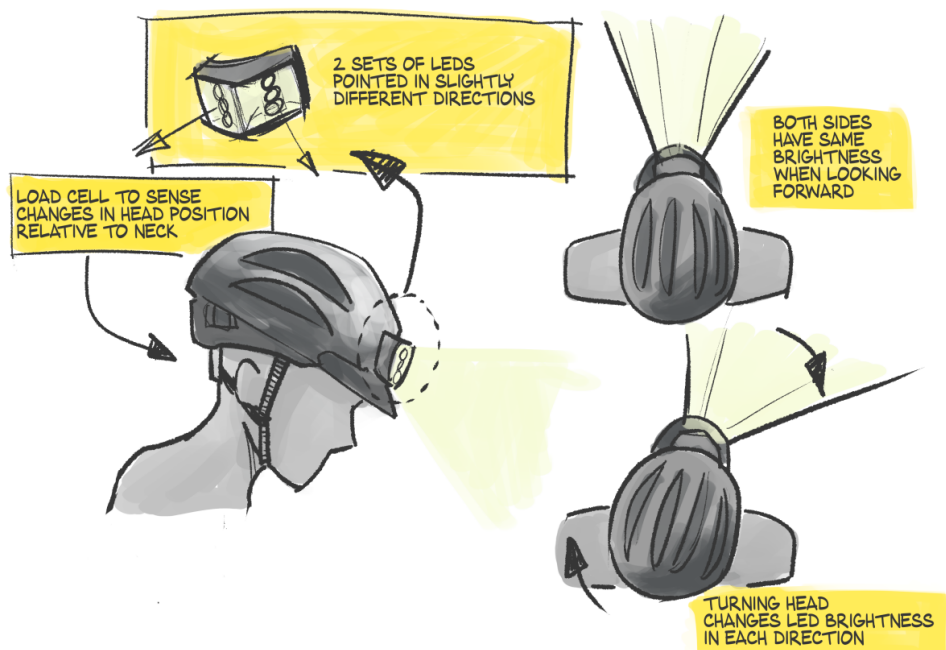# Arduino Shield HeadLight

Edward Manson (75953196)
Alex Stiles (18120890)

