DEPARTMENT OF PHYSICS, UNIVERSITY OF COLOMBO
PH3032-EMBEDDED SYSTEM LABORATORY

Bluetooth based RPM detector

Name: M.A.S. Ahamed

Index: s14784

Date: 28th Oct 2023

Abstract

In today's era of automation and precision, RPM measurement has become integral in various industries and applications. The "Bluetooth based RPM detector" project strives to address this need by combining hardware, IR sensor technology, and mobile app integration. It offers users a versatile, and user-centric solution for RPM measurement, enabling real-time data transmission, remote blade count input, and adaptability to different fan types. Testing confirms the system can further improve its accuracy, while the mobile app enhances user experience. Overall the project has been able to reach its primary objectives and is open for future advancements.

Table of figures

Table of Contents

1. Introduction

In the modern world the technology had played a major role in uplifting the standards of living to a whole new different level. The advancement in technology have improved the standards of households and all industries. With the advancement in technology, there arose the need for precise control and accuracy, since minute errors may lead to huge setbacks. In the era of automation and precise control, the need for accurate RPM detection systems is paramount across various industries and applications. The need for RPM detection comes to play in areas as diverse as industrial automation, automotive engineering, HVAC systems, and more. Through this project it is intended to propose and develop a low cost, robust and simple way of developing a home-made RPM detector which is user-friendly.

Fan systems are integral to numerous industrial, commercial, and residential applications, often serving as the heart of ventilation, cooling, and circulation. In order to ensure the optimal performance of these fans it becomes mandatory to precisely measure RPM and monitor them throughout. Even though there are existing RPM detectors, through this project it is intended to check the feasibility of designing a RPM detector with easily available sensors and microcontrollers.

1.1 Objectives

The primary objective of this project is to design and implement an RPM detection system which is robust, user-friendly and versatile through Bluetooth integration with a dedicated mobile app. The scope of the project can be broken down into the following aspects:

- Hardware and sensor setup for precise RPM measurement using an IR sensor.
- Development of an intuitive and user-friendly mobile app to interact with the RPM detector.
- Calibration mechanisms to adapt to different fan types.
- Comprehensive testing and evaluation of RPM detection accuracy and the mobile app's usability.

2. Theory

2.1 RPM (Revolutions Per Minute)

RPM, is a unit of measurement used to quantify the rotational speed of an object, typically a rotating or spinning element. It represents the number of full revolutions an object completes within one-minute time frame. RPM is an important parameter for a wide range of applications, including machinery, engines, fans, and any system where the rate of rotation is critical.

RPM is often measured using various sensors, such as tachometers, optical encoders, or IR sensors. It is a fundamental quantity for controlling, monitoring, and optimizing rotational systems, making it a vital concept in the fields of engineering, mechanics, and automation.

Figure 1 - Depicts a tachometer

2.2 Atmega32 Microcontroller

The low power CMOS 8-bit ATmega32 microcontroller is based on the AVR enhanced RICS architecture. It features 40 pins with various uses. The 32 general purpose working registers known as PORTS A, B, C, and D, each with 8 pins, are combined with a robust instruction set in the AVR core. Each of the 32 registers has a direct connection to the arithmetic logic unit (ALU). The ATmega32 also contains an extra 8 pins, each of which serves a different function. This microcontroller has been equipped with two ground pins.

