TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

Figures

Tables

1. Overall Algorithm (Pseudocode)

As shown in Figure 1 below, when the mBot is turned on and a button pressed, it immediately begins moving forward. Upon detecting a black strip below, it stops. The LED is activated, which then scans for a colour above the mBot. Depending on the colour it detects, apart from black, the mBot's motor will cause it to move in a certain direction. If the LED scans black, the microphone on the mBot will be activated. The mBot will then scan for an audio. If a high or low frequency is played, the mBot will again move accordingly. If no audio is played, i.e. the mBot has reached the end of the maze, a victory sound will be played. IR sensors were used to prevent the mBot from colliding with the boundaries and keep it on a straight path.

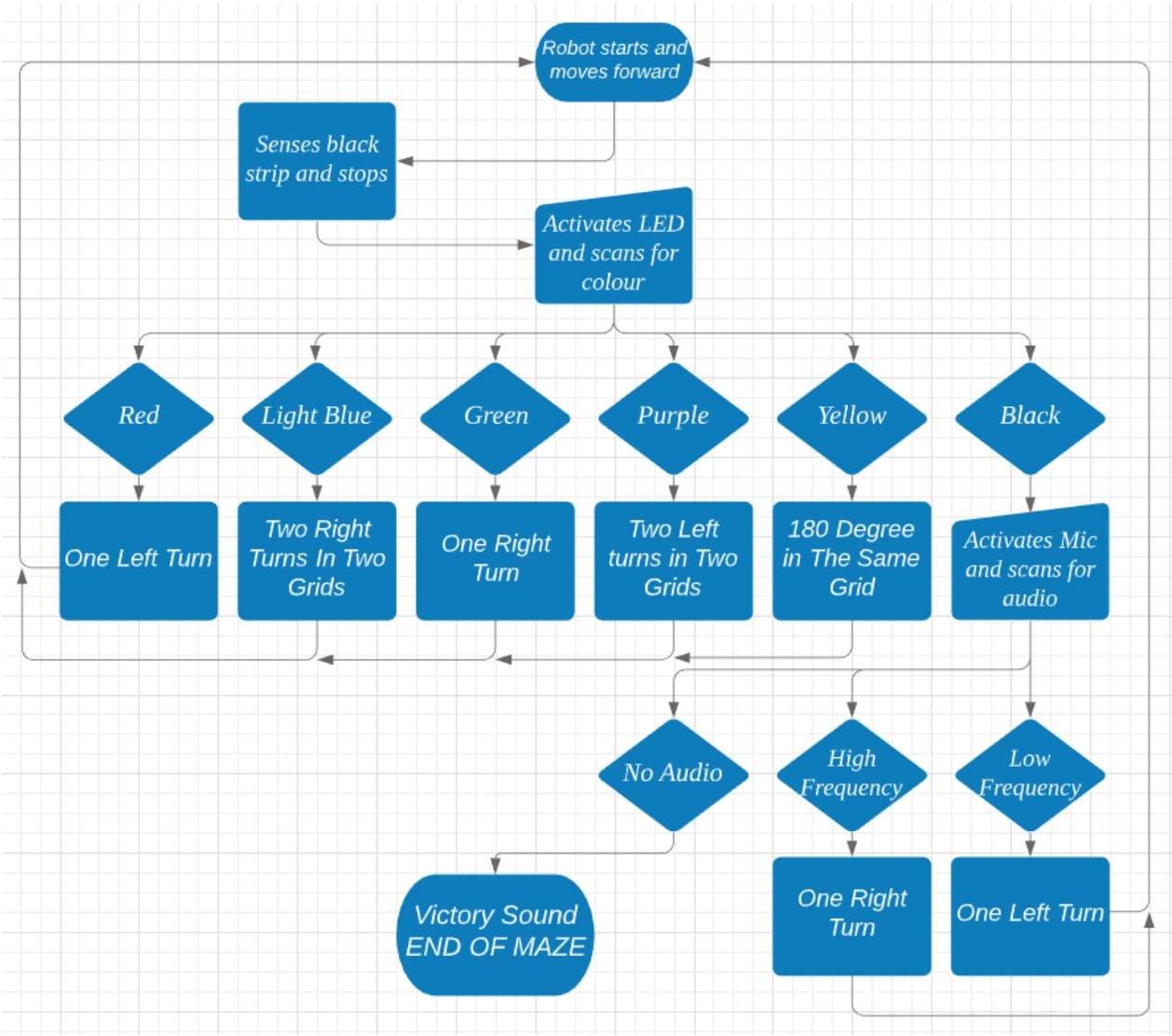

Figure 1: Algorithm Pseudocode

2. Project Design Details

2.1 Keeping mBot Straight

InfraRed (IR) sensors ensured our mBot moved in a straight path, without colliding into any potential obstructions (e.g. white tiles). We defined a variable, IRDISTSPACE, to be of value 3.2, which represents the voltage threshold, corresponding to the minimum distance our mBot should be from any obstruction. As the IR sensor gets closer to the tiles, voltage increases. lVoltage and rVoltage, obtain voltage outputs from the left and right IR sensors respectively via analogRead(). The IR sensors (on breadboards) are attached to the left and right sides of the mBot as shown in Figure 2 with the specific circuit design shown in Figure 3.

