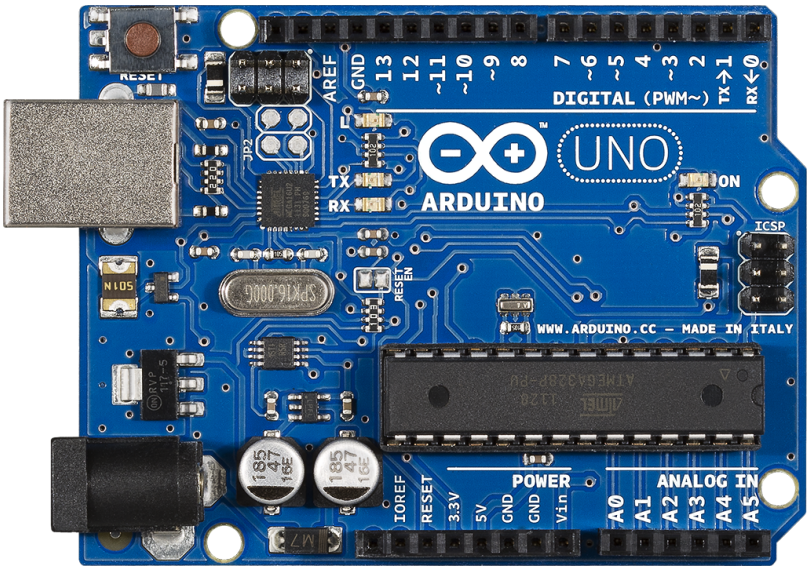
Data Acquisition and Sensor Network



Lab Manual

Instructors:
Dr. Fangning Hu

JACOBS
UNIVERSITY

Contents

Chapter 1

Acquire Sensor Data by Arduino

1.1 Objective

In this lab, we are going to become familiar with how to acquire sensor data by Arduino.

Note: After doing each task or sub-task, call the Instructor/TA to show them your result. The grading system of this lab is as follows:

- Tasks in the lab (40%);
- Final Project (60%)

1.2 Getting Started with Arduino

1.2.1 Get Familiar with the Arduino Software

The Integrated Development Environment (IDE) of Arduino is free and open source. It is already installed on our lab computers. Please double click the Arduino icon on the desktop. A screenshot of the Arduino software is shown in Fig. 1.1.

In the Tool Bar, you can find the most useful buttons. The button-descriptions can be found in the Table 1.1.

Button	Description
	Compile and check errors
	Upload codes to Arduino
	Create a new sketch
	Open an existing sketch
	Save the current sketch
	Open the serial monitor

Table 1.1: Description of the IDE Buttons

Try and become familiar with **New**, **Open**, **Save** in the Tool Bar. Open an existing file "Blink.ino" from the menu **File** $\longrightarrow$ **Examples** $\longrightarrow$ **01.Basic** $\longrightarrow$ **Blink**. If you don't find where to open it, you can simply type in the code in Listing 1.1 in the code space and save it.

```
void setup() { // Called once to initialize
  pinMode(13, OUTPUT);    // Initialize pin 13 for
```

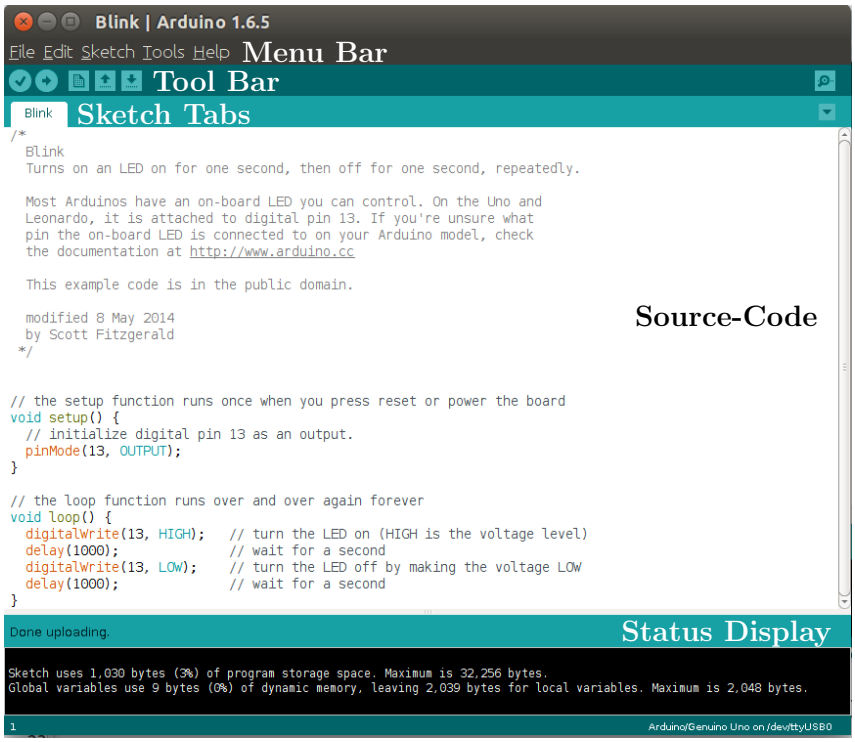


Figure 1.1: The Arduino IDE Window.

```
                          // digital-write
}

void loop() { // Called repeatedly
  digitalWrite(13, HIGH); // Set pin 13 to 5V
  delay(1000);            // Wait 1 sec = 1000 millisec.
  digitalWrite(13, LOW);  // Set pin 13 to 0V
  delay(1000);
}
```

Listing 1.1: Blinking the Light Emitting Diode (LED).

Warning: If you cut and paste code from a listing (e.g. from Listing 1.1) in this manual to the Arduino IDE, some unnecessary spaces may be introduced: e.g. "2L" could possibly become "2 L" and as a result your code does not compile. Carefully check and correct such cut and paste errors.

1.2.2 Arduino code basics: digitalWrite

The Arduino uses a slightly simplified C/C++ programming language. There are two mandatory functions: the setup() function and the loop() function.