August 27, 2021

Contents

List of Figures

List of Tables

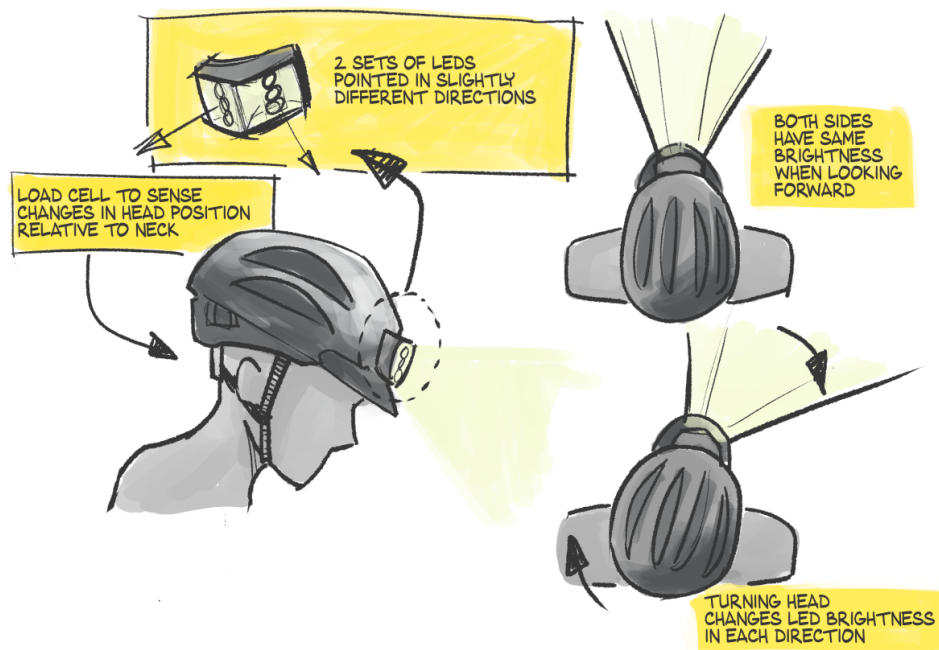# Listings

1 Introduction

2 Background

Figure 1: A sketch of the HeadLight

3 Requirements & Specifications

3.1 Requirements

3.2 Specifications

4 Design Overview & Rationale

4.1 Arduino Shield

A load cell is used to measure the force of the head rotation, which has an internal Wheatstone bridge. The output from the Wheatstone bridge goes into an instrumentation amplifier. The instrumentation amplifier consists of non-inverting amplifiers on the output of the Wheatstone bridge, which then goes into a differential amplifier, as shown in figure **??**. The output of the differential amplifier goes into an LC low pass filter to remove high-frequency noise.
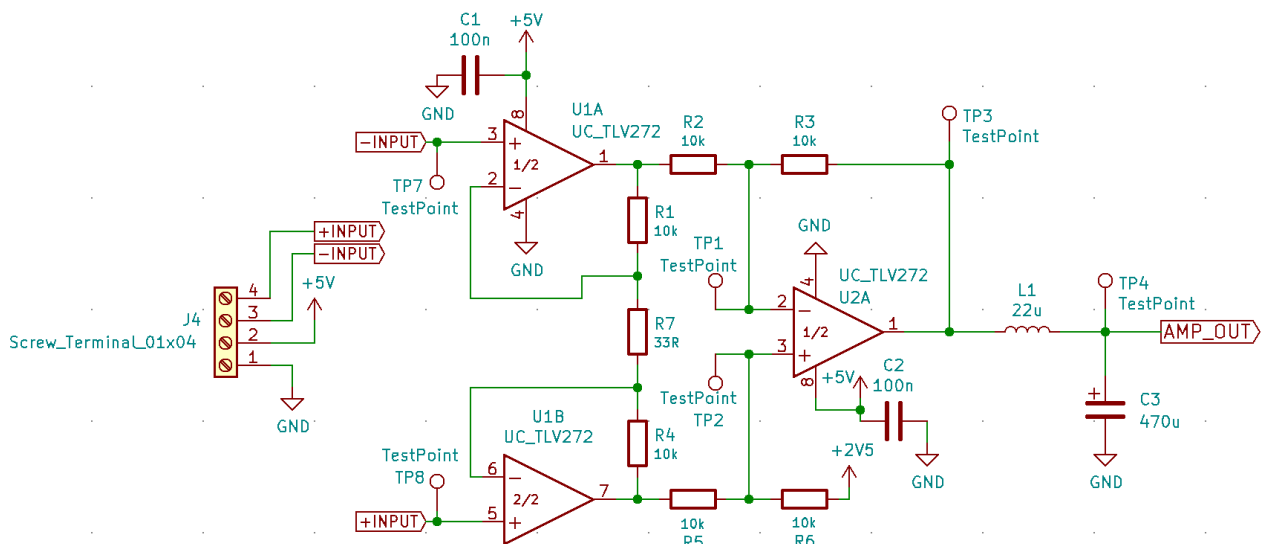
Figure 2: The layout of the instrumentation amplifier.

The output of the differential amplifier can end up being a negative value, as the Arduino can not read negative voltage, and there is no negative supply for the differential amplifier; the differential amplifier has to be biased. A voltage reference was constructed out of a voltage divider of equal values and a buffer to bias the instrumentation amplifier (shown in figure **??**).
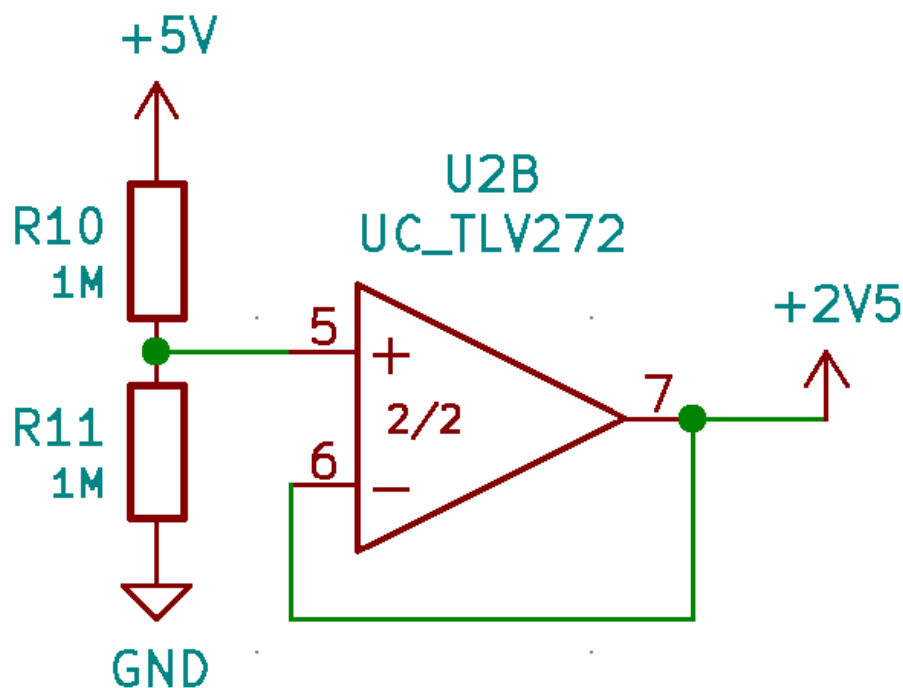

Figure 3: The layout of the voltage reference.

For each LED array, the PWM output from the Arduino goes into the gate of a MOSFET. The current limiting resistor is connected to the anode of the LEDs, and the cathode of the LEDs is connected to the drain of the MOSFET.
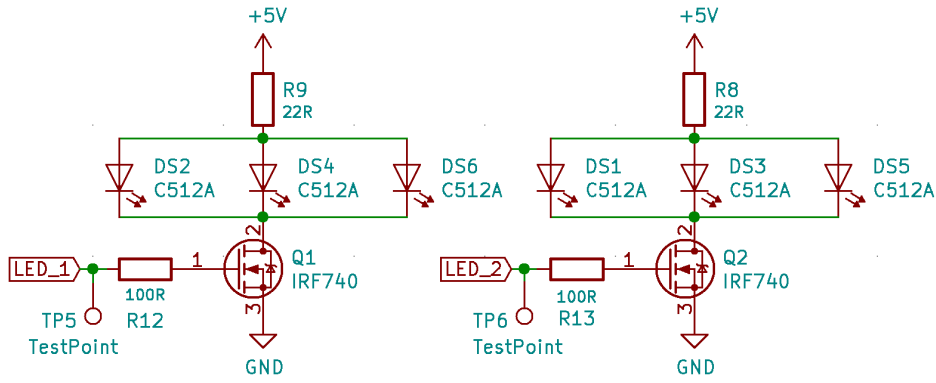
Figure 4: The layout of the LED arrays.

4.1.1 PCB Layout

The PCB was laid out to ensure that the ground plane would have no cuts, maximising EMC performance. Decoupling capacitors were placed right next to the power pins of the ICs. Additional care was taken so that traces were not too close to any antipads, as having traces too close to antipads can worsen EMC **emc**. Test points were added so that we can troubleshoot any problems when assembling the prototype. 1x1 header were used as ground test points so that the ground clip of the oscilloscope probe can be attached.

