Software Development for Real Time Embedded Systems- 605.715, Fall 2025

Matthew Halligan

Project 3

September 28, 2025

Professor Doug Ferguson

Project 3: Arduino Uno + Electronic Speed Control Measurement and Serial Transmission

Project Requirements:

- Measure the speed of a brushless propeller which is driven by an Arduino and measured using an IR Emitter/Detector pair measured by the Arduino using a Round Robbin with Interrupts design.
- Capture the RPMs over time, download them to a Host and create a graph of the RPMs.

Design:

Hardware list:

- Using an Arduino Uno,
- IR LED
- IR Photodiode
- 2x 10K Ohm resistors
- a bread board,
- 1x QWinOut brushless motor + propeller + Secure mounting rig
- 1x esc module
- 9 jumper wires

Configure the circuit according to Figure 1.

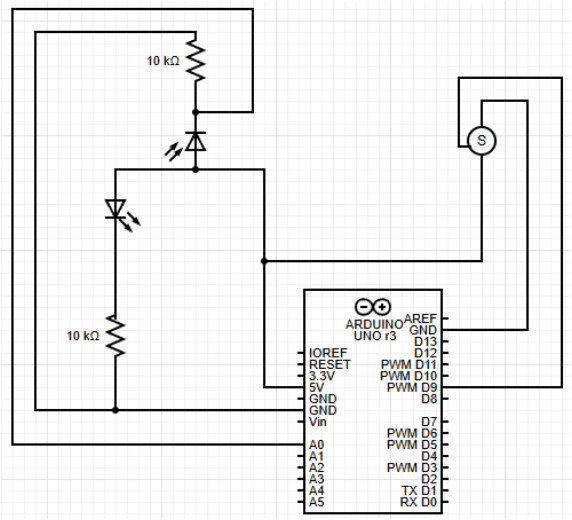

Figure 1 Brushless Servo Motor IR-based Measurement Circuit diagram.

Software Design:

The software design is influenced by the assignment instructions.

This firmware measures IR beam breaks caused by a propeller and converts those breaks to an approximate RPM using a timed collection window and interrupt-driven detection. This firmware continuously powers an IR LED and places the IR phototransistor on A0. The ADC is configured in free-running mode so conversions run continuously and trigger an ADC completion ISR. The ISR performs the only time-critical work. The ISR applies a “hysteresis,” a method for creating stability in IR detections and maintaining the state machine despite minor variations, and a short consecutive-sample requirement to the ADC results, and configures flags when a falling such as when a beam break occurs or rising when a beam restore event is detected i.e., the propeller moves out of the way.

A hardware timer, timer 2, is configured in CTC mode to generate a 1ms tick. The Timer2 ISR accumulates milliseconds and signals the main loop when a configurable collection window elapses. The data and demo utilize a 15 second collection window.

At startup the firmware enables the IR LED and samples A0 briefly to compute a baseline. From that baseline it computes threshLow and threshHigh using a configured MIN_DIFF_ADC and HYSTERESIS_ADC. The ADC ISR uses these thresholds and requires CONSECUTIVE_REQUIRED consecutive samples below/above the thresholds before it flags an edge.

CSV records are emitted automatically over the serial port at the end of each collection window. Figure 2 shows control flow of the program from setup of timers, timing registers, ISR vector selection, and main loop workflow.

