Nathan Roe
EN.606.715.81.SP23
2023-02-26

Project 3 - Propeller RPM Logging

The first project for this class involves using an Arduino to log propeller rotation speed data. A full demonstration video can be found on Youtube HERE and the source code for my project is included in the document, or available on Github HERE.

Requirements

- Measure propeller speed using an IR Emitter/Detector pair with an arduino
    - Derived Requirement - Measure propeller speed
        - Met by setting up one of the drone propellers to spin at varying speeds based on the drone remote controller
    - Derived Requirement - Use IR Emitter/Detector pair
        - Met by setting a high signal through an IR Emitter, and measuring whether the transmission is visible to the IR detector
- Use either a Round Robin with Interrupts or Function Queue Scheduling
    - Derived Requirement - Use Round-Robin with Interrupts
        - Met by waiting in the loop function until RPM count update is triggered by Digital Pin interrupt or logging is triggered by timer interrupt
- Capture RPMs over time, download to a host, and create a graph of the RPMs
    - Derived Requirement - Save file with values in CSV format containing time and RPM data
        - Met by writing output as two comma separated values and plotting results in the Results section

Hardware Design

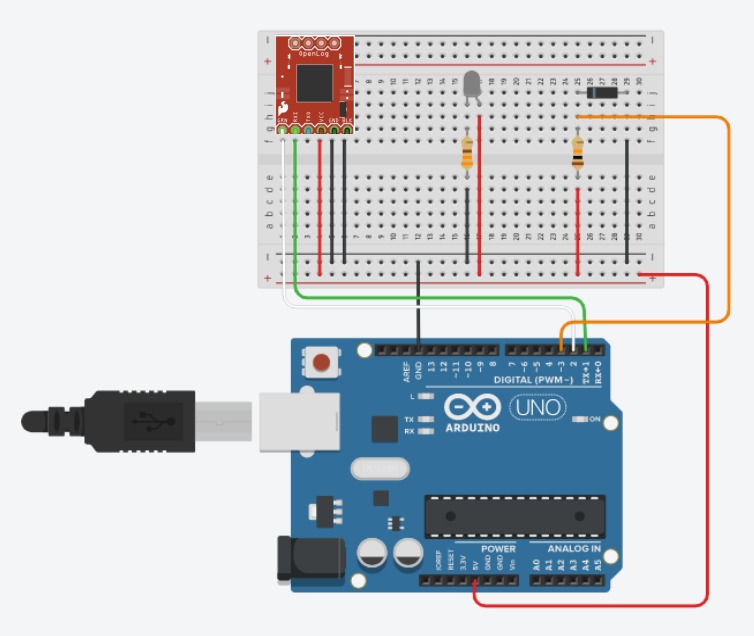
The hardware design for my project consists of three main component sets: an Infrared (IR) emitter, an IR receiver, and the SD card logger for logging data across the Serial bus.

The first component set consists of an IR emitter and a 330 Ω resistor connected in series. The high side of the IR emitter is connected to the Arduino 5v pin and the low side of the resistor is connected to the Arduino ground pin, causing the IR emitter to constantly emit infrared light. The emitter is placed below the drone propeller.

Nathan Roe
EN.606.715.81.SP23
2023-02-26

The second component set consists of an IR receiver and a 10 kΩ resistor. One side of the resistor is connected to 5v from the Arduino, and the other side is connected to the cathode of the IR receiver. The IR receiver anode is connected to ground and a wire runs from the resistor-diode junction to digital pin 3 on the Arduino. Reversing the diode means that Pin 3 reads high if the diode is blocking current (no IR detected), and drops towards 0 as the diode allows most current to pass. This Pin can then be used to trigger a digital interrupt as the pin changes from low to high whenever the IR is blocked by a propeller blade.