The setup() function executes only once initially, when Arduino first powers on or right after someone presses the reset button, causing the currently loaded program to restart from the beginning. Usually, in the setup() function we configure the pins of Arduino or start communication protocols. In the above example, the Pin13 on Arduino is configured to be an output pin by function pinMode(13, OUTPUT).

The loop() function runs repeatedly until Arduino is switched off. In the above example, digitalWrite(13, HIGH) function sets pin 13 to HIGH(5V) and digitalWrite(13, LOW) sets pin 13 to LOW(0V). The Pin13 will keep HIGH or LOW for 1000 milliseconds by function delay(1000).

Now you can compile the above sketch and check whether there are errors by clicking on the Compile Button in the Tool Bar. The error messages will be presented in red in the Status Display window. In order to run this code on Arduino, we need to upload this code onto Arduino board.

1.2.3 Upload the sketch to Arduino

Arduino can communicate to our computer simply by a USB cable. Now use the USB cable to connect your Arduino to the computer. Arduino will automatically power on after connection. The driver is already installed on our computer. If everything connects properly, a green LED light (power-on light) on Arduino will turn on.

Check the menu-item **Tools** $\longrightarrow$ **Serial Port** to configure the correct serial port on which the Arduino is connected to the computer. Usually, the Arduino is installed on serial port COM3 on Windows, but not always. On Linux, this is usually /dev/ttyUSB0. You can find out the correct port by checking the Windows **Control Panel - Device Manager**.

Now click the "Upload" Button, if everything was connected correctly, a yellow (or blue, depending on the board) LED light on the Arduino board will blink every 1 second. This on-board LED is connected to Pin13.

Task 1.1 *Show the working example to the instructor/TA.*

1.3 Build up a simple circuit

Now we can use the Arduino pins to control electronic devices. Before that, we will first build a circuit on a breadboard and then connect the Arduino pins to our circuit. The first experiment is to make a LED blink as before, but now we want to use our own LED on the breadboard rather than the on-board LED.

1.3.1 The Breadboard and Jumper Wires

A breadboard is very useful to create temporary prototypes of a circuit design and to do experiments on it since it does not require soldering. In Fig. 1.2, the breadboard used in our experiments is shown.

One can also connect two points by connecting a jumper wire between these two points.

Warning: Never connect an Arduino OUTPUT pin directly to the GND.

To restrict current to $\leq 20mA$, always put a resistor $R \geq 250\Omega$ in between.

1.3.2 LED and Resistors: digitalWrite

An LED (Light Emitting Diode) is a two-lead semiconductor light source which emits light when a suitable voltage is applied to the leads. The current can only flow in one direction, i.e., from the long lead (+) to the short lead (-). BE CAREFUL of the positive and negative lead of the LED when you build a circuit.

Warning: Never connect an LED to Arduino pins directly without a resistor in between.

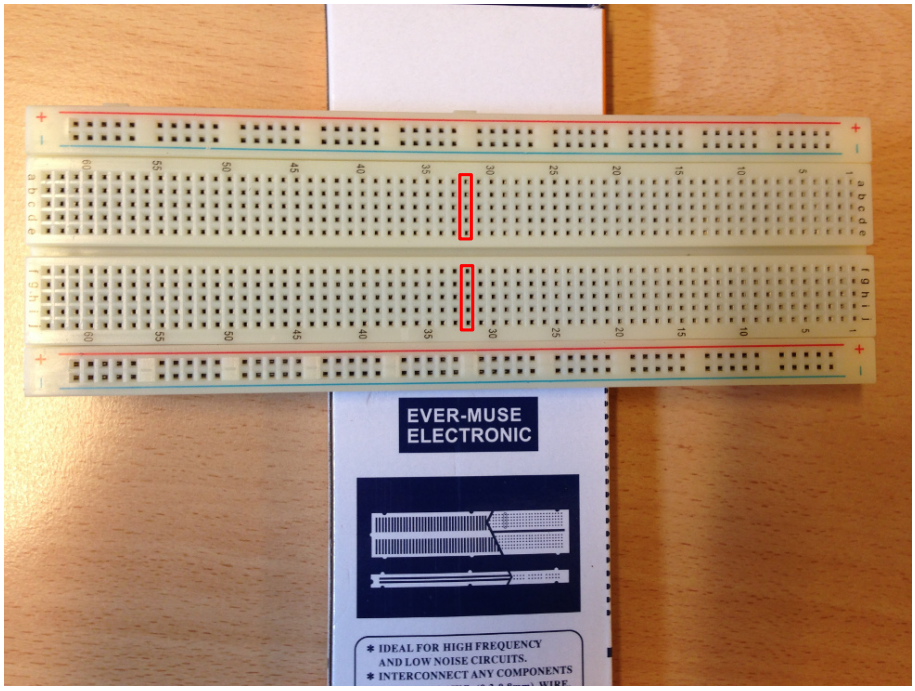


Figure 1.2: The breadboard used in the experiments. If your breadboard looks different, ask for the standard one. The top and bottom two rows are connected internally, as shown by the red and blue lines. In the middle, as shown by red rectangles, the top 5 vertical holes of each column are connected to each other. Similarly, the bottom 5 vertical holes of each column are connected to each other.

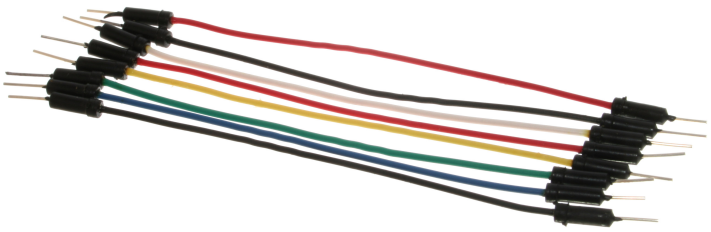


Figure 1.3: Jumper cables.