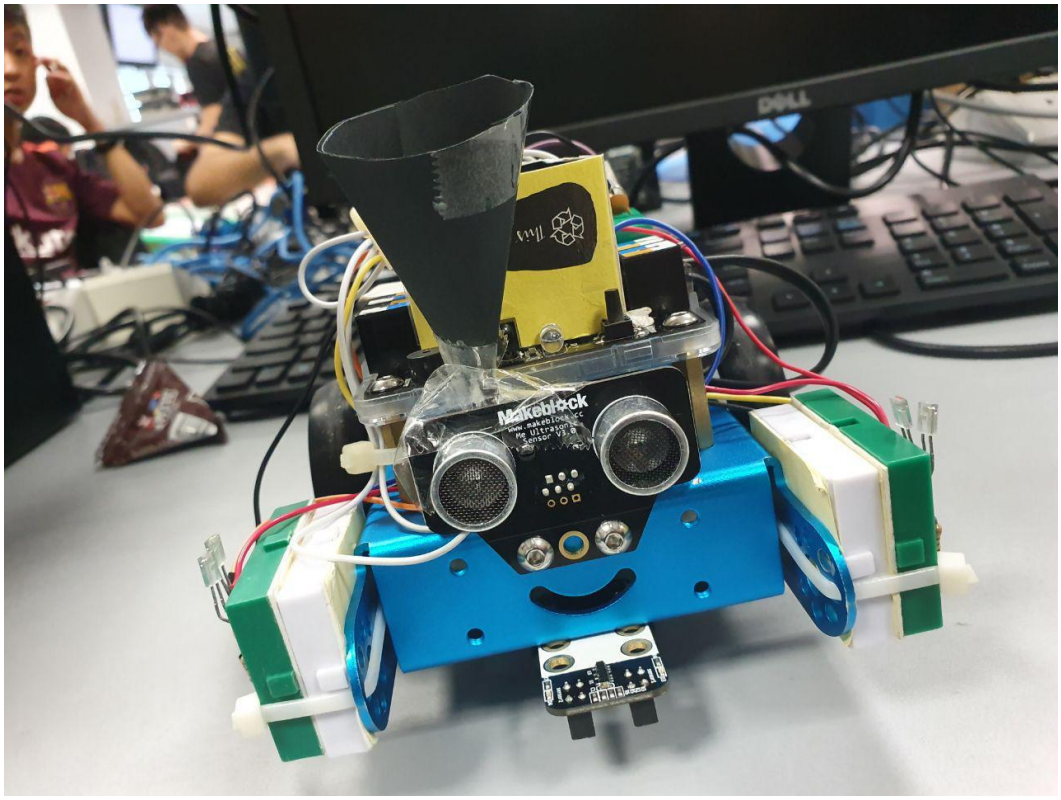

Figure 2: IR Sensors on the mBot

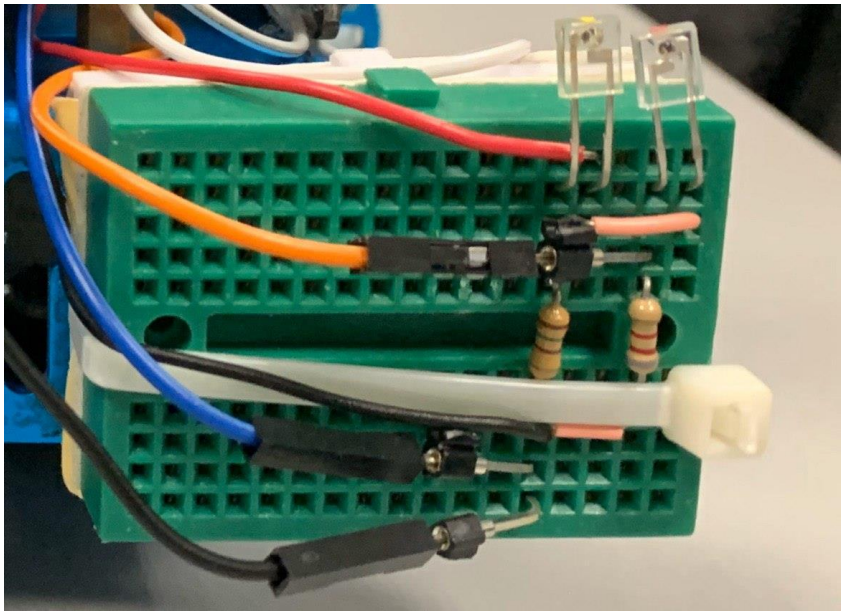

Figure 3: IR Sensor Circuit

If the mBot's IR sensor on the left side returns a voltage exceeding 3.2V, it means that it senses that it's left side is too close to an obstacle. Hence, the mBot will turn right until voltage read is within 3.2V. To do so, we stop the right wheel's motor, and allow the left wheel to continue moving (at a slower speed) so the robot moves right. Likewise, for the right side.