4.1.2 Amplifier

An instrumentation amplifier was chosen to amplify the signal from the load cell. The instrumentation amplifier consists of two non-inverting amplifiers, of which the outputs go into a differential amplifier. As the Arduino cannot read negative voltages, nor is there a negative supply, the differential amplifier was biased with 2.5 V. This results in the positive difference being between 2.5 V and 5 V, and the negative difference is between 0 V and 2.5 V. A one op-amp differential amplifier has an extremely low input impedance; thus the Wheatstone bridge must have a matched impedance. A two op-amp instrumentation amplifier would require one less package but would be less stable than the three op-amp instrumentation amplifier. The three op-amp instrumentation amplifier also has a spare op-amp which is then utilised for the voltage reference.

An initial gain of 24 Ω was calculated (as shown in equation (**??**)) **ti:sloa034**, but after some test on a breadboard, the gain was changed 33 Ω. This was partly due to the load cell to measure ∼50 % more than rated **htc:tal221**.

$$V_O = \left[(\text{Sig}^+) - (\text{Sig}^-) \right] \left[\frac{R_4}{R_2} \left[\frac{2R_f}{R_g} + 1 \right] \right] \tag{1}$$

$$R_g = \frac{2R_f}{\left[\frac{V_O}{(\text{Sig}^+) - (\text{Sig}^-)} \frac{R_2}{R_4} \right] - 1} \tag{2}$$

$$= \frac{2 \cdot 10\,\text{k}\Omega}{\left[\frac{2.5\,\text{V}}{3\,\text{mV}} \frac{10\,\text{k}\Omega}{10\,\text{k}\Omega} \right] - 1} \tag{3}$$

$$= 24.03\,\Omega \tag{4}$$

4.1.3 Voltage reference

A voltage divider was used to produce 2.5 V to bias the differential amplifier. It was decided to use a buffer on the output of the voltage divider so that the common-mode rejection error is minimised. This is due to the op-amp's low output impedance, which does not introduce any noticeable common-mode rejection error. A 2.5 V Zener diode would have been preferred. The resistors in the voltage divider have tolerances and therefore may not give exactly 2.5 V; however, 2.5 V Zener diodes were not available from the store. 1 MΩ resistors were chosen to limit the current and thus power consumption.

4.1.4 Filter

A filter was used to filter out high-frequency noise from the load cell. The filter was chosen to be an LC filter. An RC filter could have been used, but it would have caused the signal's voltage to drop slightly due to the voltage drop across the resistor. The LC filter does require more space on the PCB, but the PCB had enough room for the

capacitor and the inductor. LC filters also have better attenuation as the attenuation slope is twice as steep as an RC filter. The filter was placed on the output of the differential amplifier. If filters were placed on the inputs of the differential op-amp, more components would be needed, and there would not be enough space on the board. A 22 µH inductor was chosen due to its availability in the electronics store. A 470 µF capacitor was selected as a compromise of its physical size, cutoff frequency, and availability. The cutoff frequency was found to be 1.595 kHz, as shown below:

$$f = \frac{1}{2\pi\sqrt{LC}} \tag{5}$$

$$= \frac{1}{2\pi\sqrt{22\,\mu H 470\,\mu F}} \tag{6}$$

$$= 1.595\,\text{kHz} \tag{7}$$

4.1.5 LED arrays

Each LED array has a MOSFET; this was done to have enough current supplied to the LEDs. The pins that can output PWM are only able to supply 40 mA, so the outputs from these pins were connected to the gates of the MOSFETs. This allows the LEDs to take full current and are thus brighter. IRF740 MOSFETs were chosen due to their availability in the store and their low resistance. They have a max current of 10 A, so more LEDs can be added if required. The low resistance of the IRF740 ($0.55\,\Omega$), reduces the power losses when the MOSFET is on. Individual resistors could have been matched to each LED; however, this would increase the number of components and take up extra space on the PCB. $100\,\Omega$ resistors were placed between the PWM output from the Arduino and the gate of the MOSFETs, this was done to minimise ringing and to limit current.

5 Testing

The PCB will be assembled and attached to an Arduino. The code will be uploaded to the Arduino. The load cell will be manipulated, and the PWM signal will be probed using an oscilloscope. The duty cycle of the PWM signal will be compared to the expected duty cycle. The LEDs will then be connected, and the load cell manipulated to check that the LEDs behave as expected in terms of brightness and direction changes.

6 Conclusions

Appendices

A   Applications of the Design Toolkit

Table 1: Applications of the Design Toolkit.

Tool Type	Tool Name	Where/How Used
Strategy	Divide & Conquer	Divide and conquer was used for the hardware design, the

B   Code

```
1  #include <stdint.h>
2  #include <stdbool.h>
3
4  #include "button.h"
5
6  Button_t init_button(uint8_t pin_number, void (*callbackFunction)()) {
7      Button_t button = {
8          .pin_number = pin_number,
9          .debounce_delay = DEBOUNCE_DELAY,
10         .last_trigger_time = 0,
11         .state = false,
12         .last_state = false,
13         .callbackFunction = callbackFunction
14     };
15
16     return button;
17 }
18
19 void button_clicked(Button_t* button) {
20     button->callbackFunction();
21 }
22
23 void debounce_state(Button_t* button, bool read_button_state, long unsigned current_time)
       {
24     if (read_button_state != button->last_state) {
25         button->last_trigger_time = current_time;
26     }
27
28     if ((current_time - button->last_trigger_time) > button->debounce_delay) {
29         if (read_button_state != button->state) {
30             button->state = read_button_state;
31             if (button->state) {
32                 button_clicked(button);
33             }
34         }
35     }
36
37     button->last_state = read_button_state;
38 }
```

