Topic:
3. Sensors and Actuators

Summary session:

This session will introduce the third lab of the module. The third lab of the module is about the sensors and actuators. In this lab, you will get a hands-on introduction to sensors and actuators. You will gain an understanding of controlling light with PWM and distinguish the difference between digital and analogue connections.

WARM-UP ACTIVITY

Time: 5'

1. Revision of last week's blog posts.
2. Revision of last week's code.

BLOGGING

Time: 5' (setup), 110' (full activity)

1. **Write your third blog post at https://matd3039.our.dmu.ac.uk/**
    1. A good practice is to write a blog post during the whole lab session. This way, you can keep notes of the whole process, which can be useful for later.
    2. Start your third blog post of the module.
    3. Make sure to keep writing from the beginning to end of the lab session with key ideas and findings.

BREADBOARD

Time: 5'

1. Revise how to use a breadboard skimming through this blog post: https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all

PUSH BUTTON TO CONTROL THE LED

In the blinking an LED example seen last week, the LED was your actuator, and the Arduino was controlling it. We will focus now on bringing a sensor to complete the picture. We are going to use the simplest form of sensor available: a pushbutton switch.

Time: 20'

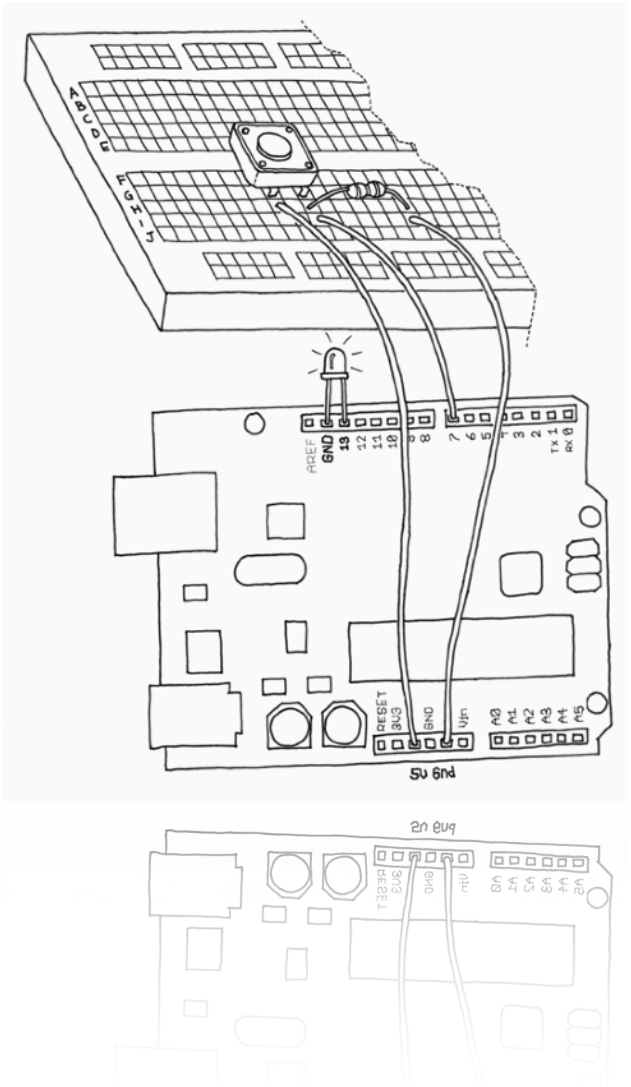
2. Understand how a push button works:
    1. It is a very simple device: two bits of metal kept apart by a spring, and a plastic cap that when pressed brings the two bits of metal into contact. When the bits of metal are apart, there is no circulation of current in the pushbutton; when you press it, you make a connection. You can do a quick test with a handheld tester, crocodile wires and a push button.
    2. All switches are basically just this: two (or more) pieces of metal that can be brought into contact with each other, allowing electricity to flow from one to the other, or separated, preventing the flow of electricity.
    3. To monitor the state of a switch, there's an Arduino instruction: the `digitalRead()` function that we saw last week. `digitalRead()` checks to see whether there is any voltage applied to the pin that you specify between parentheses, and returns a value of

HIGH or LOW, depending on its findings. Last week we were using this instruction with the following code:

```
void checkstatus() {
  if (digitalRead(LED) == HIGH) {
    Serial.write('H');         // print the value to the serial port
  }
  else {
    Serial.write('L');         // print the value to the serial port
  };
}
```

4. With `digitalRead()`, you can “ask a question” of Arduino and receive an answer that can be stored in memory somewhere and used to make decisions immediately or later.