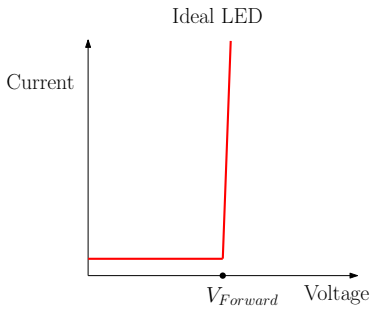
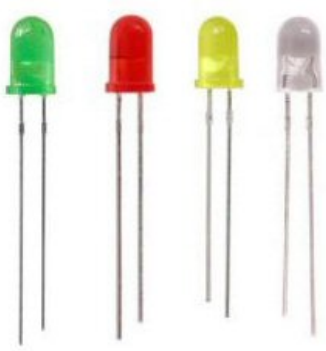


Figure 1.4: The ideal characteristics of an LED.



Different colors of LEDs have different forward-voltages (refer to Fig. 1.4). With a fixed voltage source (Arduino offers a $V_{source} = 5V$ voltage source), one can use different resistors to modify the voltage across the LED and hence keep the current flowing through it within safe limits.

Ohm's law states that the voltage (V) across a resistor is the product of the current and the resistance.

$$V_R = I \cdot R \tag{1.1}$$

By serially connecting a resistor and the LED (Figs. 1.6 and 1.5), the voltage across the resistor is V_R is:

$$V_R = V_{source} - V_{Forward} \cdot$$

$V_{Forward}$ is the forward voltage of the LED (see Fig. 1.4) and V_{source} from an Arduino output pin is typically 5 V.

To keep the current from the output pin of the Arduino as well as through the LED within safe limits, we should put a resistor R in series with the LED such that

$$I = \frac{V_R}{R} = \frac{5 - V_{Forward}}{R} \leq 20 \times 10^{-3} A. \tag{1.2}$$

LED	White	Red	Yellow	Green	Blue
$V_{Forward}$	$3.3V$	$2.1V$	$2.2V$	$3.7V$	$3.1V$
Resistance	100Ω	200Ω	200Ω	100Ω	100Ω

Table 1.2: Recommended resistor values.

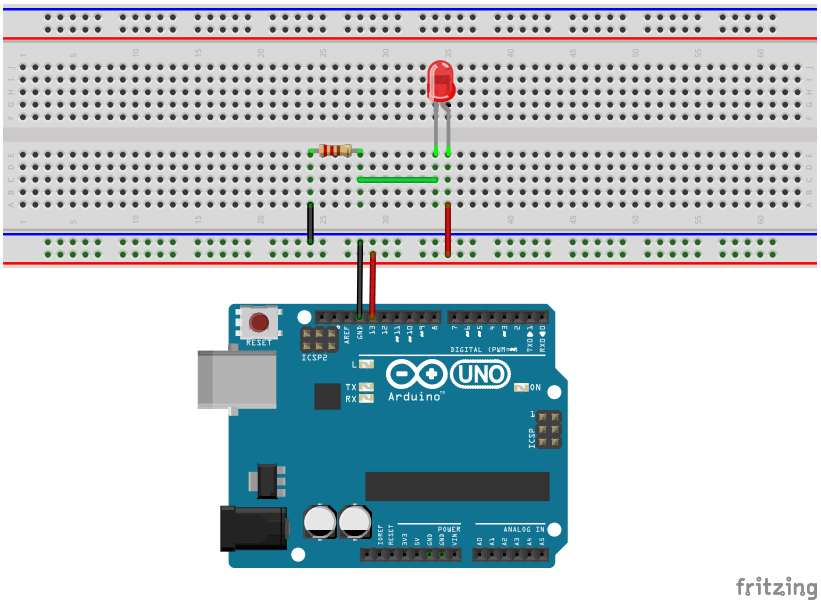


Figure 1.5: The Blink-LED Circuit with the Arduino Board.

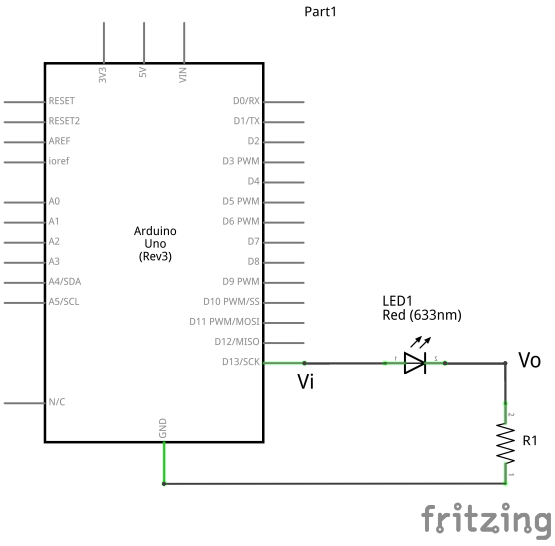


Figure 1.6: The Schematic of the Blink-LED Circuit

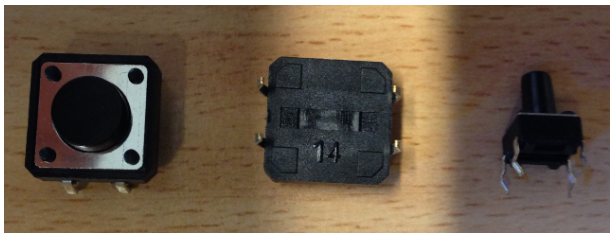


Figure 1.7: The bigger button is shown in top and bottom views. The smaller button is shown from the side. In the middle button, the shiny horizontal line just above the number "14" separates the two sides.

Table 1.2 lists $V_{Forward}$ for LEDs with different colors as well as a corresponding resistance R which satisfies (1.2).

The schematic of the circuit is as in Fig. 1.6.