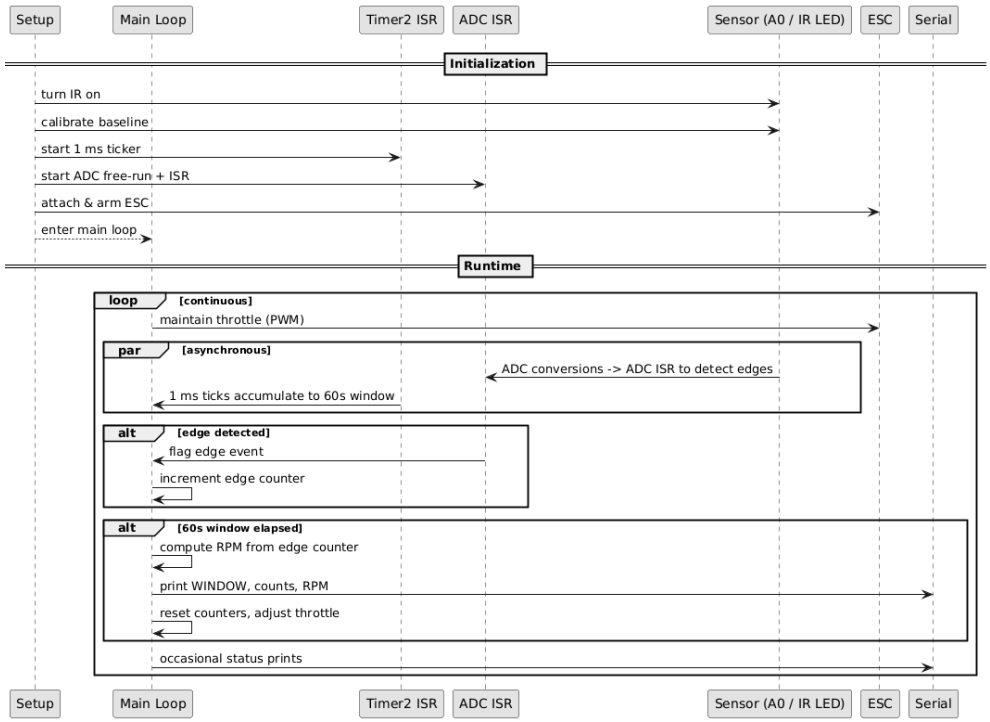

Figure 2 Timer Setup, Hardware- and Timer-based ISR Configuration, and Arduino Uno Main loop sequence diagram

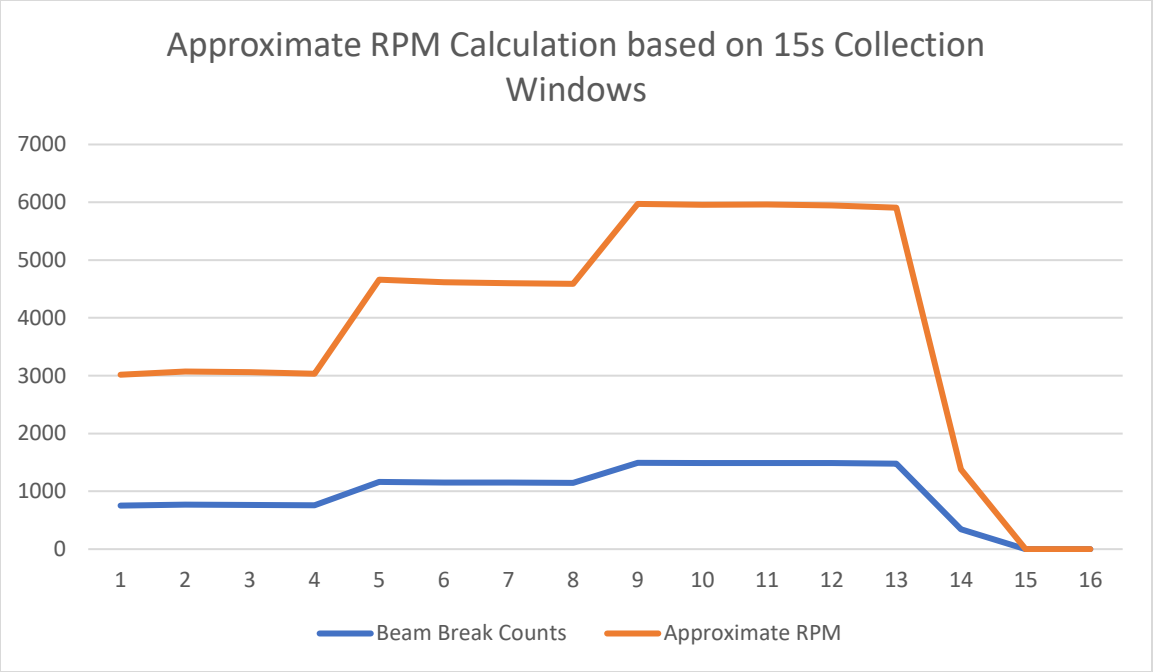

Figure 3 Incremental Approximate RPM calculation with varying PWM signals

Figure 3 demonstrates a 4 minute data sampling with minimum throttle measurement two 1/3 throttle measurement capturing the quantity of beam breaks during the collection window and the calculated RPMs. After 4 sampling windows the PWM signal is increased, and the measured RPM is measured accordingly showing the expected, plateaued increase. At the 13th collection window, power is removed to the esc and the RPM measurement is shown to decrease to 0 after the 14th collection window is complete.