3. Build the following circuit with the following parts:
    1. Solderless breadboard.
    2. Jumper wires.
    3. One 10K Ohm resistor.
    4. One tactile pushbutton switch.



4. Write the following code on the Arduino IDE software to turn on LED while the button is pressed. Save it as “PushButtonControl”.

```
// Turn on LED while the button is pressed

const int LED = 13;   // the pin for the LED
const int BUTTON = 7; // the input pin where the
                      // pushbutton is connected
int val = 0;          // val will be used to store the state
                      // of the input pin

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop(){
  val = digitalRead(BUTTON); // read input value and store it

  // check whether the input is HIGH (button pressed)
  if (val == HIGH) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

The if statement is possibly the most important instruction in a programming language, because it allows a computer (and remember, the Arduino is a small computer) to make decisions. After the if keyword, you have to write a “question” inside parentheses, and if the “answer”, or result, is true, the first block of code will be executed; otherwise, the block of code after else will be executed.

Notice that the == symbol is very different from the = symbol. The former is used when two entities are compared, and returns true or false; the latter assigns a value to a constant or a variable.

It’s important to realise that the switch is not connected directly to the LED. Your Arduino sketch inspects the switch, and then makes a decision as to whether to turn the LED on or off. The connection between the switch and the LED is really happening in your sketch.

One advantage of digital, programmable electronics is that we can implement different “behaviours” using the same electronic circuit just by changing the software.

PUSH BUTTON TO CONTROL THE LED

Next, you will create a new version of this code where you can turn on the LED when the button is pressed and keep it on after it is released. To do this, you’re going to use what is called a variable.

Time: 20’

1. Create a copy and save it as “PushButtonControl_Pressed”.
2. Write the following code on the Arduino IDE software to turn on the LED when the button is pressed and keep it on after it is released.

```
// Turn on LED when the button is pressed and keep it on after it
is released
```

```
const int LED = 13;   // the pin for the LED
const int BUTTON = 7; // the input pin where the
                      // pushbutton is connected
int val = 0;          // val will be used to store the state
                      // of the input pin
int state = 0;        // 0 = LED off while 1 = LED on

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop(){
  val = digitalRead(BUTTON); // read input value and store it

  // check if the input is HIGH (button pressed)
  // and change the state
  if (val == HIGH) {
    state = 1 - state;
  }

  if (state == 1) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

`int val = 0;`
`int` means that your variable will store an integer number, `val` is the name of the variable, and = `0` assigns it an initial value of zero.

In the above program, the variable `val` stores the result of `digitalRead();` whatever Arduino gets from the input ends up in the variable and will stay there until another line of code changes it.

`int state = 0;`
The variable state remembers whether the LED has to stay on or off after we release the button.

Notice that variables use a type of memory called RAM. It is quite fast, but when you turn off your board, all data stored in RAM is lost (which means that each variable is reset to its initial value when the board is powered up again). Your programs themselves are stored in flash memory, which retains its content even when the board is off.

3. Let's analyse the code with more detail.
    1. `state` is a variable that stores either 0 or 1 to remember whether the LED is on or off. After the button is released, we initialise it to 0 (LED off).
    2. Later, we read the current state of the button, and if it's pressed `(val == HIGH)`, we change state from 0 to 1, or vice versa. We do this using a small trick, as state can be only either 1 or 0. The trick used involves a small mathematical expression based on the idea that 1 – 0 is 1 and 1 – 1 is 0: `state = 1 - state;` The new value of state is assigned the value of 1 minus the old value of state.
    3. Later in the program, you can see that we use state to figure out whether the LED has to be on or off.
    4. The results are flaky because of the way we read the button. Arduino is really fast; it executes its own internal instructions at a rate of 16 million per second. While your

finger is pressing the button, Arduino might be reading the button's position a few thousand times and changing state accordingly. So the results end up being unpredictable; it might be off when you wanted it on, or vice versa.