Task 1.2 *Proceed as follows:*

1. *After uploading the code in Sec. 1.2.2, disconnect Arduino from the computer by pulling out the USB cable from the Arduino.*
2. *Pick an LED and the corresponding resistor from Table 1.2, serial connect them on the breadboard as shown in Figs. 1.5 and 1.6. To close the circuit, connect one end to the GND pin and another end to the Pin* 13 *of Arduino.*
3. *Get your circuit checked by the TAs or the instructors.*
4. *Now plug-in the USB cable to Arduinio. The LED should light-up every second.*

1.4 Buttons: digitalRead

Push-buttons or switches connect two points in a circuit when you press them. The push-button provided to you, shown in Fig. 1.7, has four pins. Two of them are connected to each other and form one-side of the connection; the other two are connected to each other and form the other side. When you press the button both sides are shorted.

The next example turns on the built-in LED on Pin13 when you press the button. The schematic of the circuit is shown in Fig. 1.8.

When the push-button is open (unpressed) there is no connection between the two legs of the push-button, so the pin 2 is connected to ground (through the pull-down resistor) and we read a LOW from it. When the button is closed (pressed), it makes a connection between its two legs, connecting pin-2 to 5 volts, so that we read a HIGH.

You can run the following code on Arduino and check the results.

```
/*  Button
  Turns on and off a light emitting diode(LED) connected to digital
 pin 13, when pressing a pushbutton attached to pin 2. */

// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
```

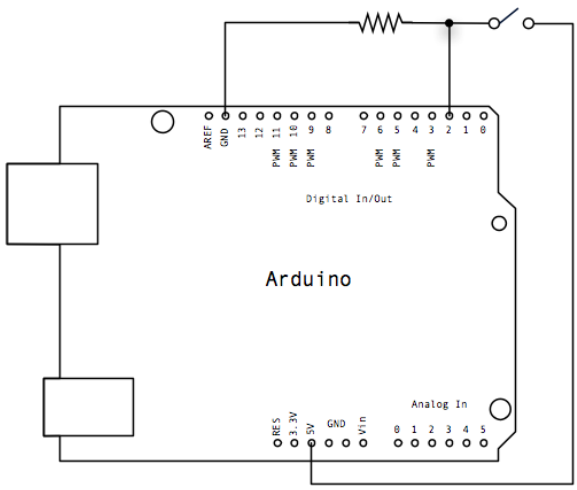


Figure 1.8: Using a push-button. Use a resistance of at least 330 Ω. We use the built-in LED connected to PIN13. Image credits: https://www.arduino.cc/en/Tutorial/Button

```
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

sourcecodes/Button/Button.ino

Notice that in the code, one can set the pin to INPUT mode in setup(). In this example, pin 2 is set to INPUT mode in order to read in the voltage value from the circuit.

Task 1.3 *Perform the following subtasks:*

- *Setup the circuit as shown in Fig. 1.8. Use a resistor of at least* 330 Ω. *Run the code on Arduino.*
- *Press the button – the built-in LED should light up.*

Task 1.4 *Connect three LEDs to Arduino, turn them on and off one by one. When the button is pressed, blink these three LEDs at the same time.*

1.5 Potentiometer and Serial Monitor: analogRead

This example shows you how to read an analog input as an integer value from 0 to 1023, convert it into voltage and print it out to the serial monitor. The outer pins of the potentiometer connect

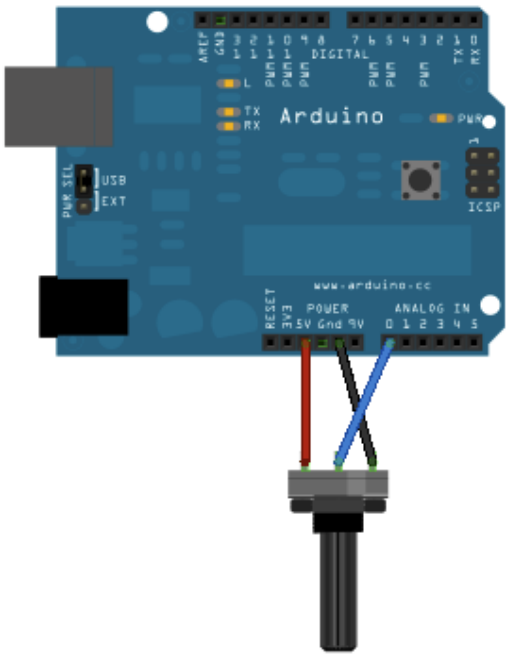


Figure 1.9: Using a potentiometer. Image credits: https://www.arduino.cc/en/Tutorial/ReadAnalogVoltage

to GND and $5V$ respectively and the middle pin to an analog input (pin $A0$). The potentiometer is a variable voltage divider - the middle pin divides the internal resistor in 2 series resistors depending on the knob position, therefore dividing the voltage applied to the outer terminals.

The potentiometer pins cannot be inserted directly into the breadboard: use a female header set, or use female-to-female cables, as shown in Fig. 1.11.

The Arduino analog pin goes to the analog-to-digital converter (ADC) inside the microcontroller that measures the voltage 0 to $5V$ and converts it into a number sensorValue between 0 and 1023 by the following command:

```
int sensorValue= analogRead(A0);
```

Note that $1024 = 2^{10}$, so the ADC resolution is 10 bits. The value of 0 corresponds to 0V and the value of 1023 to 5V. To change the values from $0 - 1023$ to a range that corresponds to the voltage the pin is reading, you'll need to create another variable, a float, and do a little math. To scale the numbers between 0.0 and 5.0, divide by 1023.0 and multiply that by sensorValue:

```
float voltage= sensorValue * (5.0 / 1023.0);
```

In order to print this value on our computer, we need to set up a serial communication protocol between the computer serial port and the Arduino by our USB cable. We can set up a serial communication at 9600 bits of data per second by the following command in the setup() function.