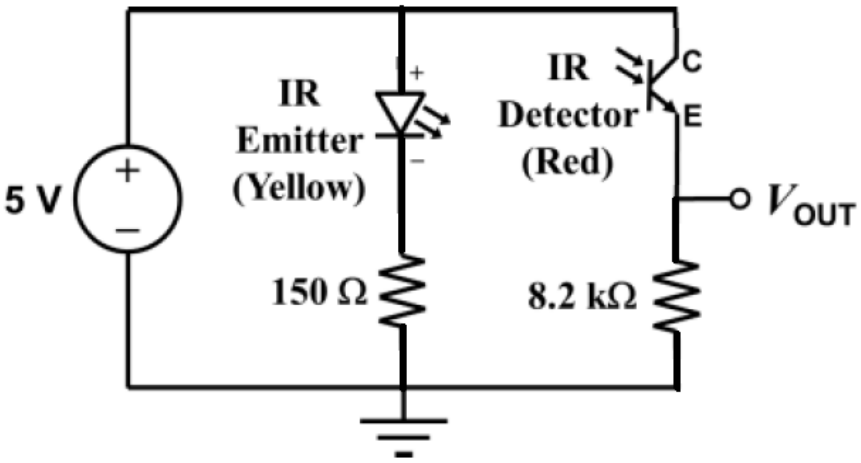

Figure 4: IR Sensor Circuit Diagram

Our IR circuit layout was different from the one we built in our studio. As shown in Figure 4 above, we designed it such that when the wall is closer the voltage increases. We felt that it is more intuitive and easier to code. Based on the results we got during studio, we observed that 3.2V is the optimal range. As shown in Figure 5, a distance of less than 5cm shows distinct changes in voltage.