4. Let's improve the code.

```
// Turn on LED when the button is pressed and keep it on after it
is released (improved)

const int LED = 13;   // the pin for the LED
const int BUTTON = 7; // the input pin where the
                      // pushbutton is connected
int val = 0;          // val will be used to store the state
                      // of the input pin
int old_val = 0;      // this variable stores the previous
                      // value of "val"
int state = 0;        // 0 = LED off while 1 = LED on

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop(){
  val = digitalRead(BUTTON); // read input value and store it

  // check if there was a transition
  if ((val == HIGH) && (old_val == LOW)){
    state = 1 - state;
  }

  old_val = val;  // val is now old, let's store it

  if (state == 1) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

Here you detect the exact moment when the button is pressed—that is the only moment that you have to change state. The way we like to do it is to store the value of `val` before we read a new one; this allows you to compare the current position of the button with the previous one and change state only when the button changes from LOW to HIGH.

5. This is better but there is still a little problem: there's bouncing from the two pieces of metal inside the pushbutton. Although the bouncing is only for a very small distance and happens for a fraction of a second, it causes the switch to change between off and on a number of times until the bouncing stops, and Arduino is quick enough to catch this.
6. When the pushbutton is bouncing, the Arduino sees a very rapid sequence of on/off signals. There are many techniques developed to do debouncing, here's one potential solution.

```
// Turn on LED when the button is pressed and keep it on after it
is released (improved with debouncing)

const int LED = 13;   // the pin for the LED
```

```
const int BUTTON = 7; // the input pin where the
                      // pushbutton is connected
int val = 0;          // val will be used to store the state
                      // of the input pin
int old_val = 0;      // this variable stores the previous
                      // value of "val"
int state = 0;        // 0 = LED off while 1 = LED on

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop(){
  val = digitalRead(BUTTON); // read input value and store it

  // check if there was a transition
  if ((val == HIGH) && (old_val == LOW)){
    state = 1 - state;
    delay(10);
  }

  if ((val == LOW) && (old_val == HIGH)) {
    delay(10);
  }

  old_val = val;  // val is now old, let's store it

  if (state == 1) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

In this code we add a 10- to 50-millisecond delay when the code detects a transition. In other words, you just wait a bit for the bouncing to stop.
With this circuit, you've been able to read digital input and control digital output.

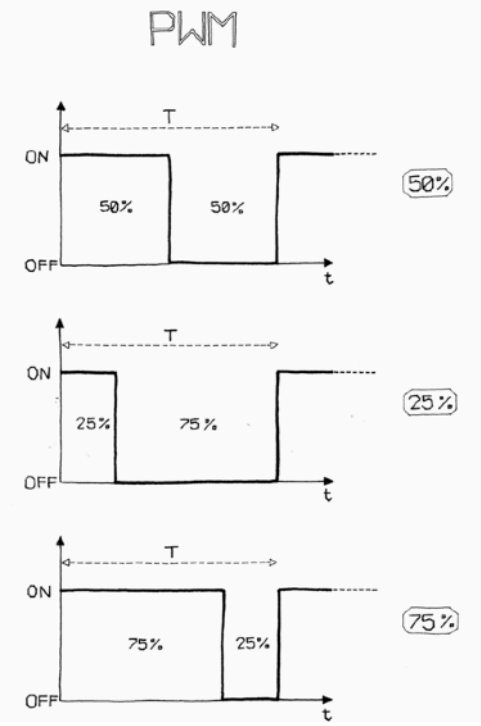
Note: you can consider replacing the pushbutton with another digital sensor. For example, with a passive infrared (PIR) sensor, you could make your lamp respond to the presence of human beings, or you could use a tilt switch to build one that turns off when it's tilted on one side.