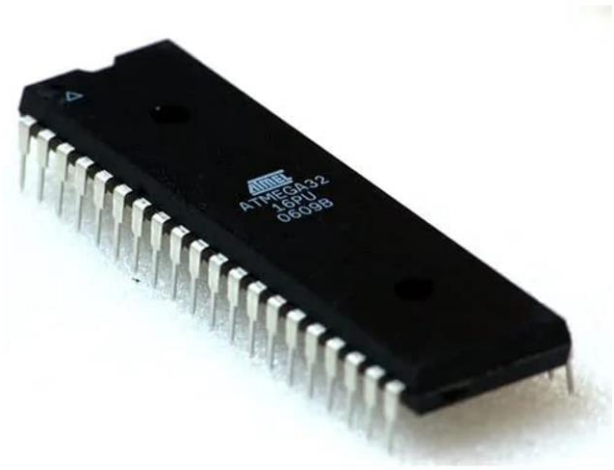

Figure 2 - Depicts and Atmega32 microcontroller

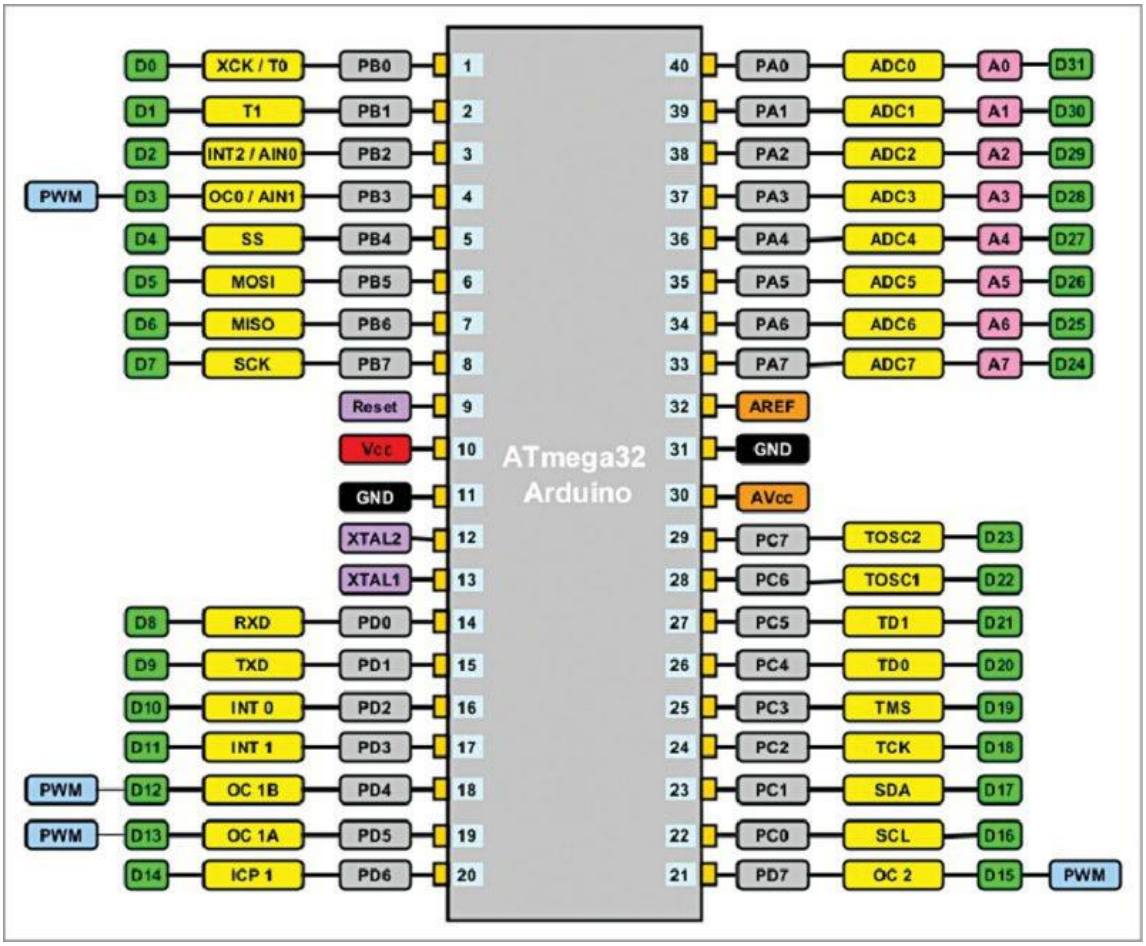

Figure 3 - Pinout of Atmega32 microcontroller

2.3 16x2 LCD display

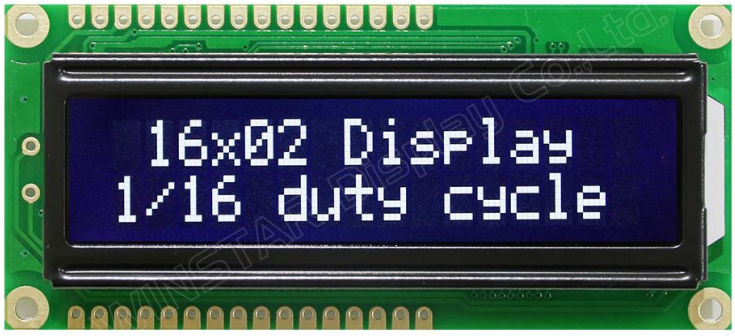

Figure 4 - Depicts a 16x2 LCD display module

The 16x2 LCD display's name refers to the display's 16 columns and 2 rows. It can display 32 characters, where each of those characters are made up of 5x8 pixel squares. Its operational voltage ranges from 4.7V to 5.3V, and it draws a current of 1mA without backlight. Both 8bit and 4bit modes may be used with this.