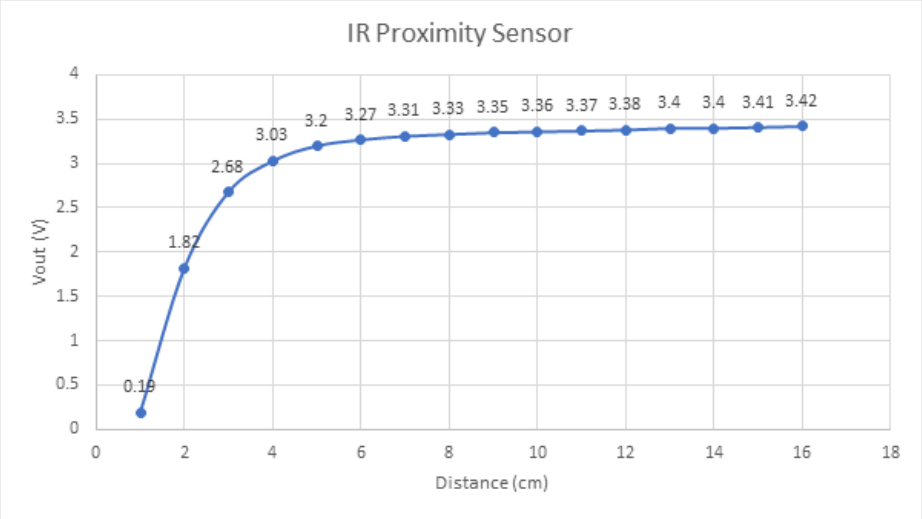

Figure 5: IR Sensor's Voltage vs Distance Graph

2.2 Colour Sensing Algorithms

The LED shone Red, Green and Blue (RGB) light onto the coloured paper. Based on the colour of the paper, it absorbs some amount of that colour light and reflect the rest of the light back to the LDR. The difference in intensity of the reflected light gave us the information we need to determine the colour of the paper. We stored the input from the LDR into an array of RGB values. The values were inputted into an RGB calculator to get the actual colour of the paper. After many rounds of obtaining the LDR readings, we came up with a range of RGB values that enabled us to determine the colour of the paper. For example, a black colour absorbs all light. Even though the RGB values should have been 0, we gave a buffer for inaccurate readings. Hence, each value of RGB was below 50 (out of 255). From the studios, we can see ambient light affects colour detection. In order to get accurate RGB values, we had to factor in the values of ambient light. We can do so by getting white and black colour paper RGB values. The white and black values give the contrast of the colour sensing. In order to save time for the race, we separated the measurement of white and black, and calibrated the white and black values. Once black and white calibrations were completed, our mBot was ready to sense different colours.

In our movement function, we utilised switch case, to allow the mBot to make turns corresponding to the colour it senses, after sensing a black strip below (refer to Figure 1 above). We did this through our movements() function. For this to work, we also nested a turns() function within movements(), which controls both motors 1 and 2, as well as the number of turns the mBot makes.

This turn() function controls:

1. Turn direction i.e. left or right turn
2. Number of turns e.g. 1 or 2

For 1 turn:

For the mBot to make one left turn, we set motors 1 and 2 to MOTORSPEED. This causes the left wheel to move backwards, and the right wheel forward. Similarly, for the mBot to make one right turn, we set motors 1 and 2 to -MOTORSPEED. This causes the left wheel to move forward, and the right wheel backwards.

For 2 turns (in the same grid):

The same applies in terms of direction. However, for the 180-degree turn, we increase the delay time. More specifically, we increased our UTURN delay time from 225 (LEFTTURN and RIGHTTURN time delay) to 470.

For 2 turns (over two grids):

The same applies in terms of direction. However, we did a 90-degree turn first. Following the turn, the mBot moved forward. We then activated the ultrasonic sensor. After numerous tries, experimentally we deduced that 12cm was a good threshold and so as long as the distance between the mBot and the white tile was greater than 12cm, it would continue to move forward. When the threshold was met, the mBot would stop and then make the second 90-degree turn.

2.3 Audio Processing Circuit Design and Algorithms

2.3.1 Building the High-Pass Filter

We first connect our microphone to a 5V power supply and the ground. The mBot's microphone detects a tune of a certain frequency, producing a differential voltage. By using an R_f value of 400k and Ri value of 1k, a gain of 401 will amplify the differential voltage, giving us an output voltage. The envelope detector then demodulates the high frequency. This is illustrated in Figure 6 below. Our actual circuit is represented in Figure 7.