CONTROLLING LIGHT WITH PWM

To make it more interesting, we can simulate a dimmable light taking advantage of the effect of Persistence of Vision, or POV.

By changing delay times you will notice that if you make the on delay different from the off delay, you can make the LED brighter by leaving it on for longer, and you can make the LED dimmer by leaving it off for longer.

This technique is called pulse-width modulation, or PWM, because you are changing the LED's brightness by modulating (or changing) the width of the pulse. This works because our eyes can't see distinct pictures if they change too fast.



The microcontroller used by your Arduino has a piece of hardware that can very efficiently blink your LEDs while your sketch does something else. On the Uno, this hardware is implemented on pins 3, 5, 6, 9, 10, and 11. The `analogWrite()` instruction is used to control this hardware.

For example, writing `analogWrite(9,50)` will set the brightness of an LED connected to pin 9 to quite dim, while writing `analogWrite(9,200)` will set the brightness of the LED to quite bright. `analogWrite()` takes a number between 0 and 255, where 255 means full brightness and 0 means off.

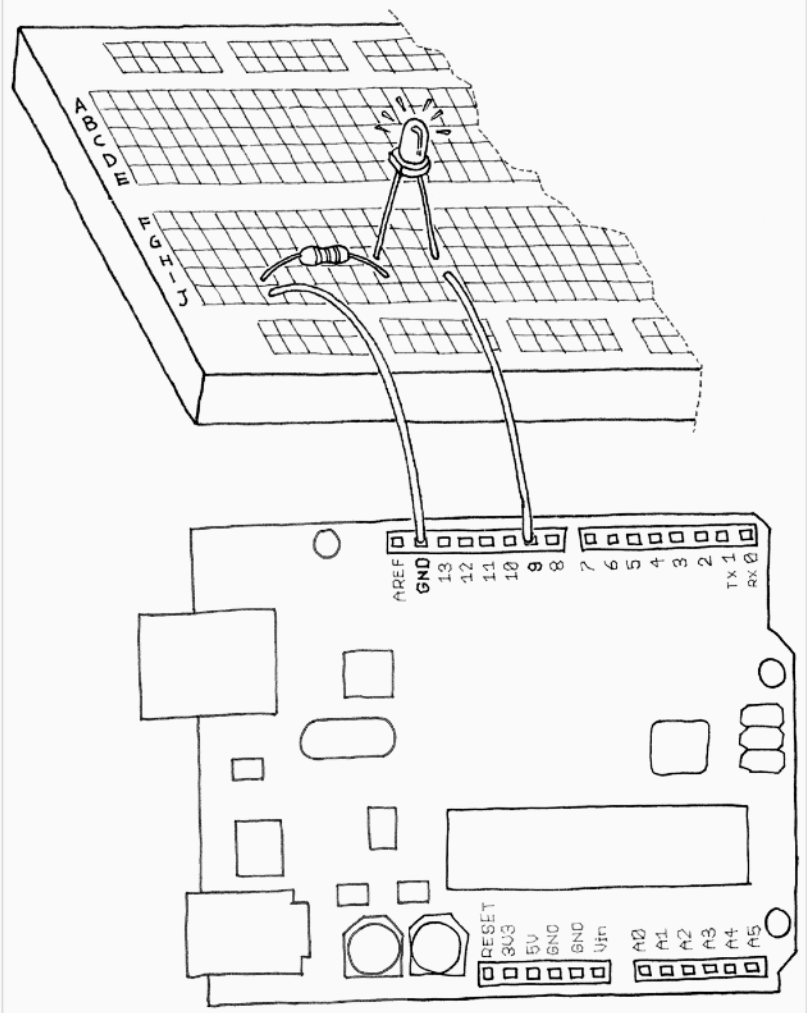
Time: 20'

1. Build the following circuit with the following parts:
    1. LED of any colour.
    2. Some 220 Ohm resistors.

Note that LEDs are polarised, which means they care which way the electric current goes through them. The long lead indicates the anode, or positive lead, and in our case should go to the right, which connects it to pin 9 of the Arduino. The short lead indicates the cathode, or negative lead, and in this case it should go to the left, connecting it to the resistor.