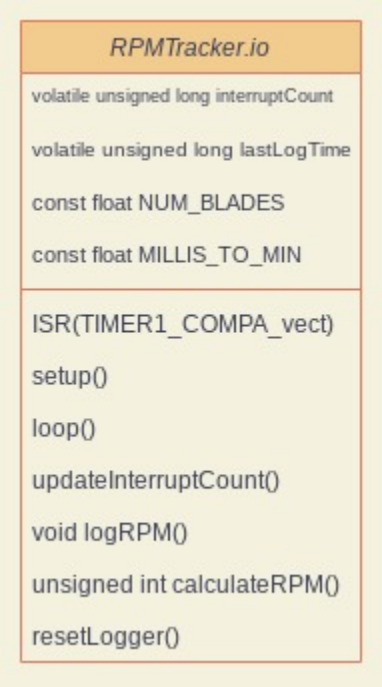
The last component set consists of an micro-SD card reader OpenLog component that records the Serial bus. The 5 V pin was connected to the logger's VCC pin, and Gnd was connected to the logger's GND and BLK pins. The Arduino Tx pin was then connected to the logger's Rx pin to allow the logger to receive and record Serial bus communications. I also connected the logger's GRN pin to a Digital pin 2 of the Arduino; having the pin set to high and then pulling it low allowed me to reset the logger, creating a new logging file, which was useful for testing.



Nathan Roe
EN.606.715.81.SP23
2023-02-26

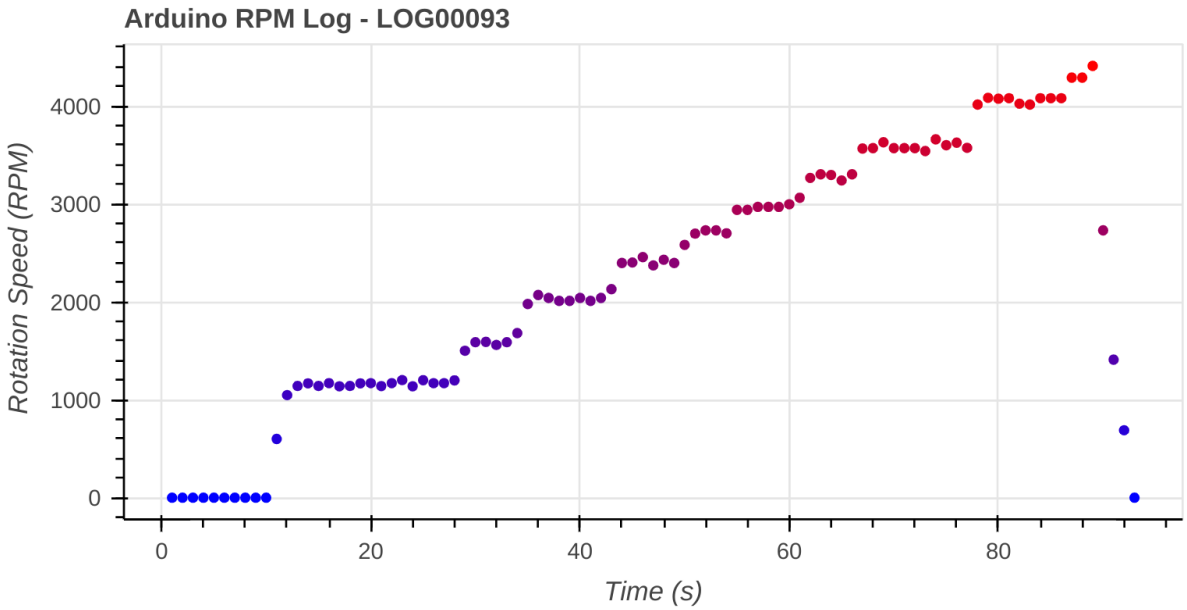
Software Design

The software design for my program consists of two main interrupt service routines (ISRs). The first is triggered whenever the digital pin connected to the IR receiver goes from low to high, signaling that something has blocked the IR signal. This ISR calls a function that updates the number of times that the signal has been blocked since the last update time. The second ISR is based on a timer interrupt, set to trigger every 1 second. Whenever this ISR is triggered, it calls the logging function which calculates the latest average RPM value and writes it to the Serial bus for logging.