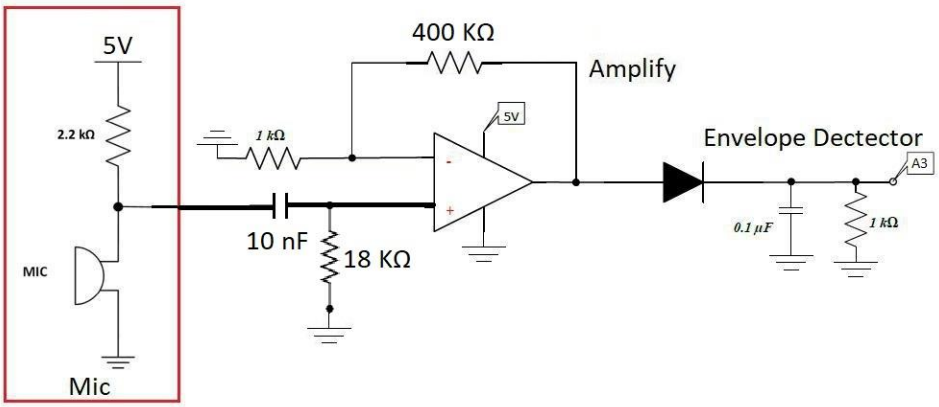

Figure 6: Microphone Circuit Diagram

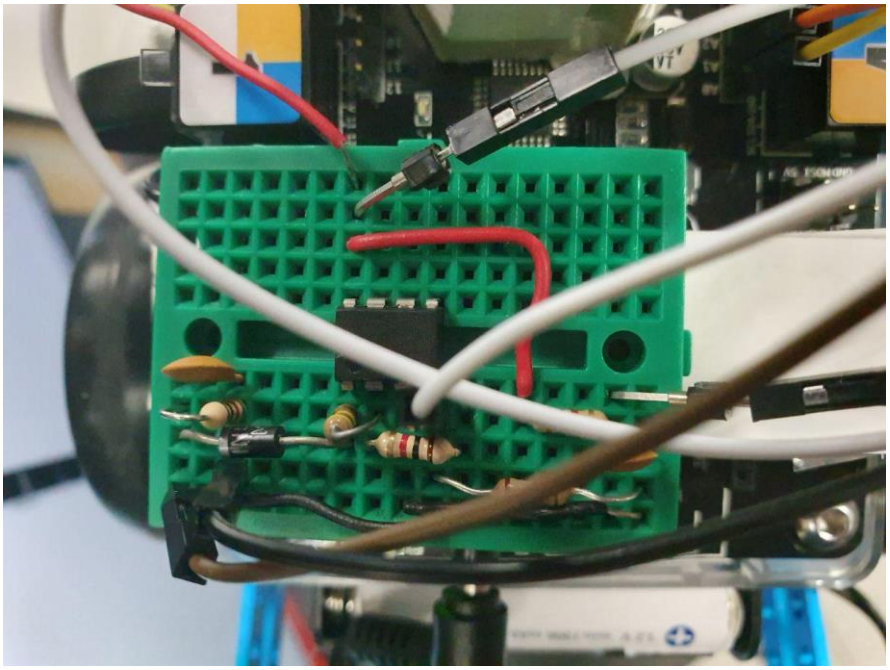

Figure 7: Microphone Circuit

As shown in Figure 8, we can differentiate the 100~300Hz from the 3kHz and above frequency by using high-pass filter with a low cut-off frequency. When the frequency detected is above 3kHz, the output will reach saturation. While, if frequency detected is between 100~300Hz, it will be in the stop band range, giving some output. This helps to easily differentiate between a no frequency and low frequency situation.