2.4 IR module

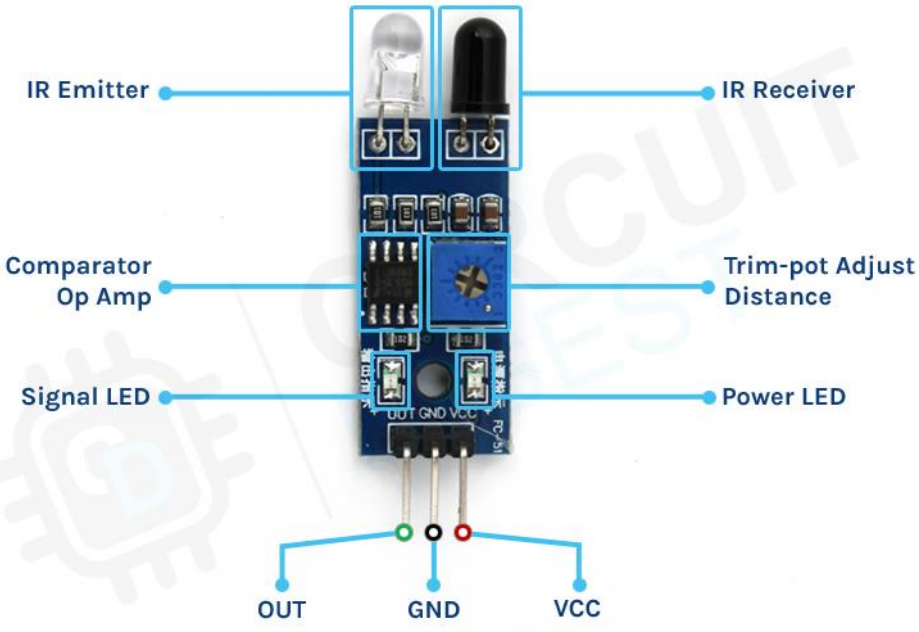

Figure 5 - Depicts an IR module and its components

Infrared Obstacle Avoidance IR Sensor Module (Active Low) consists of a pair of infrared transmitting and receiving tubes. When the transmitted light waves are reflected back, the reflected IR waves will be received by the receiver tube. The onboard comparator circuitry does the processing and the green indicator LED turns on.

The module features a 3-wire interface with VCC, GND, and an OUTPUT pin on its tail. It works fine with 3.3 to 5V levels. Upon hindrance/reflectance, the output pin gives out a digital signal (a low-level signal). The onboard preset helps to fine-tune the range of operation, the effective distance range is 2cm to 80cm.

2.5 HC-06 Bluetooth module

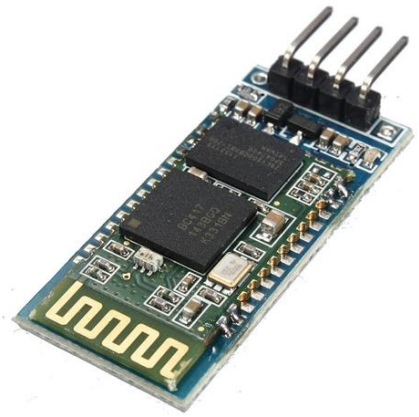

Figure 6 - Depicts a HC06 Bluetooth module

HC-06 Bluetooth module is a convenient way to establish Bluetooth communication using Serial Port Protocol. It consists of a built-in antenna and can cover a range of around 300ft. Its operating voltage is 3.3V and it has a default baud rate of 9600.

3. Methodology

- Initially spent some amount of time in going through on how an rpm detector works
- Then researched a bit on how to implement a simple tachometer.
- Then decided the means in which the device will use to detect the rpm and deliver it to the user.
- Then came up with the components and sensors that would be required to implement this project.
    - 16x2 LCD display
    - IR module
    - HC-06 Bluetooth module
    - Jumper wires
- After that a schematic diagram was designed on EasyEDA

