Prerequisites

This worksheet builds upon the Lab 4 worksheet. You will have needed to complete the Lab 4 worksheet before you can continue with this lab. Out of your Arduino pack, you will need: 1 Arduino, 1 A3144 hall effect sensor, 1 Servo, 1 LED, 1 $10K\Omega$ resister, 1 220Ω resister, and 8 jumper cables.

Objectives

The Objectives of today's lab is to build a full control system; namely, using a hall effect sensor to determine if an Electronic Speed Controller (ESC) — and associated motor — should be powered up or down. While the system we build today is designed for an ESC and motor, we will be using a servo as a proxy. As mentioned in the Lab 3 worksheet, both an ESC and a servo operate the same way within the context of an Arduino; however, an ESC requires additional configuration outside the scope of an Arduino.

1 Creating the circuit

There are three circuits needed for this worksheet, the first is for the hall effect sensor (HES), the second is for the visual display LED, and the third is for the ESC.
HES circuit: Arduino +5V→HES left leg, Arduino GND→HES middle leg, Arduino Pin 2→HES right leg. HES left→$10K\Omega$ resister→HES right legs.
LED circuit: Arduino Pin 4→220Ω resister→LED→Arduino GND.
ESC circuit: Arduino +5V→Servo power cable, Arduino GND→Servo ground cable, Arduino Pin 9→Servo sensor cable.

2 System Functionality

The basis of this system is the same as is described in Lab 4: *creating a system with a hall effect sensor, so that whenever a magnet passes the field of the hall effect sensor, we turn on a LED.* The worksheet in Lab 4 walks you through how to do this through an *interrupt* to record the time in milliseconds of when the hall effect value changed. In the 'loop' function, if the time difference between the current time, and the last recorded hall effect interrupt time is less than 500 milliseconds, we turn on the light; otherwise, we turn it off.

This worksheet simply expands that logic to include an ESC — say connected to a motor on the back wheel of a bike. If the last recorded time of the hall effect interrupt is less than 1500 milliseconds, we slowly power up the ESC until it reaches maximum output. If at any point during

the power up of the ESC, the last recorded hall effect interrupt drops outside that 1500-millisecond window, we power down the ESC.

3 Changing the existing code

Firstly, open your sketch from the Lab 4 worksheet into Arduino IDE. Once this is open, save the sketch under a new name ('File'→'Save As'). This will make sure you still have a copy of the sketch from Lab 4. The code changes required can be summed up into three sections: 1 - defining the ESC and required constants (global scope), 2 - setup the pins ('setup' scope), 3 - determine to power up or down the ESC ('loop' scope). While this is all the code required to meet the specification of the system, we will also be adding in some additional debug information, so you can see how the system is powering up the ESC.

3.1 Defining the ESC

The first thing we need is to include the ESC definitions and library to correctly control the ESC — a servo is our case. We need to include the standard servo library, define a new 'ESC_PIN' on pin 9, define a 1.5-second maximum delay for the hall effect sensor, and define a new 1-second interval on how often to display a debug message.

```
#include <Servo.h>
#define ESC_PIN 9
#define CRANK_PASS_MAXIMUM_DELAY 1500
#define MESSAGE_MAXIMUM_INTERVAL 1000
```

The next section of logic to include is two constant values that dictate the rate of increase and decrease of power to the ESC. The first constant sets that when we are powering up the ESC, we will increment the ESC output value by 0.5 every iteration of the 'loop' function. The second constant sets that when we are powering down the ESC, we will decrement the ESC output value by 3.0 every iteration of the 'loop' function. This gives us the facility to accelerate slowly but decelerate quickly.

```
const float esc_step_up = 0.5;
const float esc_step_down = 3.0;
```

The last section of logic within the global scope is to create a store a variable of when we last displayed the diagnostic message, the current power output to the ESC and a reference to the servo library to control the ESC.

```
unsigned long message_previous_time;
float esc_power_output;
Servo esc;
```

3.2 Pins

The code within the 'setup' function is the standard code we have seen before, firstly you initialise any variables that don't have an initial value '*message_previous_time*' in our case, then we need to setup the ESC pin and library as we did in Lab 3.

```
message_previous_time = 0;
pinMode(ESC_PIN, OUTPUT);
esc.attach(ESC_PIN);
esc.write(0);
```

3.3 Powering the ESC

The code within the 'loop' function is new code that you haven't seen before, but it builds upon everything covered in previous labs. The first piece of code to add to your sketch is the code that determines if you should be powering up or powering down the ESC. We do this through a nested 'IF ELSE' statement.

The outer 'IF' statement checks if the current time of the 'loop' minus the last time the hall effect interrupt was recorded is less than 1500 milliseconds. If this is true, we increment the ESC output value by the 'step up' value as long as the power output value wouldn't exceed the maximum of 180 — if it does, we set it to the maximum value. If the outer 'IF' statement was false — the last interrupt wasn't recorded within the last 1500 milliseconds of the current 'loop' time, we start to power down the ESC. We decrement the ESC power output value by the 'step down' value as long as the output value wouldn't go below the minimum 0 value — if it does, we set it to the minimum value.

The last line of code is to finally write the calculated output value to the ESC.

```
if (current_loop_time - crank_interrupt_current_time < CRANK_PASS_MAXIMUM_DELAY) {
  if (esc_power_output + esc_step_up <= 180) {
    esc_power_output += esc_step_up;
  } else {
    esc_power_output = 180;
  }
} else {
  if (esc_power_output - esc_step_down >= 0) {
    esc_power_output -= esc_step_down;
  } else {
    esc_power_output = 0;
```

```
  }
}
esc.write((int)esc_power_output);
```

The last section of code is to help us understand what is going on with the system; however, before we can add this into our sketch we need to remove 1 line of code: '*Serial.println("CrankActive");*'. Once the above code has been removed, we can all the code in below which will print a diagnostic message every second to tell us what the current output power of the ESC is.

```
if (current_loop_time - message_previous_time >= MESSAGE_MAXIMUM_INTERVAL){
  message_previous_time = current_loop_time;
  Serial.print("ESC power output: ");
  Serial.println(esc_power_output);
}
```

Try it out on your Arduino

You now should be able to compile your sketch ('Sketch'→'Verify/Compile') and then upload to your Arduino ('Sketch'→'Upload'). You should now be able to pass the magnet across a side of the hall effect sensor, and every time you do, it should light up the LED and your servo should start to power up. When you stop passing the magnet past your hall effect sensor, the servo will start to power down.

Once you have a working sketch, try to expand your sketch to calculate the speed (Kmph) that the magnet is passing your hall effect sensor and display the speed in your debug message. *Hint: You will need to record the last two interrupt times.*

Help

If you are having issue compiling the project or don't understand where each section goes, you can see a complete version of the sketch with additional source code comments at `https://github.com/dsbrennan/mec217-arduino-labs/blob/main/lab-5-control-system/lab-5-control-system.ino`.