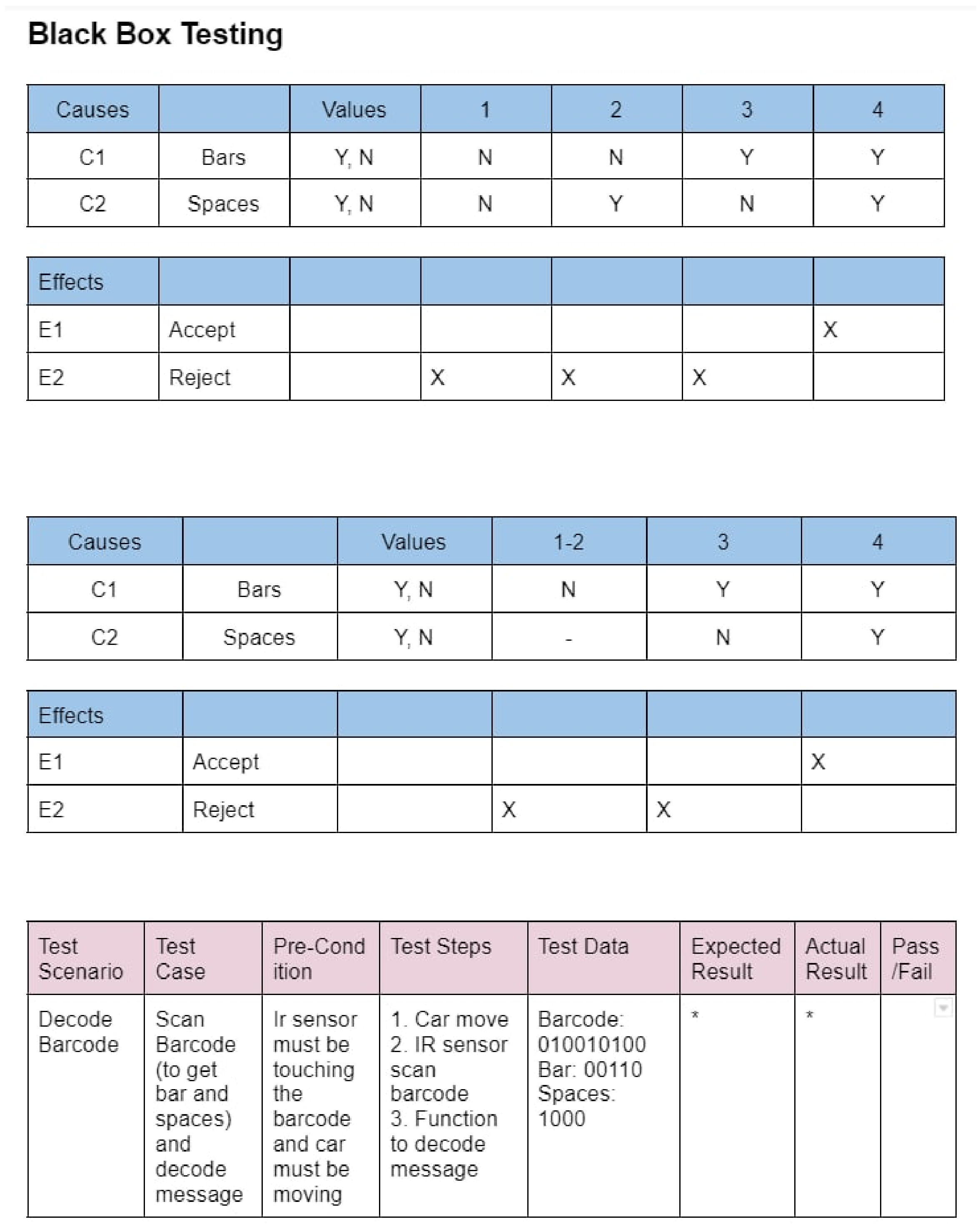


Figure 4.1: IR and Wheel Encoder State Diagram

5. Black Box Testing for IR



Black Box Testing

Causes		Values	1	2	3	4
C1	Bars	Y, N	N	N	Y	Y
C2	Spaces	Y, N	N	Y	N	Y

Effects						
E1	Accept					X
E2	Reject		X	X	X	

Causes		Values	1-2	3	4
C1	Bars	Y, N	N	Y	Y
C2	Spaces	Y, N	-	N	Y

Effects					
E1	Accept				X
E2	Reject		X	X	

Test Scenario	Test Case	Pre-Condition	Test Steps	Test Data	Expected Result	Actual Result	Pass /Fail
Decode Barcode	Scan Barcode (to get bar and spaces) and decode message	Ir sensor must be touching the barcode and car must be moving	1. Car move 2. IR sensor scan barcode 3. Function to decode message	Barcode: 010010100 Bar: 00110 Spaces: 1000	*	*	