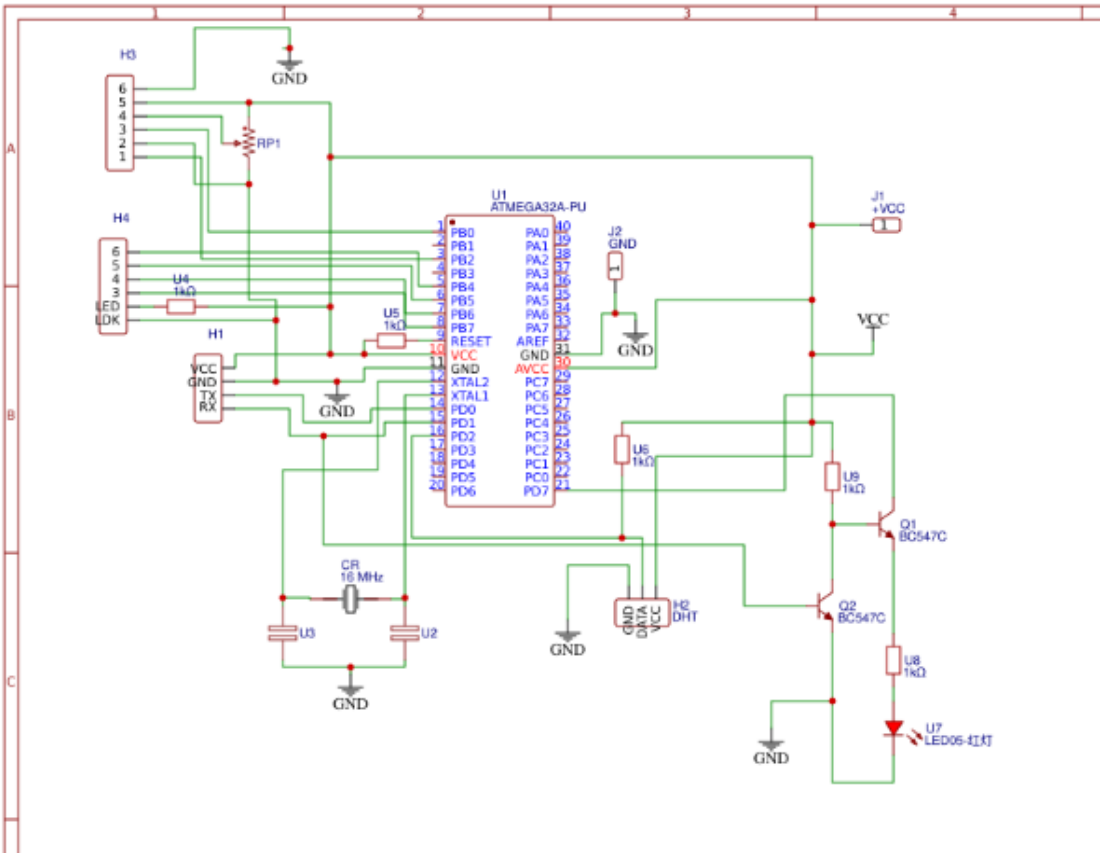

Figure 7 - Depicts the schematic drawn in EasyEDA

- Then the schematic diagram was brought to life by building the circuit using breadboard, jumper wires and other components.
- Up next the schematic was used to come up with a PCB design in EasyEDA software.
- Then the PCB design was printed onto an ink transferrable sheet and was transferred onto a copper board. Then FECl3 was used to remove the excess copper in order to create the circuit paths. Then headers, resistors and all other necessary components were soldered onto the PCB.
- Then the microcontroller, sensors and other peripherals were connected to the PCB