```
Serial.begin(9600);
```

Now we can print the voltage value on the serial monitor by the command

```
Serial.println(voltage);
```

Now, when you open your Serial Monitor in the Arduino development environment, you should see a steady stream of numbers ranging from $0.0 - 5.0$. As you turn the shaft of the Potentialmeter, the values will change, corresponding to the voltage coming into pin A0.

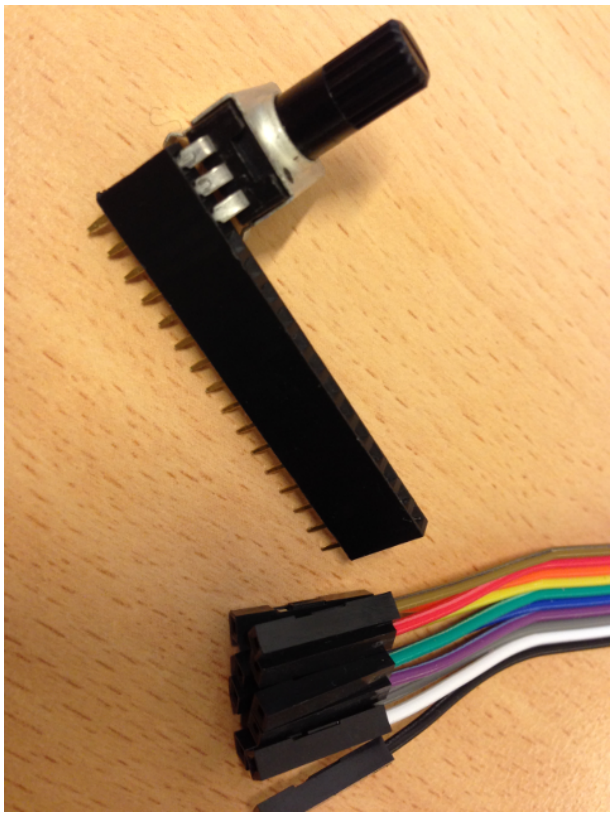


Figure 1.10: The potentiometer attached to a female header and female-to-female cables.

The complete codes is as follows:

```
/*  ReadAnalogVoltage
  Reads an analog input on pin 0, converts it to voltage,
  and prints the result to the serial monitor.
 */

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
```

Listing 1.2: Using the potentiometer with analogRead

Task 1.5 *Refer to Fig. 1.9, and proceed as follows:*

1. *Setup the circuit and run the program on Arduino.*
2. *Change the amount of the resistance by turning the shaft of the potentiometer.*
3. *Open the serial monitor to observe the print-out.*

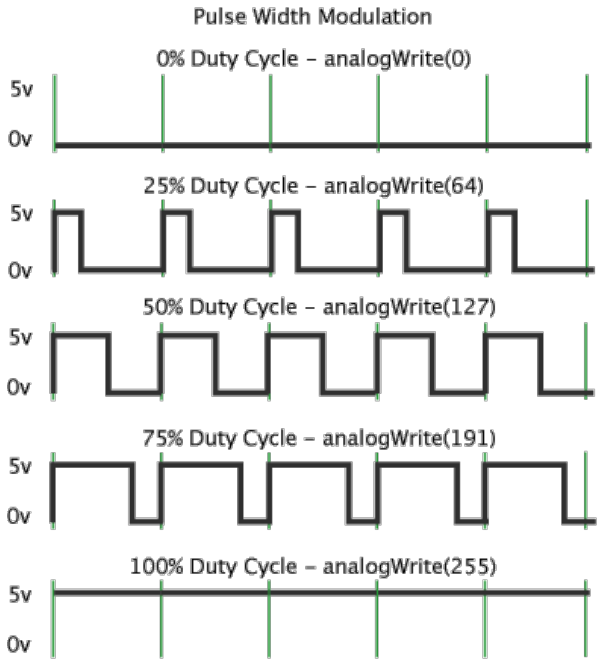


Figure 1.11: Pulse Width Modulation (PWM) waveform.

1.6 Control the Brightness of an LED: analogWrite

On the Arduino board, several pins are marked with $\sim$, those pins can generate Pulse Width Modulation (PWM) output signal. Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of $0 - 255$, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.

Now build your circuit similar as Fig. 1.5, but connect one of the LED pin to one of Arduino pins marked with $\sim$ (pin 3,5,6,9,10,11). With the following commands, one write an analog value 200 to drive a PWM output pin 3 and this value control the voltage applied on the corresponding LED which connected to this pin and thus control the brightness of the LED.

```
analogOutpin = 3; analogOutvalue = 200; analogWrite(analogOutpin, analogOutvalue);
```

Task 1.6 *Change the analogOutvalue of the above codes and see the different brightness of the controlled LED.*

Now, we want to use the potentialmeter to control the brightness of the LED. Connect your circuit as in Fig. 1.12

Task 1.7 *Write codes to control the brightness of the LED by the potentialmeter.*

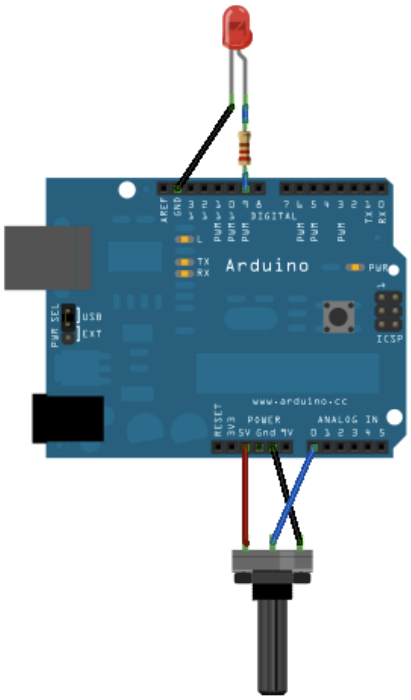


Figure 1.12: Using Potentialmeter to Control the Brightness of an LED.

1.7 Light Dependent Resistor (LDR)

An LDR or photo-cell (refer to Fig. 1.13) changes its resistance based on the light intensity. Bright light leads to a lowering of resistance (around 0.93 K Ohms) while covering the LDR to block light will lead to a substantial increase of resistance (around 17.7 K Ohms). You can find more information regarding LDR at: this link.

This experiment is reminiscent of the one with a potentiometer in Sec. 1.5. Look at the circuit in Fig 1.14. The objective is to turn the LED on as soon as the LDR detects that it has become dark. The LDR is read by analog-input.