Advice: You should always use a resistor with an LED to prevent burning out the LED. Use a 220-ohm resistor (red-red-brown).

2. Create a new sketch in Arduino. Save it as "Fading_in_out_LED". Add the following code:




```
// Fade an LED in and out, like on a sleeping Apple computer

const int LED = 9; // the pin for the LED
int i = 0;         // We'll use this to count up and down

void setup() {
  pinMode(LED, OUTPUT); // tell Arduino LED is an output
}

void loop(){

  for (i = 0; i < 255; i++) { // loop from 0 to 254 (fade in)
    analogWrite(LED, i);      // set the LED brightness
    delay(10); // Wait 10ms because analogWrite
               // is instantaneous and we would
               // not see any change
  }

  for (i = 255; i > 0; i--) { // loop from 255 to 1 (fade out)

    analogWrite(LED, i); // set the LED brightness
    delay(10);           // Wait 10ms
  }
}
```

3. Upload the sketch, and the LED will fade up and then fade down continuously.
4. Let's understand the code:
    1. `analogWrite()` changes the LED brightness. The other important part is the `for loop`: it repeats the `analogWrite()` and the `delay()` over and over, each time using a different value for the variable i as follows.
    2. The first for loop starts the variable `i` with the value of 0, and increases it up to 255, which fades the LED up to full brightness.
    3. The second for loop starts the variable `i` with the value of 255, and decreases it up to 0, which fades the LED all the way down to completely off.
    4. After the second for loop, Arduino starts our `loop()` function over again.
    5. The `delay()` is just to slow things down a bit so you can see the changing brightness; otherwise, it would happen too fast.
5. Add the circuit we used to read a button to this breadboard. You will basically combine the circuit you just built with the pushbutton circuit.

If you have just one pushbutton, how do you control the brightness of a lamp? You can detect how long a button has been pressed. To do this, we need to upgrade the previous code to add dimming. The idea is to build an interface in which a press-and-release action switches the light on and off, and a press-and-hold action changes brightness.

3. Create a new sketch in Arduino. Save it as "Change_brightness_LED_PushButtonControl". Add the following code:

```
// Change the brightness as you hold the button

const int LED = 9;     // the pin for the LED
const int BUTTON = 7;  // input pin of the pushbutton

int val = 0;     // stores the state of the input pin

int old_val = 0; // stores the previous value of "val"
int state = 0;   // 0 = LED off while 1 = LED on

int brightness = 128;        // Stores the brightness value
unsigned long startTime = 0; // when did we begin pressing?

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop() {

  val = digitalRead(BUTTON); // read input value and store it

  // check if there was a transition
  if ((val == HIGH) && (old_val == LOW)) {

    state = 1 - state; // change the state from off to on
                       // or vice-versa

    startTime = millis(); // millis() is the Arduino clock
                          // it returns how many milliseconds
                          // have passed since the board has
                          // been reset.
```

```
      // (this line remembers when the button
      // was last pressed)
      delay(10);
    }

    // check whether the button is being held down
      if ((val == HIGH) && (old_val == HIGH)) {

        // If the button is held for more than 500 ms.
        if (state == 1 && (millis() - startTime) > 500) {
         brightness++; // increment brightness by 1
         delay(10);    // delay to avoid brightness going
                       // up too fast
           if (brightness > 255) { // 255 is the max brightness
             brightness = 0; // if we go over 255
                             // let's go back to 0
          }
       }
     }

     old_val = val; // val is now old, let's store it

     if (state == 1) {
       analogWrite(LED, brightness); // turn LED ON at the
                                     // current brightness level
     } else {
       analogWrite(LED, 0); // turn LED OFF
     }
  }
```

`if (state == 1 && (millis() - startTime) > 500) {`
This instruction checks to see if the button is held down for more than 500 ms by using a built-in function called `millis()`, which is just a running count of the number of milliseconds since your sketch started running. By keeping track of when the button was pressed (in the variable `startTime`), we can compare the current time to the start time to see how much time has passed.