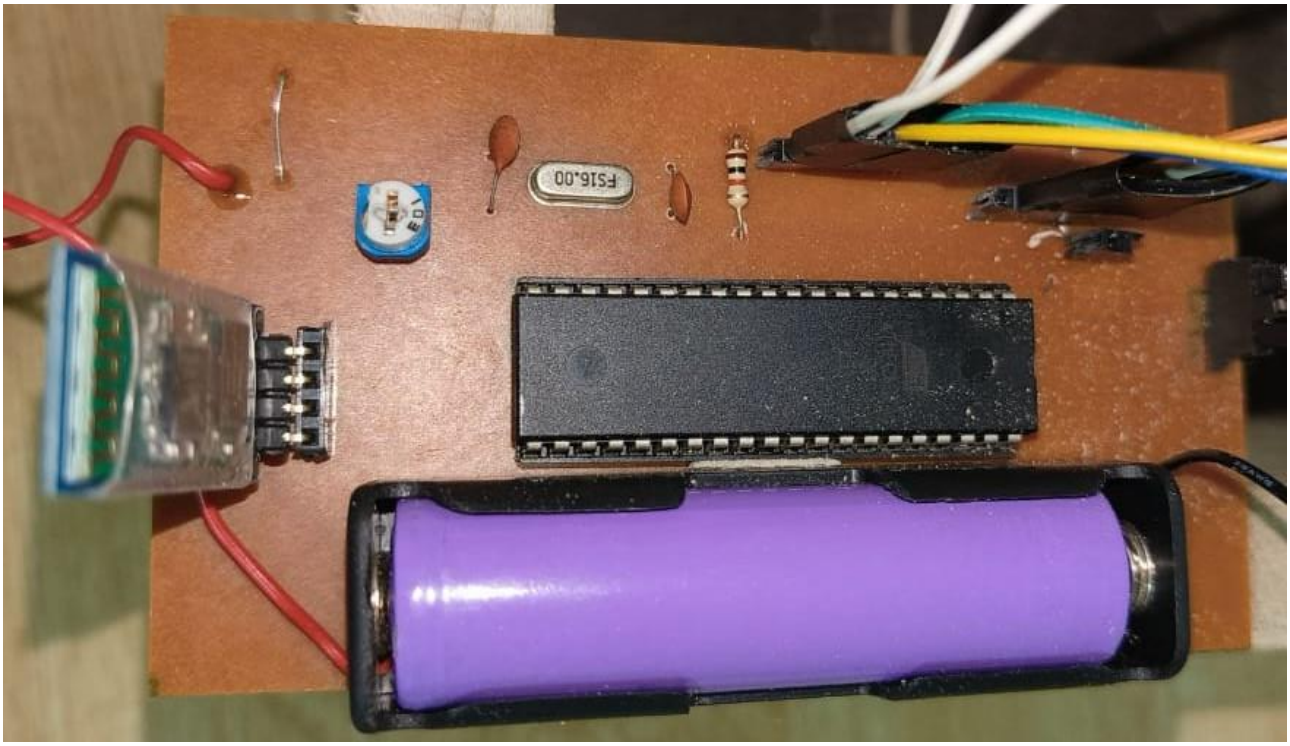

Figure 8 - upper view of PCB after completion

Figure 9 - Connection paths of the PCB.

- The device was powered by a 3.7V rechargeable battery to make the device stand-alone and easy to use.
- Then the app required to establish connection between user and device was built using the platform MIT app inventor.

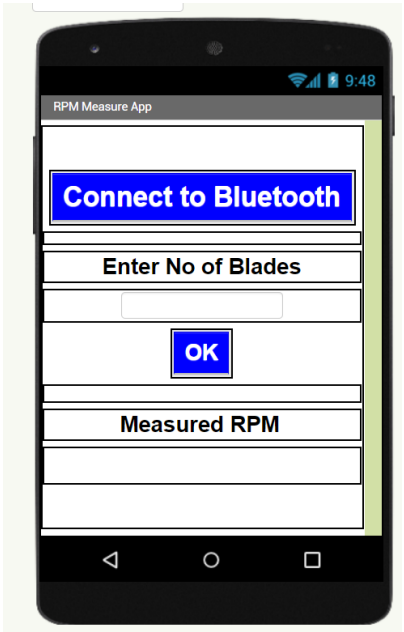

Figure 10 - User Interface of the app

- The relevant background functionalities were coded accordingly.