Figure 5.1: Black Box Testing for IR

6. White Box Testing for Wheel Encoder

6.1 get_current_speed()

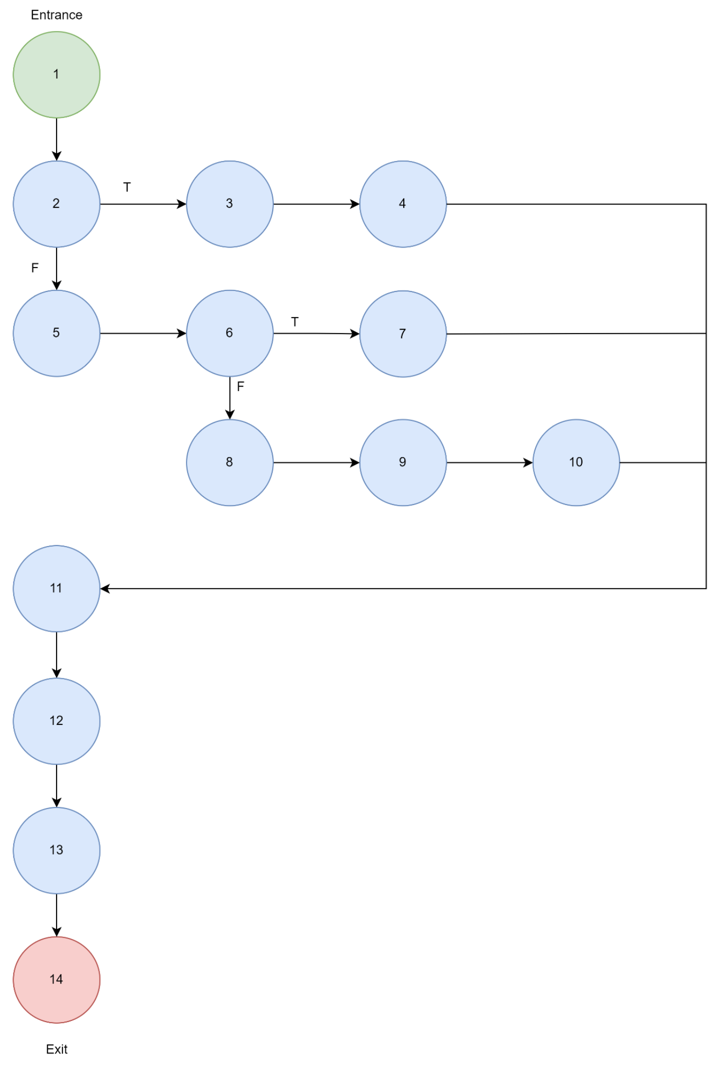


Figure 6.1.1: get_current_speed() CFG

Formular for Cyclomatic Complexity of CFG (M = E - N + 2P)
We have 14 nodes, 16 edges, M = 16 - 14 + 2 = 4
Giving us 4 possible paths.
Path 1: 1-2-3-4-11-12-13-14
Path 2: 1-2-5-6-7-11-12-13-14
Path 3: 1-2-5-6-8-9-10-11-12-13-14

Code:

```
bool get_current_speed(struct repeating_timer *t) {
  if (tempDistance == 0.0) {
      currentDistance = totalDistance;
      tempDistance = totalDistance; }
  else {
      if (tempDistance == totalDistance) {
          currentDistance = 0.0; }
      else {
          currentDistance = totalDistance - tempDistance;
            tempDistance = totalDistance;}}
          printf("total distance: %.f cm\n", totalDistance);
          printf("speed per second: %.2fcm/s \n", currentDistance);
   return true;
}
```

#	Pre - Conditions	Variables	Expected Result
1	Car is not moving	total distance = 0, current distance = 0, temp distance = 0	"total distance: 0.0cm" & "speed per second: 0.00 cm/s "
2	Car is moving	total distance > 0, current distance > 0, temp distance != total distance	"total distance: x cm" & "speed per second: x cm/s "
3	Car stop moving	total distance > 0, current distance = 0, temp distance = total distance	"total distance: x cm" & "speed per second: 0 cm/s "