```
int input_pin= A0;
int LED = 10;
int sensor_sample = 0;
```

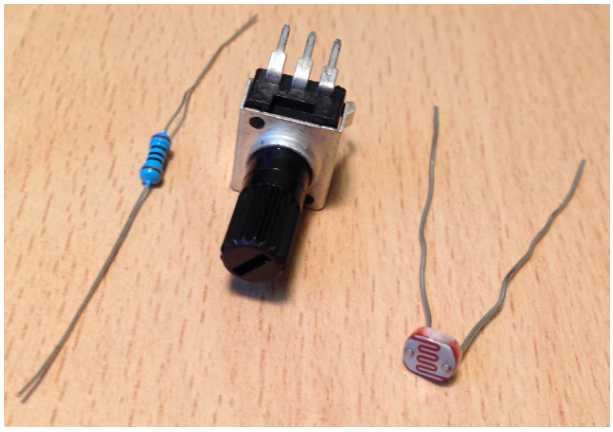
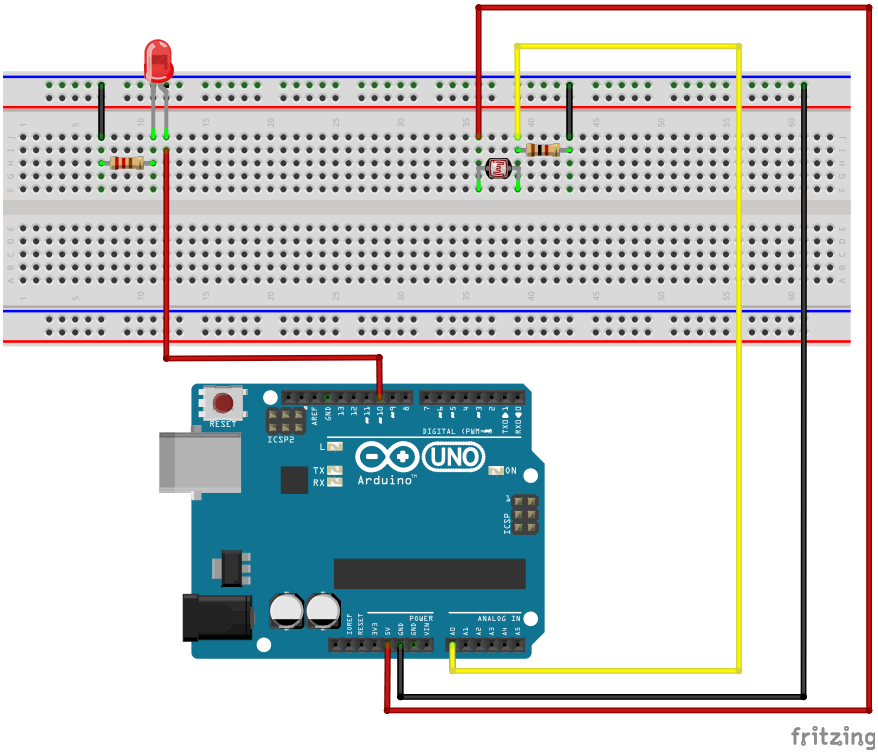
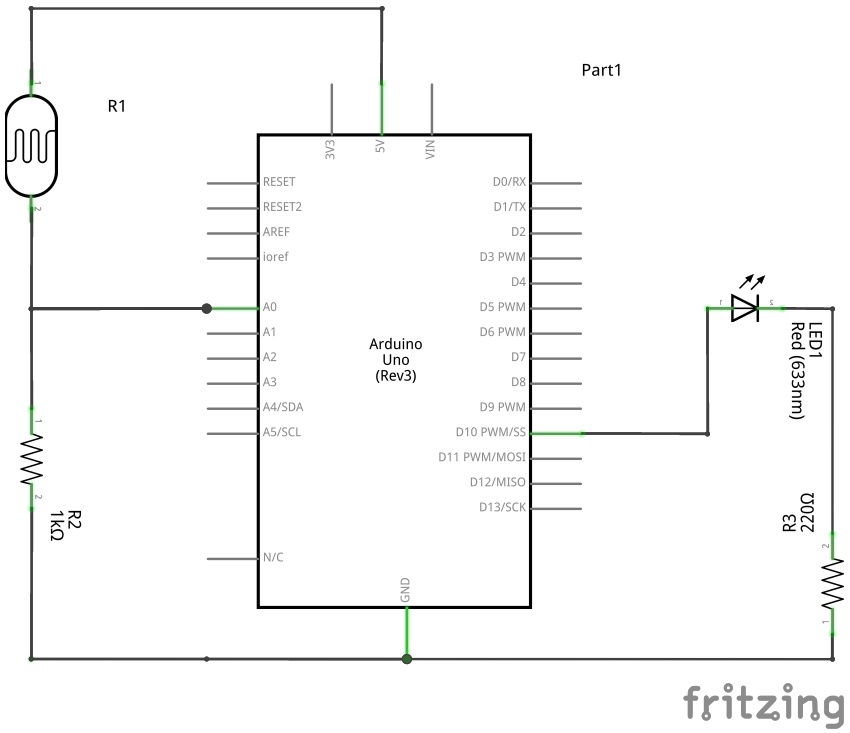


Figure 1.13: From left to right: A normal resistor, a potentiometer, and an LDR.

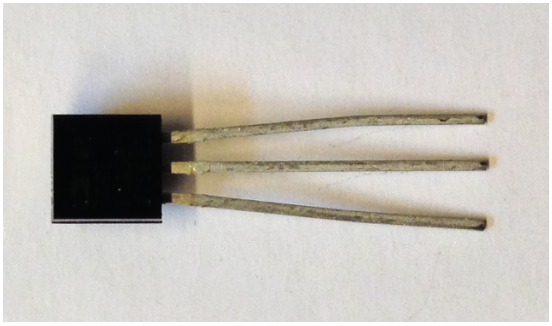


(a) Breadboard connections.



(b) Schematic.

Figure 1.14: LDR in a voltage divider



(a) Temperature Sensor

(b) Infra-Red Sensor

Figure 1.15: The temperature sensor and the IR sensor.