Listing 1: Button code.

```
1  /**
2   * @file button.h
3   * @author Alex Stiles
4   * @brief Provides the types and function prototypes required for the button.
5   * @version 0.1 alpha
6   * @date 2021-08-15
7   *
8   * @copyright Copyright (c) 2021
9   *
10  */
11
12 #ifndef BUTTON_H
13 #define BUTTON_H
14
15 #if defined(__cplusplus)
16 extern "C" {
17 #endif
18
19 #include <stdint.h>
20 #include <stdbool.h>
21
22 #define DEBOUNCE_DELAY 20 // Debounce delay in ms
```

```
23
24  typedef struct Button_s Button_t;
25
26  struct Button_s {
27      /* General */
28      uint8_t pin_number;
29
30      /* Debounce */
31      long unsigned debounce_delay;
32      long unsigned last_trigger_time;
33
34      bool state;
35      bool last_state;
36
37      /* Callback */
38      void (*callbackFunction)();
39  };
40
41  Button_t init_button(uint8_t pin_number, void (*callbackFunction)());
42  void buttonClicked(Button_t* button);
43  void debounce_state(Button_t* button, bool read_button_state, long unsigned current_time)
        ;
44
45  #if defined(__cplusplus)
46  }
47  #endif
48
49  #endif // BUTTON_H
```

Listing 2: Button header code.

```
1   /**
2    * @file LED_Group.c
3    * @author your name (you@domain.com)
4    * @brief Contains the source code for a LED group.
5    * @version 0.1
6    * @date 2021-08-13
7    *
8    * @copyright Copyright (c) 2021
9    *
10   * This source file provides the functionality to get and set the group's brightness (
         from 0 to 255) as required by the Arduino's PWM output.
11   * It also provides getters for the group pin number and direction.
12   *
13   */
14
15  #include <stdint.h>
16
17  #include "LED_Group.h"
18
19  /**
20   * @brief Get the group brightness
21   *
22   * @param led_group
23   * @return int8_t
24   */
25  uint8_t get_group_brightness(const LED_Group_t* const led_group) {
26      return led_group->group_brightness;
27  }
28
29  /**
30   * @brief Set the group brightness
31   *
32   * @param led_group
33   * @param group_brightness
34   */
35  void set_group_brightness(LED_Group_t* const led_group, uint8_t group_brightness) {
```

```
36          led_group->group_brightness = group_brightness;
37 }
38
39 /**
40  * @brief Get the group pin number
41  *
42  * @param led_group
43  * @return int8_t
44  */
45 uint8_t get_group_pin_number(const LED_Group_t* const led_group) {
46     return led_group->group_pin_number;
47 }
48
49 /**
50  * @brief Get the group direction
51  *
52  * @param led_group
53  * @return Direction_t
54  */
55 Direction_t get_group_direction(const LED_Group_t* const led_group) {
56     return led_group->direction;
57 }
```

Listing 3: LED group code.

```
 1 /**
 2  * @file LED_Group.h
 3  * @author Alex Stiles
 4  * @brief Provides the types and function prototypes required for the LED groups.
 5  * @version 0.1 alpha
 6  * @date 2021-08-13
 7  *
 8  * @copyright Copyright (c) 2021
 9  *
10  */
11
12 #ifndef LED_GROUP_H
13 #define LED_GROUP_H
14
15 #if defined(__cplusplus)
16 extern "C" {
17 #endif
18
19 #include <stdint.h>
20
21 #include "voltage.h"
22
23 typedef enum {
24     LEFT = 0,
25     RIGHT
26 } Direction_t;
27
28 typedef struct {
29     uint8_t group_pin_number;
30     Direction_t direction;
31     uint8_t group_brightness;
32 } LED_Group_t;
33
34 uint8_t get_group_brightness(const LED_Group_t* const led_group);
35 void set_group_brightness(LED_Group_t* const led_group, uint8_t group_brightness);
36
37 uint8_t get_group_pin_number(const LED_Group_t* const led_group);
38
39 Direction_t get_group_direction(const LED_Group_t* const led_group);
40
41 #if defined(__cplusplus)
42 }
```

```
43  #endif
44
45  #endif // LED_GROUP_H
```