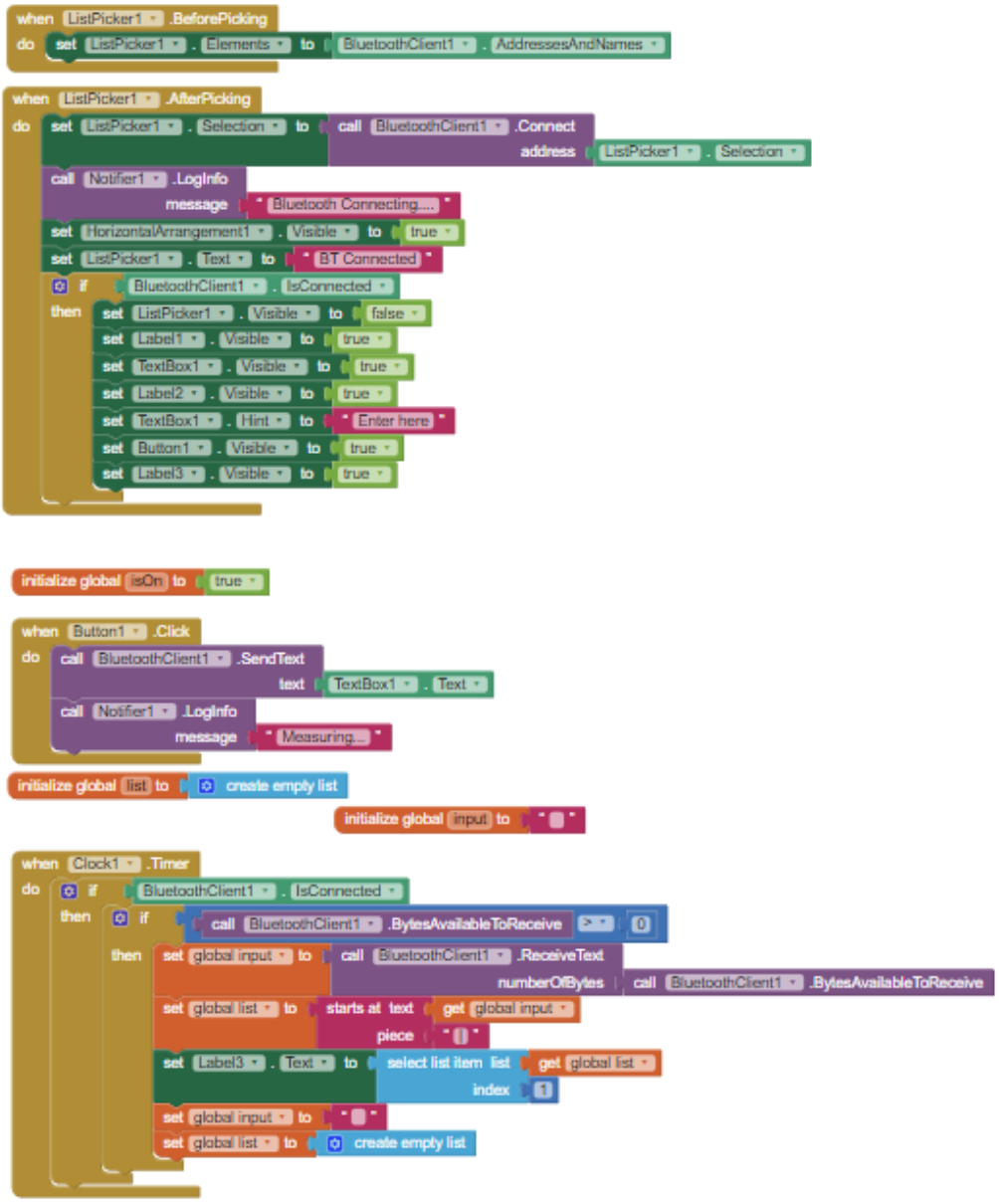

Figure 11 - Functionality block diagram of the mobile app

- Then the C code to be uploaded to the microcontroller was developed.
- Then at last all components were setup in a wooden box to give the complete product.