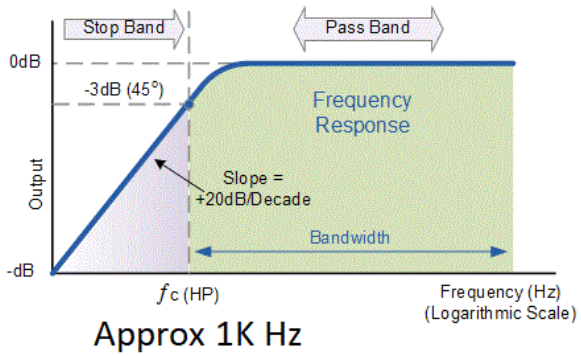

Figure 8: Gain vs Frequency Graph

2.3.2 Implementing our Arduino code

When the mBot senses black colour paper above it, the microphone (on the breadboard) will be activated. The function, frequencyCheck(), will detect for a sound (in this case a tune). The variable reading, gives a voltage output from the microphone, via analogRead(). A high frequency tune will be passed through the high-pass filter which will produce an output voltage of at least 2.8V. A low frequency tune would produce an output of between 0.5V and 2.8V. For accuracy, we did 300 loops to ensure that we get the correct readings as the tune played was not of a constant tone. Furthermore as the difference in readings were very minimal at times, we had to scale up the value by multiplying the reading by 1000. Referring to Figure 1, and as mentioned previously, the turns() function corresponds with the desired direction the mBot needs to turn.

2.4 End of Maze Detection Algorithm

As mentioned above, when the mBot senses black above, the microphone (on the breadboard) will be activated. In the case where the mBot reaches the end of the maze and no tune is played, output voltage would be close to 0V. We set a buffer and as such if our frequencyCheck() returns a count value of less than 30, it means that there was no audio and only ambient noise was recorded and hence it would signal the end of the maze. This was the cue to play our celebratory tone. The celebratory music played was the Nokia Ringtone.

3. Steps Taken for Calibration

3.1 Movement

Through various trial and error, we deduced the time required for the mBot to turn 90 degrees left or right. Since battery levels affect the motor speed, we also calibrated a backup set of values for a slower speed and duration required for the turn to complete.

3.2 Colour Sensing

We created 5 arrays to scan for and take in the corresponding RGB values for each of the RGB values (red, green and blue respectively) as detailed in Table 1.

Array Name	**Stores RGB values for:**
colourArray[]	Red, Green and Blue
whiteArray[]	White
blackArray[]	Black
greyDiff[]	white[] - black[]
testcolour[]	Sample colours

Table 1: List of Arrays and Corresponding Values

We scanned a white sample (white paper) to set the white balance. Our code goes through one colour at a time (of RGB), and sets the maximum reading for each colour to the whiteArray[]. Similarly for the black sample, the minimum reading was set as the black balance in the blackArray[]. The greyDiff[] array, represents the difference between the maximum and minimum range of possible values for test colours. Then we proceeded with data collection for the various coloured samples.

We scanned and stored the individual RGB values into colourStr[][]. For higher accuracy, we take the average of 100 samples. We then created if else loops to compare the RGB values stored in colourStr[][] to those in our colourArray[]. The detailed code taken for calibration can be found under the folder named Light_Sensor_Colour_mCore_Final.

4. Project Difficulties & How We Overcame Them

One of our biggest obstacles during the project was the sound challenge. In the beginning, we built two pass filters, a low and a high-pass filter. This was to detect low and high frequency tunes. The low frequency band-pass filter is shown in Figure 9. However, we realised we could have made do with one pass filter. Hence, we ended up only utilising a high-pass filter. Since the filters cannot fully suppress noise, if we use the correct gain and cut off frequency, we will get a higher voltage reading for high frequency and a lower voltage reading for a low frequency. When designed well, we could have distinctly differentiated a low, high and no frequency audio inputs. In addition, our mBot was less complex since we used fewer components in our circuit to perform the same action.