Nathan Roe
EN.606.715.81.SP23
2023-02-26

Results



The plot above shows the results of logging from my project demo. The logger shows corresponding jumps in RPM for each time that I bumped the throttle on the RC Controller for the drone, increasing the rotation speed.

```ino
// I/O Data
// const int rxPin = 0; // Currently unused
const int txPin = 1;
const int logResetPin = 2;
const int readPin = 3;

// RPM Data
volatile unsigned long interruptCount = 0;
volatile unsigned long lastLogTime;
const float NUM_BLADES = 2;
const float MILLIS_TO_MIN = 60000.;

// Interrupt Data
ISR(TIMER1_COMPA_vect)
{
    logRPM();
}

// Setup Serial bus transmissions and interrupts for
// timer interrupt and digital pin interrupt
void setup()
{
    // Default state high, low to reset logging
    pinMode(logResetPin, OUTPUT);
    digitalWrite(logResetPin, HIGH);

    Serial.begin(9600);
    pinMode(readPin, INPUT);
    Serial.print("Time (s), Rotation Speed (RPM)\n");

    // Setup timer interrupt for logging
    TCCR1A = 0;
    TCCR1B = 0;
    // Set Timer 1 compare value for 1s (0x3D09 at 1024prescalar)
    OCR1A = 0x3D09;
    // Set Timer 1 to prescaler 1024
    TCCR1B = (1 << CS12) | (1 << CS10);
    // Set for comparison to OCR1A
    TCCR1B = TCCR1B | (1 << WGM12);
    // Set Timer 1 to use compare match
    TIMSK1 = (1 << OCIE1A);

    // Setup digital pin interrupt
    lastLogTime = millis();
    attachInterrupt(digitalPinToInterrupt(readPin),
                    updateInterruptCount,
                    RISING);
}

// Loop doing nothing unless interrupts called
void loop()
{
    // Empty, wait for interrupts
}
```

```
// Update propeller pass count on digital pin interrupt
void updateInterruptCount()
{
    ++interruptCount;
}

// Log current RPM rate to serial bus
void logRPM()
{
    Serial.print(millis() / 1000.);
    Serial.print(", ");
    Serial.print(calculateRPM());
    Serial.print("\n");
    interruptCount = 0;
    lastLogTime = millis();
}

// Calculate RPM based on propellor pass count and delta time
unsigned int calculateRPM()
{
    return ((interruptCount / NUM_BLADES) / (float)(millis() - lastLogTime)) *
MILLIS_TO_MIN;
}

// Reset Logger to create new log file
void resetLogger()
{
    // Pull logger interrupt low
    digitalWrite(logResetPin, LOW);
    delay(10);
    digitalWrite(logResetPin, HIGH);
}
```

```
import glob
import os

import numpy as np
import pandas as pd

from bokeh.plotting import ColumnDataSource, figure, show

# Find all log files -  end in .TXT extension
file_list = glob.glob('*.TXT')

# Create plot for each file
for file in file_list:
    file_name = os.path.splitext(os.path.basename(file))[0]

    x_axis_label = 'Time (s)'
    y_axis_label = ' Rotation Speed (RPM)'

    # Read in CSV data
    vals = pd.read_csv(file)  # , names=[x_axis_label, y_axis_label])\

    max_val = max(vals[y_axis_label])
    min_val = min(vals[y_axis_label])

    temp_copy = vals[y_axis_label].copy(deep=True)
    temp_copy[temp_copy > max_val] = max_val
    temp_copy[temp_copy < min_val] = min_val

    # Set up color array - 0F->#0000FF, 100F->#FF0000
    reds = np.array(['0x{:02x}'.format(int(255*temp/max_val))
                     for temp in temp_copy])
    reds[vals[y_axis_label] >= max_val] = "0xff"
    reds[vals[y_axis_label] <= min_val] = "0x00"

    blues = np.array(['0x{:02x}'.format(int(255 - (255*temp/max_val)))
                      for temp in temp_copy])
    blues[vals[x_axis_label] <= min_val] = "0xff"
    blues[vals[y_axis_label] >= max_val] = "0x00"

    colors = [
        f'#{reds[idx].split("x")[1]}00{blues[idx].split("x")[1]}' for idx in
range(len(reds))]

    # Begin Bokeh plotting
    source = ColumnDataSource({'Time': vals[x_axis_label],
                               'RPM': vals[y_axis_label],
                               'colors': colors})

    TOOLTIPS = [(x_axis_label, '@Time'),
                (y_axis_label, '@RPM')]

    tools = 'hover,wheel_zoom,pan,box_zoom,reset,undo,save'

    fig = figure(title=f'Arduino RPM Log - {file_name}',
                 toolbar_location="right",
                 aspect_ratio=2,
```

```
                tools=tools,
                tooltips=TOOLTIPS)

    fig.xaxis.axis_label = x_axis_label
    fig.yaxis.axis_label = y_axis_label

    fig.scatter('Time',
                'RPM',
                source=source,
                fill_color='colors',
                line_color='colors')
    show(fig)
```