Listing 4: LED group header code.

```
1   /**
2    * @file LED_Group_Collection.c
3    * @author Alex Stiles
4    * @brief Contains the source code for manipulating all the induvidual LED groups (see
            LED_Group.c)
5    * @version 0.1 alpha
6    * @date 2021-08-13
7    *
8    * @copyright Copyright (c) 2021
9    *
10   * This source file provides the functionality to add and get induvidual LED groups.
11   * It also provides the ability to update the required PWM output to each of the
            induvidual gruops, given a load voltage.
12   *
13   */
14
15  #include <stdint.h>
16  #include <stdio.h>
17  #include <stdlib.h>
18  #include <math.h>
19
20  #include "LED_Group_Collection.h"
21
22  /**
23   * @brief Create a LED group collection, and add an LED group to each direction (LEFT and
            RIGHT) with the pins set to left_pin_number and right_pin_number respectively.
24   *
25   * @param left_pin_number
26   * @param right_pin_number
27   * @return LED_Group_Collection_t
28   */
29  LED_Group_Collection_t init_led_group_collection(uint8_t left_pin_number, uint8_t
        right_pin_number) {
30      LED_Group_Collection_t led_group_collection = {.led_groups = {0}, .mode = FOLLOW, .
            mode_changed = false};
31
32      LED_Group_t led_group_one = {
33          .group_pin_number = left_pin_number,
34          .direction = LEFT,
35          .group_brightness = 0
36      };
37
38      LED_Group_t led_group_two = {
39          .group_pin_number = right_pin_number,
40          .direction = RIGHT,
41          .group_brightness = 0
42      };
43
44      set_led_group(&led_group_collection, led_group_one, LEFT);
45      set_led_group(&led_group_collection, led_group_two, RIGHT);
46
47      return led_group_collection;
48  }
49
50  /**
51   * @brief Get a led group
52   *
53   * @param led_group_collection
54   * @param direction
55   * @return LED_Group_t*
56   */
```

```
57  LED_Group_t* get_led_group(LED_Group_Collection_t* led_group_collection, Direction_t
        direction) {
58      return &(led_group_collection->led_groups[direction]);
59  }
60
61  /**
62   * @brief Set a direction's led group
63   *
64   * @param led_group_collection
65   * @param led_group
66   * @param direction
67   */
68  void set_led_group(LED_Group_Collection_t* led_group_collection, LED_Group_t led_group,
        Direction_t direction) {
69      led_group_collection->led_groups[direction] = led_group;
70  }
71
72  /**
73   * @brief Set a led group brightnesses
74   *
75   * @param led_group_collection
76   */
77  void set_led_group_brightnesses(LED_Group_Collection_t* led_group_collection, Load_Cell_t
        * load_cell) {
78
79      led_group_collection->mode_changed = false;
80
81      uint8_t group_one_brightness;
82      uint8_t group_two_brightness;
83
84      if (led_group_collection->mode == FOLLOW) {
85          group_one_brightness = (uint8_t) round((map_voltage_to_range(load_cell->voltage,
                load_cell->max_voltage, 1) + 0.5) * ARDUINO_PWM_WRITE_RANGE);
86          group_two_brightness = (ARDUINO_PWM_WRITE_RANGE - group_one_brightness);
87      } else if (led_group_collection->mode == FULL) {
88          group_one_brightness = 255;
89          group_two_brightness = 255;
90      }
91
92      set_group_brightness(get_led_group(led_group_collection, LEFT), group_one_brightness)
            ;
93      set_group_brightness(get_led_group(led_group_collection, RIGHT), group_two_brightness
            );
94  }
```

Listing 5: LED group collection code.

```
1   /**
2    * @file LED_Group_Collection.h
3    * @author your name (you@domain.com)
4    * @brief Provides the types and function prototypes required for the collection of LED
         groups.
5    * @version 0.1
6    * @date 2021-08-13
7    *
8    * @copyright Copyright (c) 2021
9    *
10   */
11
12  #ifndef LED_GROUP_COLLECTION_H
13  #define LED_GROUP_COLLECTION_H
14
15  #if defined(__cplusplus)
16  extern "C" {
17  #endif
18
19  #include <stdint.h>
```

```
20  #include <stdbool.h>
21
22  #include "voltage.h"
23
24  #include "Load_Cell.h"
25  #include "LED_Group.h"
26
27  #define NUM_MODES 2
28
29  typedef enum {
30      FOLLOW = 0,
31      FULL
32  } Light_Mode_t;
33
34  typedef struct {
35      LED_Group_t led_groups[2];
36      Light_Mode_t mode;
37      bool mode_changed;
38  } LED_Group_Collection_t;
39
40  LED_Group_Collection_t init_led_group_collection(uint8_t left_pin_number, uint8_t
        right_pin_number);
41  LED_Group_t* get_led_group(LED_Group_Collection_t* led_group_collection, Direction_t
        direction);
42  void set_led_group(LED_Group_Collection_t* led_group_collection, LED_Group_t led_group,
        Direction_t direction);
43  void set_led_group_brightnesses(LED_Group_Collection_t* led_group_collectionm,
        Load_Cell_t* load_cell);
44
45  #if defined(__cplusplus)
46  }
47  #endif
48
49  #endif // LED_GROUP_COLLECTION_H
```

Listing 6: LED group collection header code.

```
1   /**
2    * @file Load_Cell.c
3    * @author Alex Stiles
4    * @brief Contains the source code for the TAL221 load cell.
5    * @version 0.1 alpha
6    * @date 2021-08-13
7    *
8    * @copyright Copyright (c) 2021
9    *
10   * This source file provides the functionality to poll the load cell.
11   * Polling the load cell updates the voltage and dependant variables if required.
12   *
13   */
14
15  #include <stdint.h>
16  #include <stdbool.h>
17  #include <math.h>
18
19  #include <stdio.h>
20
21  #include "Load_Cell.h"
22
23  /**
24   * @brief Create a load cell, with the pin set to LOAD_CELL_ANALOG_PIN and the deviation
        voltage breakpoint to DEVIATION_VOLTAGE_BREAKPOINT
25   *
26   * @return Load_Cell_t
27   */
28  Load_Cell_t init_load_cell(uint8_t load_cell_pin_number) {
29      Load_Cell_t load_cell = {
```

```
30          .voltage = COMMON_MODE_VOLTAGE,
31          .strain = 0,
32          .angle = 0,
33          .pin_number = load_cell_pin_number,
34          .deviation_voltage_breakpoint = DEVIATION_VOLTAGE_BREAKPOINT,
35          .max_voltage = ARDUINO_MAX_VOLTAGE,
36          .strain_range = STRAIN_RANGE,
37          .angle_range = ANGLE_RANGE
38      };
39
40
41      return load_cell;
42  }
43
44  /**
45   * @brief Poll the load cell (by reading the voltage input at the LOAD_CELL_ANALOG_PIN)
            and compare it to the currently recorded voltage.
46   * The methodology for detecting noise is rudimentary, lacking any requirement for the
            voltage to remain in a constrained range for a given time, reminiscent of button
            debouncing.
47   * A more straightforward method is utilised due to hardware filtering using an active
            low pass filter. See shield schematic for details.
48   *
49   * @param load_cell pointer to Load_Cell_t struct
50   * @return Whether a voltage deviation was read
51   */
52  bool poll_sensor(Load_Cell_t* load_cell, float read_voltage) {
53      bool deviation = false;
54
55      // Compare to the current voltage to see if deviation detected
56
57      if (fabsf(load_cell->voltage - read_voltage) > load_cell->
            deviation_voltage_breakpoint) {
58          // Update the recorded voltage of the load cell
59          load_cell->voltage = read_voltage;
60          // Update the strain
61          load_cell->strain = map_voltage_to_range(load_cell->voltage, load_cell->
                max_voltage, load_cell->strain_range);
62          // Update the angle
63          load_cell->angle = map_voltage_to_range(load_cell->voltage, load_cell->
                max_voltage, load_cell->angle_range);
64          deviation = true;
65      }
66
67      return deviation;
68  }
```