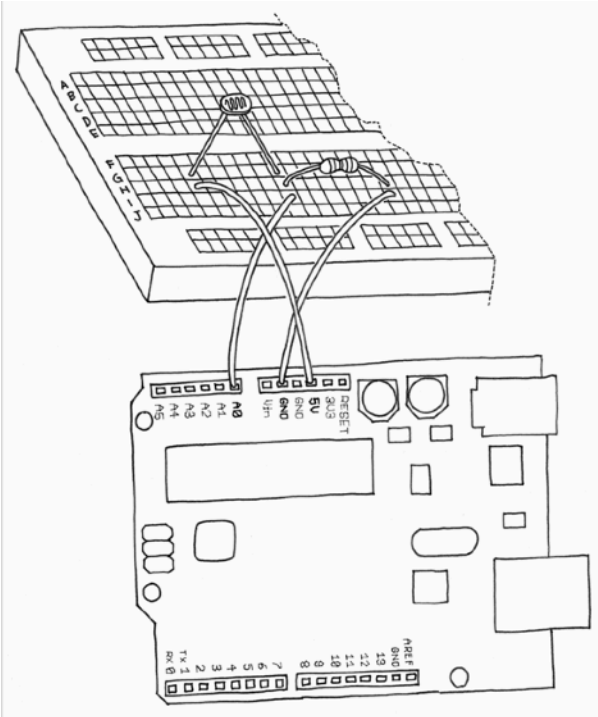
USE LIGHT SENSOR INSTEAD OF PUSHBUTTON

The light-dependent resistor (LDR) is some sort of resistor that depends on light. In darkness, the resistance of an LDR is quite high, but when you shine some light at it, the resistance quickly drops and it becomes a reasonably good conductor of electricity. It is thus a kind of light-activated switch.

Time: 20'

1. Build the following circuit with the following parts:
    1. Light-dependent resistor (LDR) or photo resistor.
    2. 10 K ohm resistor.

3. Go back to your previous code from “PushButtonControl_Pressed_Improved_Debouncing”.
4. Now carefully remove only the pushbutton, and insert the LDR into the circuit exactly where the pushbutton was. The LED should come on. Cover the LDR with your hands, and the LED turns off.
5. The light sensor that we just used is able to tell us not only whether there is light, but also how much light there is. This is the difference between an on/off or digital sensor (which tells us whether something is there or not) and an analogue sensor, which can tell us how much of something there is. In order to read this type of sensor, we need to use a special Arduino pin.
6. Turn your Arduino around so it matches the following circuit.



7. Run the following code as “Blink_LED_analogue_input”:

```
// Blink LED at a rate specified by the value of the analogue input

const int LED = 13; // the pin for the LED

int val = 0;   // variable used to store the value
               // coming from the sensor
void setup() {
  pinMode(LED, OUTPUT); // LED is as an OUTPUT

  // Note: Analogue pins are
  // automatically set as inputs
}

void loop() {

  val = analogRead(0); // read the value from
                       // the sensor

  digitalWrite(LED, HIGH); // turn the LED on
```

```
  delay(val); // stop the program for
              // some time

  digitalWrite(LED, LOW); // turn the LED off

  delay(val); // stop the program for
              // some time
}
```

8. Now add an LED to pin 9 as we did before. Because you already have some stuff on the breadboard, you'll need to find a spot on the breadboard where the LED, wires, and resistor won't overlap with the LDR circuit. You may have to move some things around, but don't worry, this is good practice because it helps your understanding of circuits and the breadboard.
9. When you are done adding the LED to your LDR circuit type the following code:

```
  digitalWrite(LED, LOW); // turn the LED off

  delay(val); // stop the program for
              // some time
}
```

Once it's running, cover and uncover the LDR and see what happens to the LED brightness.

BLOGGING

10. **Complete your blog post.**