```
void setup(){
 Serial.begin(9600);
 pinMode (LED, OUTPUT);
}

void loop(){
  sensor_sample = analogRead(input_pin);
  Serial.print("sensor value = " );
  Serial.println(sensor_sample);

  if (sensor_sample < 650 ){
    digitalWrite(LED, HIGH);
  }else{
     digitalWrite(LED, LOW);
  }
  delay (50);
}
```

Listing 1.3: Using the potentiometer with analogRead

Task 1.8 *Set up the circuit as in Fig. 1.14 and upload the code Listing 1.3. Using the voltage divider rule, the voltage-measured by the analog-input A0 is* $5V \times R_2/(R_1 + R_2)$. *Recall that the ADC returns a value between* 0 *and* 1023 *for the input voltage range of* 0 *to* $5\,V$. *Depending on which value you selected for* R_2, *you may have to change the threshold* 650 *in Listing 1.3. Tweak its value such that the LED turns on when the LDR is covered.*

1.8 Temperature Sensor

Warning: The temperature-sensor (Fig. 1.15(a)) has three pins: it is important to keep the polarity in mind, else the sensor will burn-out. The polarity can be found as follows: View the sensor from its flat side (with some text visible), with its pins facing down. Then, the left pin should be connected to 5V, the right pin to GND, and the middle pin has the temperature-signal and hence should be connected to A0. The performance of the sensor will improve, if the Arduino is connected to an external power-supply (this is not the case in the labs) instead of taking power from the USB connection.

The middle-pin gives out a voltage between $0 - 2$ V. A value of 0 V corresponds to a temperature of $-50°$ C, and a value of 2 V corresponds to a temperature of 150° C. According to the manufacturer, the sensor is accurate to $\pm 2°$ C between $-40°$ to $+125°$ C.

Task 1.9 *Set up the circuit shown in Fig. 1.16 and use the code in Listing 1.4. Open the serial-monitor: the average-temperature (on a moving window of 10 samples) and its standard*

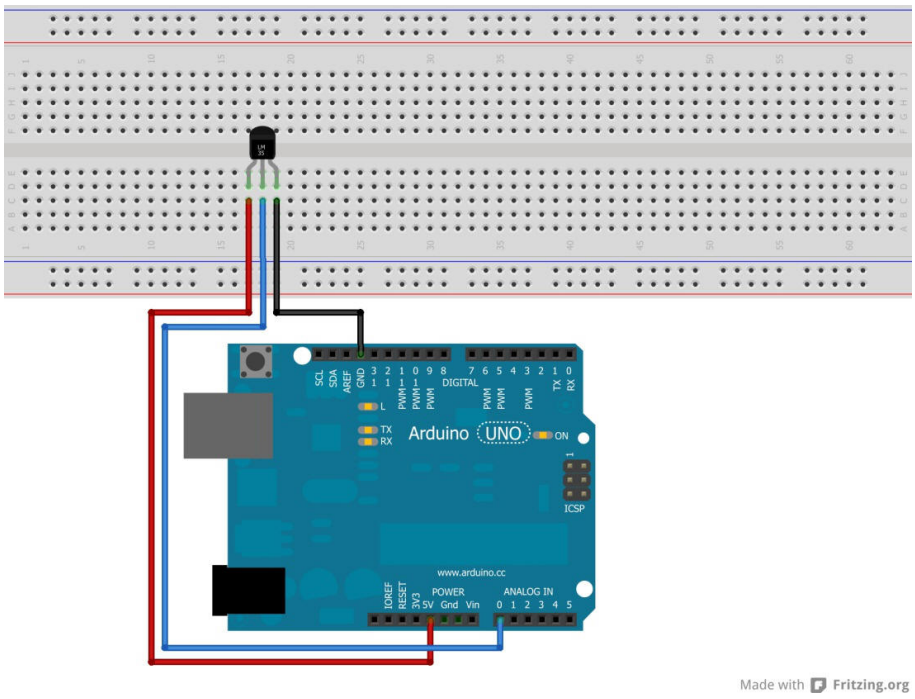
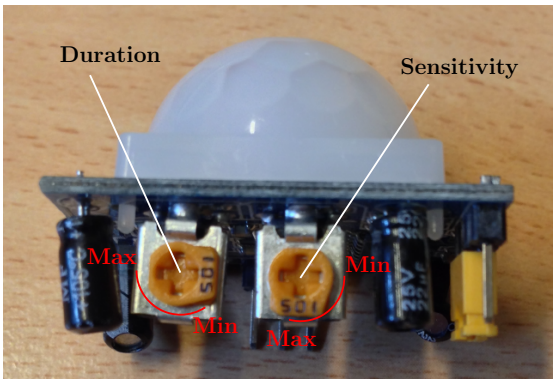


Figure 1.16: The circuit for measuring the temperature.

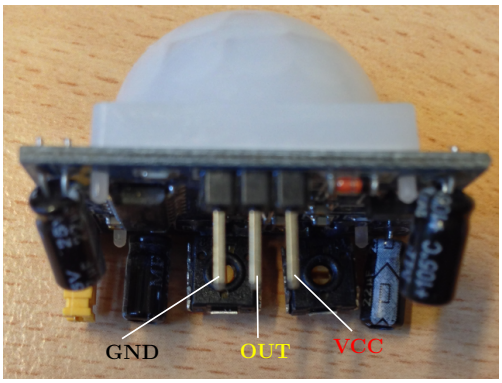
deviation is printed. Touch the sensor from the top (without touching its pins) and you will see the temperature change.

Task 1.10 *Why do we use the value* 410 *in the call to the* map *function in Listing 1.4?*

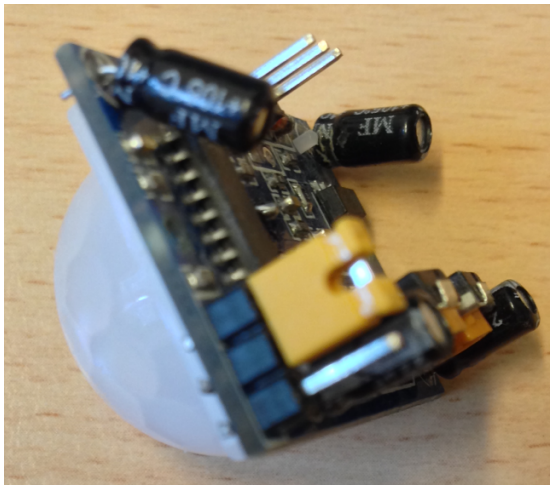
```
int TMP36= A0; // The middle lead is connected to analog-in 0.
int temperature= 0;