Listing 7: Load cell code.

```
1  /**
2   * @file Load_Cell.h
3   * @author Alex Stiles
4   * @brief Provides the types and function prototypes required for the TAL221 load cell.
5   * @version 0.1 alpha
6   * @date 2021-08-13
7   *
8   * @copyright Copyright (c) 2021
9   *
10  */
11
12  #ifndef LOAD_CELL_H
13  #define LOAD_CELL_H
14
15  #if defined(__cplusplus)
16  extern "C" {
17  #endif
18
```

```
19  #include <stdint.h>
20  #include <stdbool.h>
21
22  #include "voltage.h"
23
24  #define STRAIN_RANGE 200
25  #define ANGLE_RANGE 180
26  #define DEVIATION_VOLTAGE_BREAKPOINT 0.1
27
28  typedef struct {
29      // State of load cell at given time
30      float voltage;
31      int8_t strain;
32      int8_t angle;
33
34      // Specific details of load cell
35      uint8_t pin_number;
36      float deviation_voltage_breakpoint;
37      float max_voltage;
38      uint8_t strain_range;
39      uint8_t angle_range;
40  } Load_Cell_t;
41
42  Load_Cell_t init_load_cell(uint8_t load_cell_pin_number);
43  bool poll_sensor(Load_Cell_t* load_cell, float read_voltage);
44
45  #if defined(__cplusplus)
46  }
47  #endif
48
49  #endif // LOAD_CELL_H
```

Listing 8: Load cell header code.

```
 1  #include <stdio.h>
 2  #include <stdint.h>
 3  #include <stdbool.h>
 4  #include <stdlib.h>
 5
 6  #include "voltage.h"
 7
 8  #include "LED_Group_Collection.h"
 9  #include "LED_Group.h"
10  #include "Load_Cell.h"
11  #include "button.h"
12
13  #define LOAD_CELL_ANALOG_PIN 0
14
15  #define LEFT_GROUP_PIN 5
16  #define RIGHT_GROUP_PIN 6
17
18  #define BUTTON_PIN 7
19
20  #if !defined(ARDUINO)
21
22  LED_Group_Collection_t led_group_collection;
23  Load_Cell_t load_cell;
24  Button_t button;
25
26  void callbackFunction() {
27      led_group_collection.mode = ++led_group_collection.mode % NUM_MODES;
28  }
29
30  int main(void) {
31      led_group_collection = init_led_group_collection(LEFT_GROUP_PIN, RIGHT_GROUP_PIN);
32      load_cell = init_load_cell(LOAD_CELL_ANALOG_PIN);
33      button = init_button(BUTTON_PIN, &callbackFunction);
```

```
34
35      long unsigned current_time = 0;
36      bool button_state = 0;
37
38      debounce_state(&button, button_state, current_time);
39
40      if (poll_sensor(&load_cell, 3.75)) {
41          set_led_group_brightnesses(&led_group_collection, &load_cell);
42
43          printf("Load Cell Voltage: %.1f/5 V\n", load_cell.voltage);
44          printf("Left Group Brightness: %d/255 | Right Group Brightness: %d/255\n",
45              get_group_brightness(get_led_group(&led_group_collection, LEFT)),
                get_group_brightness(get_led_group(&led_group_collection, RIGHT)));
          printf("Load Cell Strain: %d grams\n", load_cell.strain);
46          printf("Head Angle: %d deg\n", load_cell.angle);
47      }
48
49      return EXIT_SUCCESS;
50  }
51
52  #endif
```

Listing 9: Main code.

```
1   /**
2    * @file voltage.c
3    * @author Alex Stiles
4    * @brief Provides helper functions when dealing with Arduino voltage
5    * @version 0.1
6    * @date 2021-08-13
7    *
8    * @copyright Copyright (c) 2021
9    *
10   */
11
12  #include "math.h"
13
14  #include "voltage.h"
15
16  float map_voltage_to_range(float voltage, float max_voltage, float range) {
17      return ((voltage - 2.5) / max_voltage) * range;
18  }
```