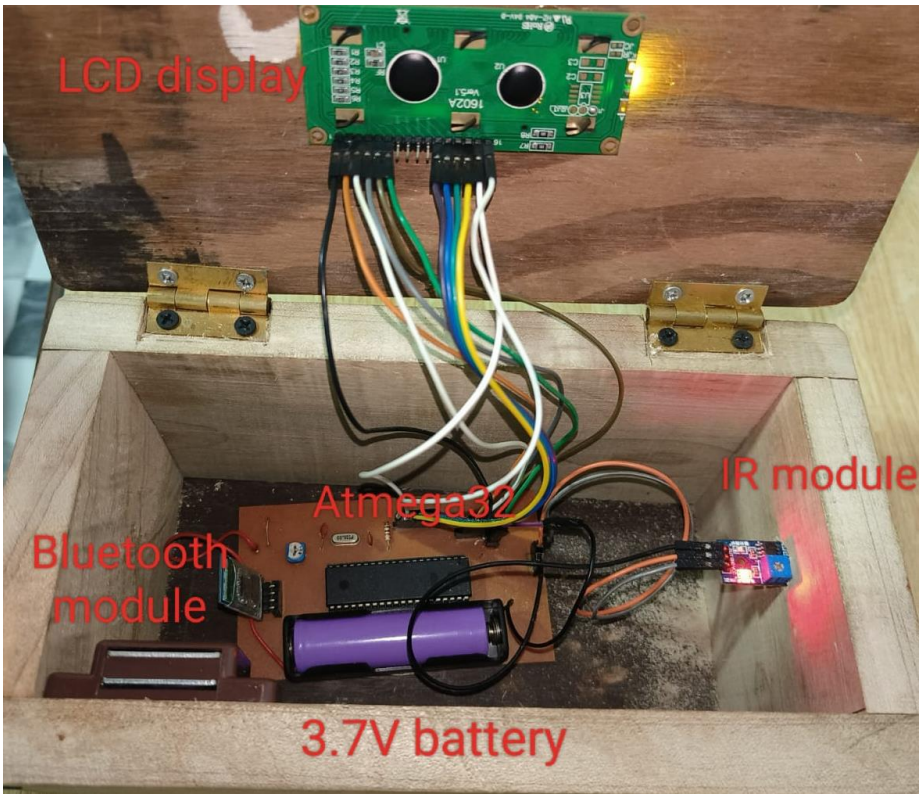

Figure 12 - Shows the completed product

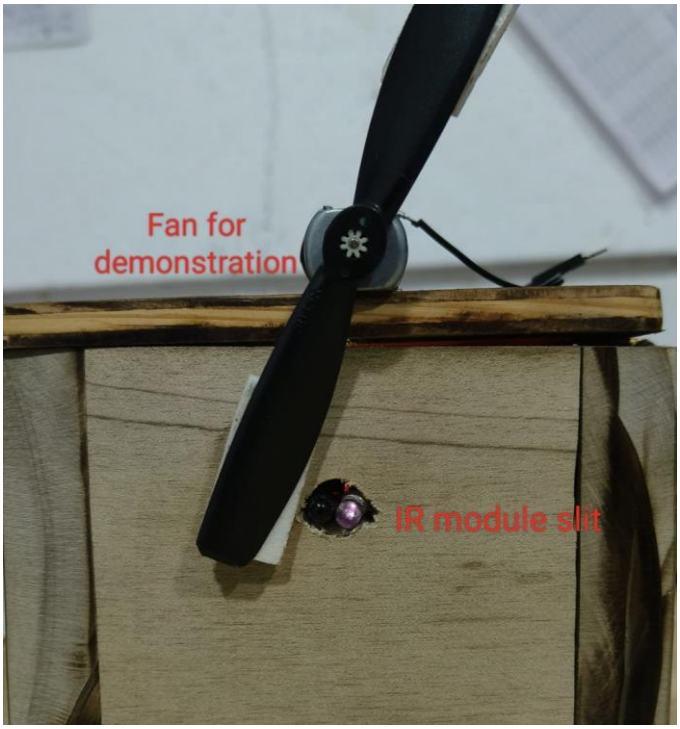

Figure 13 - Product demonstration

3.1 User instructions to handle the product

Getting started

- All you need to get started with the device is to install the Bluetooth control app
- There is no requirement to power the device since the device is powered with 3.7V rechargeable battery.
- All you have to do is to just press the power button of the device to start using it.

Operating instructions

- Initially press the power button.
- Then open the Bluetooth control app.
- Then enter the number of blades contained in the fan that is to be monitored.
- Then point the IR slit close to the blades of the fan.
- Examine the rpm value on the LCD display and the mobile app.

4. Results and discussion

RPM can be calculated in a number of methods, through this project two basic methods of detecting rpm were used. They are

1. Calculating rpm based on the number of times the fan blades are detected during a prescribed time interval
2. Calculating rpm based on the time taken to detect a prescribed number of fan blade detection.

Method 01

```
int delay1() {
    printf("reading rpm...\n");
    int i, j;
    unsigned int count = 0;
    int sensor = 1;
    for (i = 0; i < 100; i++) {
        for (j = 0; j < 100; j++) {
            if (digitalRead(sensor)) {
                count++;
                while (digitalRead(sensor));
            }
        }
    }
    return count;
}
int main() {
    int NUM_BLADES = 4;
    int time = 100;
    int RPM;

    int count = delay1();

    RPM = (60 * 100 * NUM_BLADES) / time;

}
```

Method 02

```
int getTime(){
  long t = millis();
  for(int i = 0; i<100; i++) {
    while(PINC7==0);
    while(PINC7==1);
  }
  return millis()-t;
}
rpm = 100*60/(b*getTime()/1000);
```

Eventhogh two different methods were adopted to calculate the value of rpm there was no huge deviation between the values obtained from the two methods.

When the accuracy of the rpm value calculated was put to test, it was initially found that the device was giving out the rpm value as maximum when there was no movement of blades at all and on the other hand it was giving out lesser and lesser values as the speed of rotation of blades were increased.

The speed of rotation was increased by increasing the voltage of the battery that was used to power the motor. Then in order to rectify this issue the rpm calculation method was edited by calibrating it based on known values of rpm.

Eventhough efforts were taken to increase the accuracy of the rpm reading, the accuracy level achieved was not great. It would have been great to use a laser sensor and compare the values obtained by the laser sensor and the Ir sensor. A problem which still a solution was not found is the fact that, how long should the fan blade be exposed to the sensor in orde to be detected by it. Getting a answer for this will have a huge positive impact on the accuracy of the readings.

As further modifications, integration of advanced sensors, such as optical encoders or Hall-effect sensors, could improve accuracy and expand the range of detectable objects. Additionally, implementing a self-calibration feature could simplify the setup process for users, ensuring precise RPM measurements across various fan types. Furthermore, the inclusion of wireless connectivity options, like Wi-Fi or IoT capabilities, would enable remote monitoring and data logging, making the product suitable for a wider array of applications

5. Conclusion

In conclusion, the "Bluetooth based RPm detector" project represents a simple way to develop a low cost tachometer. By combining hardware, sensor technology, and a user-friendly mobile app, this project focuses on users convenience and accuracy of its readings. Eventhough the readings aren't that accurate necessary steps could be taken to increase its accuracy to be used in real world scenarios. The real-time data transmission and remote blade count input feature via the mobile app make it highly accessible and user-focused. While the project has fairly achieved its primary objectives, there is a lot of room for further enhancements, including advanced sensor integration, self-calibration, and expanded connectivity options. Overall, the "Bluetooth based RPm detector" project is a promising start in developing an accurate and versatile tachometer.

6. References

how2electronics. (2023, August 15). Retrieved from how2electronics: https://how2electronics.com/digital-tachometer-ir-sensor-arduino/

Microchip. (2023, September 20). Retrieved from Microchip: https://ww1.microchip.com/downloads/en/DeviceDoc/doc2503.pdf

Pavleski, M. (2023, August 10th). *hackster.io*. Retrieved from hackster.io: https://www.hackster.io/mircemk/arduino-tachometer-rpm-meter-with-ir-sensor-module-39cbd9#:~:text=The%20sensor%20module%20consists%20of,detected%20by%20the%20Arduino%20controller.