Table 6.1.1: get_current_speed() Test Case Table

6.2 wheel_encoder()

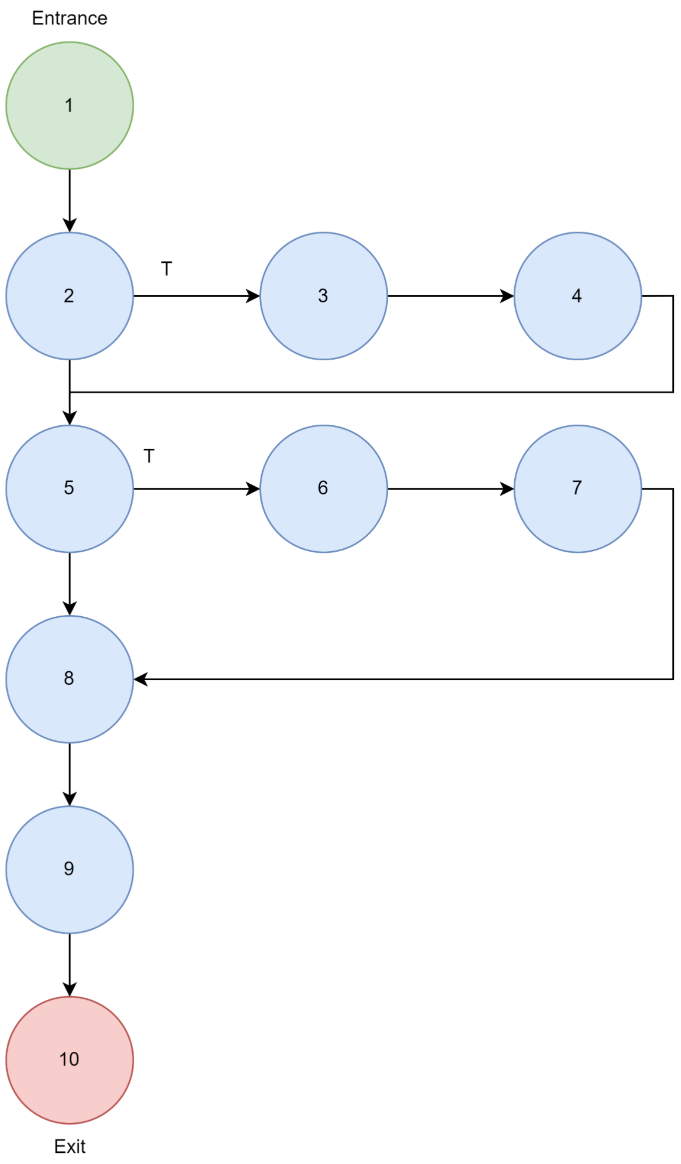


Figure 6.2.1: wheel_encoder() CFG

Formular for Cyclomatic Complexity of CFG (M = E - N + 2P)
We have 10 nodes, 12 edges, M = 12 - 10 + 2 = 4
Giving us 4 possible paths.
Path 1: 1-2-3-4-5-6-7-8-9-10
Path 2: 1-2-3-4-5-8-9-10
Path 3: 1-2-5-6-7-8-9-10
Path 4: 1-2-5-8-9-10

Code:

```
static void wheel_encoder(unsigned gpio, long unsigned int _){
    if (gpio == 17){
        wheelRotation1++; // right wheel
        wheelDistance1 = wheelRotation1 * 0.33 * PI;}
    if (gpio == 14) {
        wheelRotation2++;
        wheelDistance2 = wheelRotation2 * 0.33 * PI; }
    totalDistance = (wheelDistance1 + wheelDistance2) / 2
    irq_clear(PIO0_IRQ_0);
}
```

#	Pre - Conditions	Variables	Expected Result
1	Both GPIO detected, Car is moving	wheelRotation1 > 0, wheelRotation2 > 0, total distance > 0	total distance
2	One GPIO detected, Car is moving	wheelRotation1 = 0, wheelRotation2 > 0, total distance > 0	total distance
3	One GPIO detected, Car is moving	wheelRotation1 > 0, wheelRotation2 = 0, total distance > 0	total distance
4	Both GPIO not detected, Car is moving	wheelRotation1 > 0, wheelRotation2 = 0, total distance = 0	total distance

Table 6.2.1: wheel_encoder() Test Case Table

7. Video Links

Demo Video:
https://youtu.be/KjQtN3TZ5xo