Listing 10: Voltage ADC code.

```
1   /**
2    * @file voltage.h
3    * @author Alex Stiles
4    * @brief  Contains definations regarding the max voltage provided by the arduino and the
            common mode.
5    * @version 1
6    * @date 2021-08-13
7    *
8    * @copyright Copyright (c) 2021
9    *
10   */
11
12  #ifndef VOLTAGE_H
13  #define VOLTAGE_H
14
15  #define ARDUINO_MAX_VOLTAGE 5
16  #define ARDUINO_PWM_READ_RANGE 1023
17  #define ARDUINO_PWM_WRITE_RANGE 255
18  #define COMMON_MODE_VOLTAGE 2.5
19
20  float map_voltage_to_range(float voltage, float max_voltage, float range);
21
```

```
22   #endif // VOLTAGE_H
```
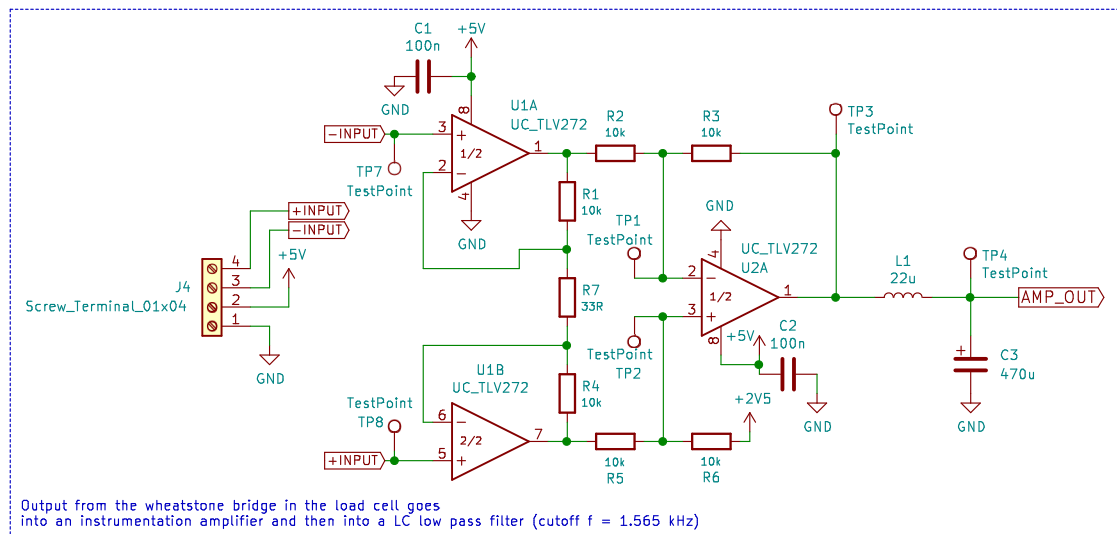
Listing 11: Voltage ADC header code.

```
1    /**
2     * @file main.ino
3     * @author Alex Stiles
4     * @details
5     * @brief Entry point. Contains high level logic and polling loop.
6     * @version 0.1 alpha
7     * @date 2021-08-13
8     *
9     * @copyright Copyright (c) 2021
10    *
11    * The entry point for compilation when compiling on Arduino.
12    * Contains the setup() and loop() functions required for the Arduino.
13    * C headers included are appropriately linked.
14    *
15    */
16
17   #include <stdio.h>
18   #include <stdint.h>
19   #include <stdbool.h>
20   #include <stdlib.h>
21
22   #include "voltage.h"
23
24   #include "LED_Group_Collection.h"
25   #include "LED_Group.h"
26   #include "Load_Cell.h"
27   #include "button.h"
28
29   #define LOAD_CELL_ANALOG_PIN 0
30
31   #define LEFT_GROUP_PIN 5
32   #define RIGHT_GROUP_PIN 6
33
34   #define BUTTON_PIN 7
35
36   LED_Group_Collection_t led_group_collection;
37   Load_Cell_t load_cell;
38   Button_t button;
39
40   /* Button */
41
42   void callbackFunction() {
43       led_group_collection.mode = (Light_Mode_t) ((led_group_collection.mode + 1) %
            NUM_MODES);
44       led_group_collection.mode_changed = true;
45   }
46
47   /* Load Cell */
48
49   float read_load_cell_voltage() {
50       return (analogRead(LOAD_CELL_ANALOG_PIN) / (float) ARDUINO_PWM_READ_RANGE) *
            ARDUINO_MAX_VOLTAGE;
51   }
52
53   /* LED Groups */
54
55   void update_led_group_pwm_signals() {
56       set_led_group_brightnesses(&led_group_collection, &load_cell);
57
58       Serial.println("Left: " + String(get_group_brightness(get_led_group(&
            led_group_collection, LEFT))));
59       Serial.println("Right: " + String(get_group_brightness(get_led_group(&
            led_group_collection, RIGHT))));
```
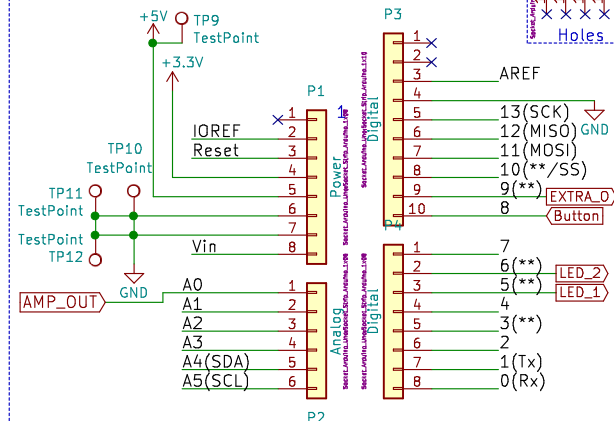
```
60
61     analogWrite(LEFT_GROUP_PIN, get_group_brightness(get_led_group(&led_group_collection,
           LEFT)));
62     analogWrite(RIGHT_GROUP_PIN, get_group_brightness(get_led_group(&led_group_collection
           , RIGHT)));
63 }
64
65 void setup() {
66     Serial.begin(9600);
67
68     led_group_collection = init_led_group_collection(LEFT_GROUP_PIN, RIGHT_GROUP_PIN);
69     load_cell = init_load_cell(LOAD_CELL_ANALOG_PIN);
70     button = init_button(BUTTON_PIN, &callbackFunction);
71
72     pinMode(LOAD_CELL_ANALOG_PIN, INPUT);
73
74     pinMode(LEFT_GROUP_PIN, OUTPUT);
75     pinMode(RIGHT_GROUP_PIN, OUTPUT);
76 }
77
78 void loop() {
79     long unsigned current_time = millis();
80     bool button_state = digitalRead(BUTTON_PIN);
81
82     debounce_state(&button, button_state, current_time);
83
84     if (poll_sensor(&load_cell, read_load_cell_voltage()) || led_group_collection.
           mode_changed) {
85        update_led_group_pwm_signals();
86
87        Serial.println("Load Cell Voltage: " + String(load_cell.voltage));
88        Serial.println("Load Cell Strain: " + String(load_cell.strain));
89        Serial.println("Load Cell Angle: " + String(load_cell.angle));
90     }
91
92     delay(20);
93 }
```
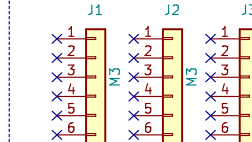
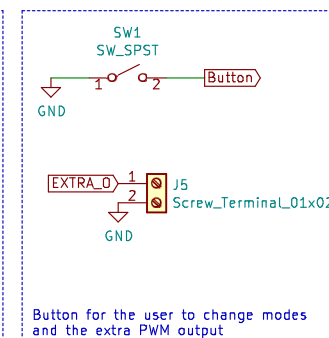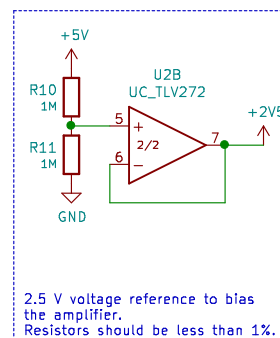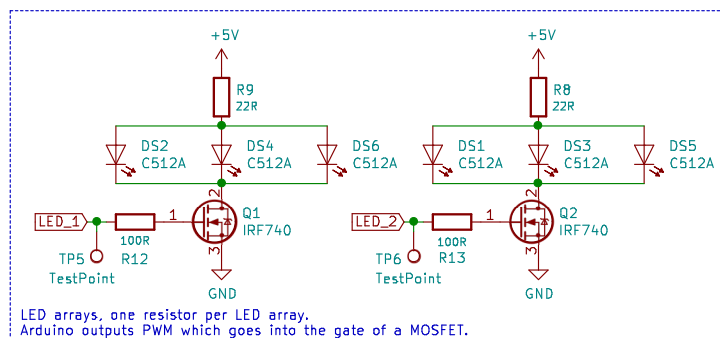Listing 12: Main code.