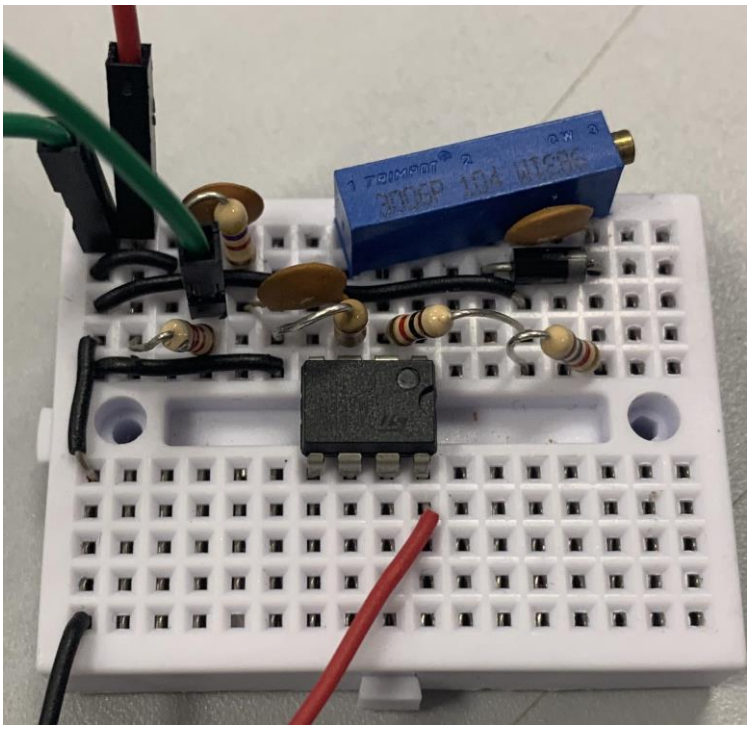

Figure 9: Backup Low-Frequency Band-Pass Filter

A second issue we faced was due to the battery output. The movement of our mBot depended largely on our battery output. A slight difference in battery output would result in inaccuracies in mBot turns. Hence, we would need to vary the speed. At the beginning, our mBot did not move smoothly, as even the slightest depletion in battery would result in a significant change in motor speed. Hence, we were required to constantly charge our batteries. A full battery lasted us about 2 mins, while a depleted battery lasted 10mins. Hence, while we used maximum speed during our test runs, we didn't want to risk it for the actual race and decided to use a slower speed at depleted battery levels as it can maintain the speed for a longer time. In other words, at full charge, the maximum speed would last for about 1-2 minutes. However,

a slightly depleted battery and at about half the speed, would allow us to use the speed values for about 20-30 minutes. Unfortunately, this resulted in a much longer time to complete the maze and we ended up taking twice as long as we could have.

In addition, shadows were another issue. They affected our colour calibration. Primary colours such as Red and Green was easily and accurately detected. However, colours such as Yellow, Purple and Blue (because of the closeness to purple) was not accurately determined. Part of the reason was due to ambient light interfering with the readings as well as the shadows casted by the board above. In order to overcome this issue, we fully covered our light sensor on the mBot with a card that we folded to prevent shadows from affecting the readings. Initially, the inside of the card was white and of a reflective material and hence during calibration, we realized it resulted in inaccurate readings. So we folded black colour paper (which was matte) and placed it inside as shown in Figure 10. It provided us with much more accurate readings and ensured the shadows had minimal impact on the readings.

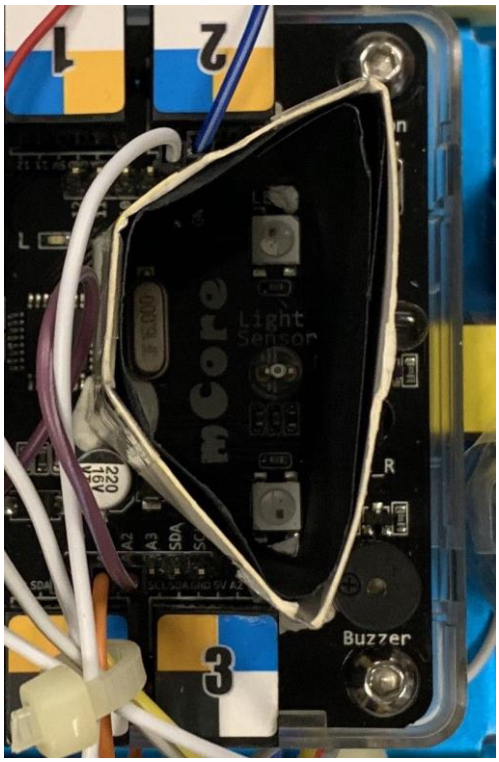

Figure 10: Card with Black Paper