Video Demo Links:

Halligan - Project 3 - Measure Brushless Motor Speed using Interrupt Driven Design Demo: https://livejohnshopkins-my.sharepoint.com/:v:/g/personal/mhallig2_jh_edu/EST6Pfb9cYJDkBU7Izuh2QIBJNZj_mTQPMDdWTRpjdgN1Q?e=3UAKjY

Project Code

```
//////////////////////////////////////////////////
//
//  Matthew Halligan
//  Software Development for Real Time Embedded Systems, Fall 2025
//  Module 4/5 Project 3
//  Professor Doug Ferguson
//
// Read and export IR Beam Breaks and convert to RPM for Electronic Speed Control Tuning
// Implements fast analog IR detection, break and recovery of signal caused by propeller interference
// using hardware based interrupts and automated RPM calculation using timer based interrupts in a
// "Round Robin with interrupts Design" and serial export
//
// To use, ensure serial monitor is attached via Arduino IDE 2.3.6 set to 115200 baud.
// Ensure circuitry is configured according to figure one in associated report.
// Provide power to esc module through external 11V power source.
// No further action is required by the user once power is provided to ESC and power and serial
// are supplied to Arduino Uno in correct sequence.
//
//////////////////////////////////////////////////
// Fast beam-break detection using ADC free-running + ISR
// For UNO- A0 sensor
// 60-second RPM collection window
#include <Servo.h>
Servo esc;
const int escPin = 9;
const int irLedPin = 7;
const int sensorChannel = 0; // A0
const int statusLed = LED_BUILTIN;
int pw = 1100;

// detection tuning
const int MIN_DIFF_ADC = 60;     // expected drop in IR when broken
const int HYSTERESIS_ADC = 12;   // hysteresis band
const uint8_t CONSECUTIVE_REQUIRED = 3; // require N consecutive samples beyond threshold to accept edge

// runtime state (shared with ISR)
volatile bool beamPresent = true;      // current state (true = beam present)
volatile bool fallingEdgeFlag = false; // set by ADC ISR when beam broken detected
```

```
volatile bool risingEdgeFlag = false;  // set by ADC ISR when beam restored detected
volatile unsigned long edgeCount = 0;  // total number of detected breaks during current
window

// internal ISR counters (kept small)
volatile uint8_t consecBelow = 0;
volatile uint8_t consecAbove = 0;

// Timer2 millisecond counter/state (for 60s window)
volatile unsigned long msCounter = 0UL;     // ms accumulator (0..14999)
volatile bool sampleReady = false;          // set true when 15s elapsed; main loop consumes it
const unsigned long MS_PER_WINDOW = 15000UL; // 15 seconds = 15000 ms

unsigned long processedWindows = 0;

// thresholds (computed at startup)
int baselineWithIR = 0;
int threshLow = 0;
int threshHigh = 0;

// main bookkeeping
unsigned long lastEdgeMicros = 0;
const unsigned int countsPerRevolution = 1; // set to pulses per one shaft revolution

// --- Configure Timer2 for ~1ms CTC ticks (OCR2A=249, prescaler=64) ---
void setupTimer2_1ms() {
  cli();              // disable global interrupts while configuring
  TCCR2A = 0;
  TCCR2B = 0;
  TCNT2 = 0;
  // CTC mode: WGM21 = 1
  TCCR2A |= (1 << WGM21);
  // OCR2A for 1ms with prescaler 64: ticks_per_ms = 16000000/64/1000 = 250 -> OCR2A = 249
  OCR2A = 249;
  // Set prescaler to 64: CS22 = 1 (CS21=0, CS20=0)
  TCCR2B |= (1 << CS22);
  TCCR2B &= ~((1 << CS21) | (1 << CS20));
  // Enable Timer2 compare A interrupt
  TIMSK2 |= (1 << OCIE2A);
  sei();              // enable interrupts
}

// Timer2 Compare Match A ISR at approx. once per 1 ms
ISR(TIMER2_COMPA_vect) {
```

```
  msCounter++;
  if (msCounter >= MS_PER_WINDOW) {
    msCounter = 0;
    sampleReady = true; // signal main loop that the 60s window finished
  }
}

// ADC free-running setup
void setupAnalogInterrupt(){
  // Configure ADC for free-running conversions on A0 with interrupt on completion
  ADMUX = (1 << REFS0) | (sensorChannel & 0x07); // AVcc reference, channel A0

  // ADCSRA: ADEN | ADSC | ADATE | ADIE | ADPS2..0 (prescaler 128)
  ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADATE) | (1 << ADIE) |
           (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

  ADCSRB = 0; // free running
  // Global interrupts enabled by caller (we'll ensure in setup)
}

// quick average using analogRead() for calibration only