C   Schematic

Arduino Uno R3 Shield Connectors

Holes

P5 P6 P7 P8

P3
1 ×
2 ×
3 AREF
4
5 13(SCK)
6 12(MISO) GND
7 11(MOSI)
8 10(**/SS)
9 9(**) EXTRA_O
10 8 Button

P1 Power
IOREF 1 ×
2
Reset 3
4
5
6
7
Vin 8

P4
1 7
2 6(**) LED_2
3 5(**) LED_1
4 4
5 3(**)
6 2
7 1(Tx)
8 0(Rx)

P2 Analog Digital
A0 1
A1 2
A2 3
A3 4
A4(SDA) 5
A5(SCL) 6

+5V TP9 TestPoint
+3.3V

TP10 TestPoint

TP11 TestPoint

TestPoint TP12

AMP_OUT

GND

Prototyping Pads

J1 M3 J2 M3 J3 M3
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6

KiCad power flags

+5V PWR_FLAG

+3.3V PWR_FLAG

PWR_FLAG

GND

C1 100n +5V
GND

U1A UC_TLV272
−INPUT 3 +
2 − 1/2 1
TP7 TestPoint
GND
GND

R2 10k   R3 10k
TP3 TestPoint

R1 10k
TP1 TestPoint
GND

UC_TLV272 U2A
4
2 −
3 + 1/2 1
8
+5V

R7 33R
TestPoint TP2

L1 22u
TP4 TestPoint
AMP_OUT

C2 100n
+2V5
GND

C3 470u
GND

J4 Screw_Terminal_01x04
4 +INPUT
3 −INPUT
2 +5V
1
GND

+INPUT
−INPUT
+5V

U1B UC_TLV272
6 −
5 + 2/2 7
TestPoint TP8
R4 10k
+INPUT

R5 10k   R6 10k

Output from the wheatstone bridge in the load cell goes
into an instrumentation amplifier and then into a LC low pass filter (cutoff f = 1.565 kHz)

+5V R9 22R
DS2 C512A DS4 C512A DS6 C512A
LED_1 R12 100R 1
Q1 IRF740
TP5 TestPoint
GND

+5V R8 22R
DS1 C512A DS3 C512A DS5 C512A
LED_2 R13 100R 1
Q2 IRF740
TP6 TestPoint
GND

LED arrays, one resistor per LED array.
Arduino outputs PWM which goes into the gate of a MOSFET.

+5V
R10 1M
R11 1M
GND

U2B UC_TLV272
5 +
6 − 2/2 7
+2V5

2.5 V voltage reference to bias
the amplifier.
Resistors should be less than 1%.

SW1 SW_SPST
1 2 Button
GND

EXTRA_O 1
2 J5 Screw_Terminal_01x02
GND

Button for the user to change modes
and the extra PWM output

(75953196) & (18120890)
Edward Manson & Alex Stiles

Sheet: /
File: ENEL200ShieldGroup10.sch

Title: Arduino Shield Headlight

Size: A4 | Date: 2021-08-17 | **Rev: A**
KiCad E.D.A. kicad 5.1.10−1.fc34 | Id: 1/1

D   Printed Circuit Board