7. Appendix

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>


// Constants
#define IR_SENSOR_PIN   PINC
#define IR_SENSOR_BIT   7
#define BLADES_PER_REV  4
#define LCD_RS (1 << PB0)
#define LCD_EN (1 << PB2)
#define LCD_DB4 (1 << PB4)
#define LCD_DB5 (1 << PB5)
#define LCD_DB6 (1 << PB6)
#define LCD_DB7 (1 << PB7)
#define LCD_DATA_PORT PORTB

#define EICRA   _SFR_MEM8(0x69)
#define EIMSK   _SFR_MEM8(0x3D)

// UART Constants for Bluetooth
#define BAUD 9600
#define MYUBRR (F_CPU / 16 / BAUD - 1)
#define b 2
```

```
// Global Variables
volatile uint32_t bladeCount = 0;
volatile uint32_t timeStart = 0;
volatile uint32_t timeEnd = 0;
volatile uint32_t rpm = 0;

// Function to initialize IR sensor and interrupts
void initializeIRSensor() {
    IR_SENSOR_PIN &= ~(1 << IR_SENSOR_BIT);
    EICRA |= (1 << ISC00) | (1 << ISC01);
    EIMSK |= (1 << INT0);
    sei();
}

void sendLCDCommand(uint8_t command) {
    PORTB &= ~LCD_RS; // Command mode
    LCD_DATA_PORT = (LCD_DATA_PORT & 0x0F) | (command & 0xF0);
    PORTB |= LCD_EN;
    _delay_us(1);
```

```
    PORTB &= ~LCD_EN;

    LCD_DATA_PORT = (LCD_DATA_PORT & 0x0F) | ((command << 4) & 0xF0);
    PORTB |= LCD_EN;
    _delay_us(1);
    PORTB &= ~LCD_EN;

    _delay_ms(2);  // Allow the LCD time to process the command
}

// Function to send a character to the LCD
void sendLCDCharacter(char data) {
    PORTB |= LCD_RS; // Data mode
    LCD_DATA_PORT = (LCD_DATA_PORT & 0x0F) | (data & 0xF0);
    PORTB |= LCD_EN;
    _delay_us(1);
    PORTB &= ~LCD_EN;

    LCD_DATA_PORT = (LCD_DATA_PORT & 0x0F) | ((data << 4) & 0xF0);
    PORTB |= LCD_EN;
    _delay_us(1);
    PORTB &= ~LCD_EN;
}

// Function to initialize the LCD
void initializeLCD() {
    DDRB |= LCD_RS | LCD_EN | LCD_DB4 | LCD_DB5 | LCD_DB6 | LCD_DB7;
    _delay_ms(15);  // LCD power-up delay

    // Initialize the LCD
    sendLCDCommand(0x02); // Initialize
    sendLCDCommand(0x28); // 4-bit, 2-line, 5x8 font
    sendLCDCommand(0x0C); // Display on, cursor off, blink off
    sendLCDCommand(0x06); // Increment cursor, no shift
    sendLCDCommand(0x01); // Clear display
    _delay_ms(2);  // LCD initialization delay
}



// Function to display RPM on the LCD
void displayRPMOnLCD(uint32_t rpm) {
    sendLCDCommand(0x01); // Clear display
    char buffer[16];
    sprintf(buffer, "RPM: %lu", rpm);
```

```
    for (int i = 0; buffer[i]; i++) {
        sendLCDCharacter(buffer[i]);
    }
}



// Function to initialize UART for Bluetooth communication
void initializeUART() {
    UBRRH = (unsigned char)(MYUBRR >> 8);
    UBRRL = (unsigned char)MYUBRR;

    // Enable receiver and transmitter
    UCSRB = (1 << RXEN) | (1 << TXEN);

    // Set frame format: 8 data bits, 1 stop bit
    UCSRC = (1 << URSEL) | (1 << UCSZ1) | (1 << UCSZ0);
}

// Function to send data over UART
void sendUARTData(uint32_t data) {
    // Wait for empty transmit buffer
    while (!(UCSRA & (1 << UDRE)));

    // Put data into the buffer, sends the data
    UDR = data;
}

int getTime(){
  long t = millis();
  for(int i = 0; i<100; i++) {
    while(PINC7==0);
    while(PINC7==1);
  }
  return millis()-t;
}

int main(void) {
    // Initialize IR sensor and interrupts
    initializeIRSensor();

    // Initialize LCD
    initializeLCD();

    // Initialize UART for Bluetooth
```

```
    initializeUART();

    while (1) {
        // Calculate RPM and update 'rpm' variable
        rpm = 100*60/(b*getTime()/1000);

        // Display RPM on the LCD
        displayRPMOnLCD(rpm);

        // Send RPM data via Bluetooth
        sendUARTData(rpm);

        // Other code as required for your project
    }

    return 0;
}
```