int quickAnalogAvg(int samples) {
  long s = 0;
  for (int i = 0; i < samples; ++i) {
    s += analogRead(A0);
    delay(1);
  }
  return (int)(s / samples);
}

void computeThresholds() {
  threshLow = baselineWithIR - MIN_DIFF_ADC;
  if (threshLow < 0) threshLow = 0;
  threshHigh = threshLow + HYSTERESIS_ADC;
  if (threshHigh > 1023) threshHigh = 1023;
}

// ADC ISR: runs at free-running ADC rate (fast). Keep short.
ISR(ADC_vect) {
  uint16_t v = ADC; // 10-bit result

  // Beam gives HIGH reading normally, and breaks drop the ADC reading.
  if (v < threshLow) {
    consecBelow++;
```

```
    consecAbove = 0;
    if (consecBelow >= CONSECUTIVE_REQUIRED && beamPresent) {
      beamPresent = false;
      fallingEdgeFlag = true;
      consecBelow = 0;
    }
  } else if (v > threshHigh) {
    consecAbove++;
    consecBelow = 0;
    if (consecAbove >= CONSECUTIVE_REQUIRED && !beamPresent) {
      beamPresent = true;
      risingEdgeFlag = true;
      consecAbove = 0;
    }
  } else {
    // within hysteresis band -> reset counters
    consecBelow = 0;
    consecAbove = 0;
  }
}

void calcAndPrintRPM() {
  // compute RPM using the 60-second window
  // RPM = (edgeCount pulses / countsPerRevolution) * (60 / windowSeconds)
  // Since windowSeconds = 60, RPM = edgeCount / countsPerRevolution
  unsigned long counts = 0;
  counts = edgeCount;
  edgeCount = 0; // reset for next window
  unsigned long rpm = counts*60/15;
  Serial.print(processedWindows + 1);
  Serial.print(',');
  Serial.print(counts);
  Serial.print(',');
  Serial.print(rpm);
  Serial.println();

  processedWindows++;
  if (processedWindows % 4 == 0){
    pw = pw + 100;
  }
}

void setup() {
  pinMode(irLedPin, OUTPUT);
```

```
  pinMode(statusLed, OUTPUT);
  digitalWrite(irLedPin, HIGH);
  Serial.begin(115200);
  while (!Serial) { ; }
  Serial.println("Fast ADC free-run beam-break detection, 15s collection window size");

  // Turn IR ON and measure baseline (simple calibration)
  digitalWrite(irLedPin, HIGH);
  delay(200);
  baselineWithIR = quickAnalogAvg(40);
  computeThresholds();
  Serial.print("Baseline: ");
  Serial.print(baselineWithIR);
  Serial.print("  threshLow=");
  Serial.print(threshLow);
  Serial.print("  threshHigh=");
  Serial.println(threshHigh);

  // Setup ADC free-running + ISR
  setupAnalogInterrupt();

  // Attach and arm ESC
  esc.attach(escPin);
  Serial.println("Arming ESC...");
  esc.writeMicroseconds(1000); // min throttle
  delay(2000);
  // optional calibration pulse — comment out if not desired
  // esc.writeMicroseconds(2000); delay(2000);
  esc.writeMicroseconds(1000);
  delay(2000);
  Serial.println("ESC armed. Ready.");
  Serial.print("Collection Window ID, Beam Break Counts, Approximate RPM");
  Serial.println();
  // Configure Timer2 for ms ticks
  setupTimer2_1ms();
}

void loop() {
  // Keep motor from overspinning during testing
  if (pw <= 1300) {
    esc.writeMicroseconds(pw);
  }

  // Process edge events flagged by ADC ISR
```

```
if (fallingEdgeFlag) {
  fallingEdgeFlag = false;
  unsigned long now = micros();
  unsigned long dt = (lastEdgeMicros == 0) ? 0 : (now - lastEdgeMicros);
  lastEdgeMicros = now;

  edgeCount++;
  digitalWrite(statusLed, HIGH);

  // Uncomment for debugging of IR beam break detection
  // Serial.print("[BROKEN] t=");
  // Serial.print(now);
  // Serial.print(" us  dt=");
  // Serial.print(dt);
  // Serial.print(" us  total=");
  // Serial.println(edgeCount);
}

if (risingEdgeFlag) {
  risingEdgeFlag = false;
  unsigned long now = micros();
  digitalWrite(statusLed, LOW);
  // Uncomment for beam detection restoration debugging
  // Serial.print("[RESTORED] t=");
  // Serial.print(now);
  // Serial.println(" us");
}

// When 60s window completes, sampleReady is true: compute and print RPM
if (sampleReady) {
  sampleReady = false;
  calcAndPrintRPM();
}

// occasional status print for IR detection baseline and thresholds
static unsigned long lastPrint = 0;
// if (millis() - lastPrint > 30000) {
//   Serial.print("Baseline=");
//   Serial.print(baselineWithIR);
//   Serial.print("  threshLow=");
//   Serial.print(threshLow);
//   Serial.print("  threshHigh=");
//   Serial.println(threshHigh);
//   lastPrint = millis();
```

```
  // }

  delay(2);
}
```