int wait_ms= 20; // wait time betweem measurements in millisec.

#define NR_SAMPLES 10
int samples[NR_SAMPLES]; // array of samples

void setup(){
  Serial.begin(9600);
}

void loop(){
  float sum= 0.0;
  for (int i=0; i< NR_SAMPLES; ++i){
    // map values from range [0, 410] to [-50, 150]
    samples[i]= map(analogRead(TMP36), 0, 410, -50, 150);
    sum += samples[i];
    delay(wait_ms);
  }
  float mean= sum/NR_SAMPLES;
  float sum_square_deviation= 0.0;
  for (int i=0; i< NR_SAMPLES; ++i){
    sum_square_deviation += (samples[i] - mean)*(samples[i] - mean);
  }
  float standard_deviation= sqrt(sum_square_deviation/NR_SAMPLES);
  Serial.print("mean: ");
  Serial.print(mean, 3);
  Serial.print(" C, \t std: ");
  Serial.println(standard_deviation);
```



(a) The knobs for tweaking the sensitivity (range) and duration of the output signal.



(b) You can also take off the plastic lens to see the markings for the pins.



(c) The orange jumper covering the inner two pins.

Figure 1.17: The PIR sensor HC-SR501.

```
}
```

Listing 1.4: Printing the Average Temperature and its Standard Deviation

1.9 Passive Infra-Red (PIR) Sensor for Motion Detection

The Passive Infra-Red (PIR) sensor (Fig. 1.17) detects IR light radiating from objects in its field-of-view (FOV) and uses this information to detect motion of humans, animals, and other objects. It is called passive because it does not emit any energy of its own for detection. When motion is detected, the output pin is set to 5V, which can be easily read by the microcontroller. More information about PIR sensor can be found here.

By shifting the jumper shown in Fig. 1.17(c), the sensor can be put into two modes:

1. If the jumper is as shown in Fig. 1.17(c), i.e. covering the inner two pins, the output signal remains active as long as a motion is detected. This mode is recommended for Arduino projects.

2. If the jumper is covering the outer two pins, the output signal is kept activated for a certain period and then deactivated – even if motion is still being detected. After a certain period of deactivation, if the motion is still being detected, the output is activated again.

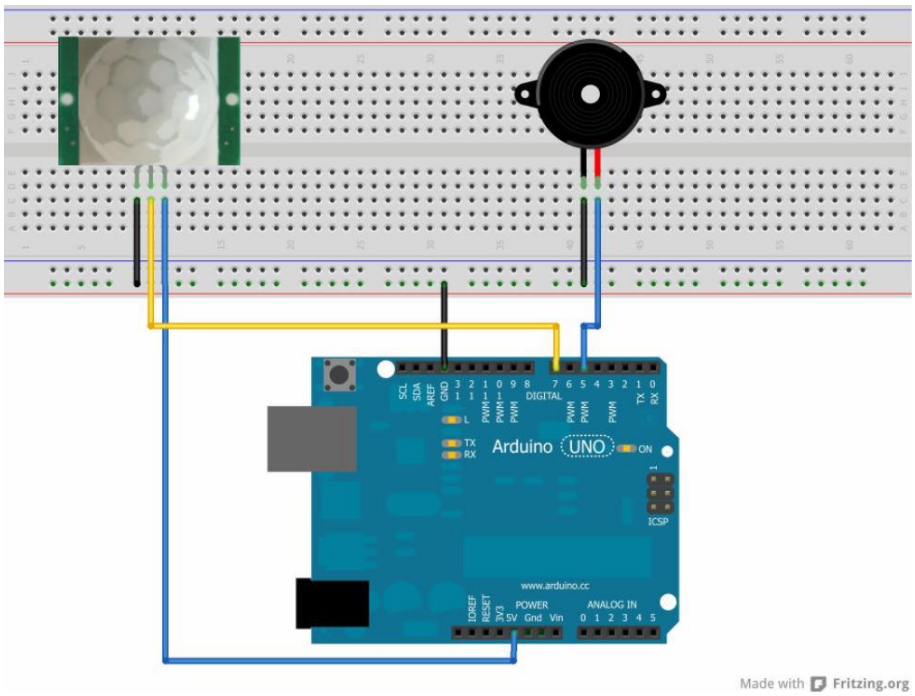


Figure 1.18: The piezo-speaker sounds when a motion is detected.

```
int piezo= 5;
int pir= 7;
int motion_status= 0;

void setup(){
  pinMode(piezo, OUTPUT);
  pinMode(pir, INPUT);
}

void loop(){
  motion_status= digitalRead(pir);
  if (motion_status == HIGH){
    digitalWrite(piezo, HIGH);
    delay(500);
    digitalWrite(piezo, LOW);
  }else{
    digitalWrite(piezo, LOW);
  }
}
```

Listing 1.5: The code for the PIR experiment.

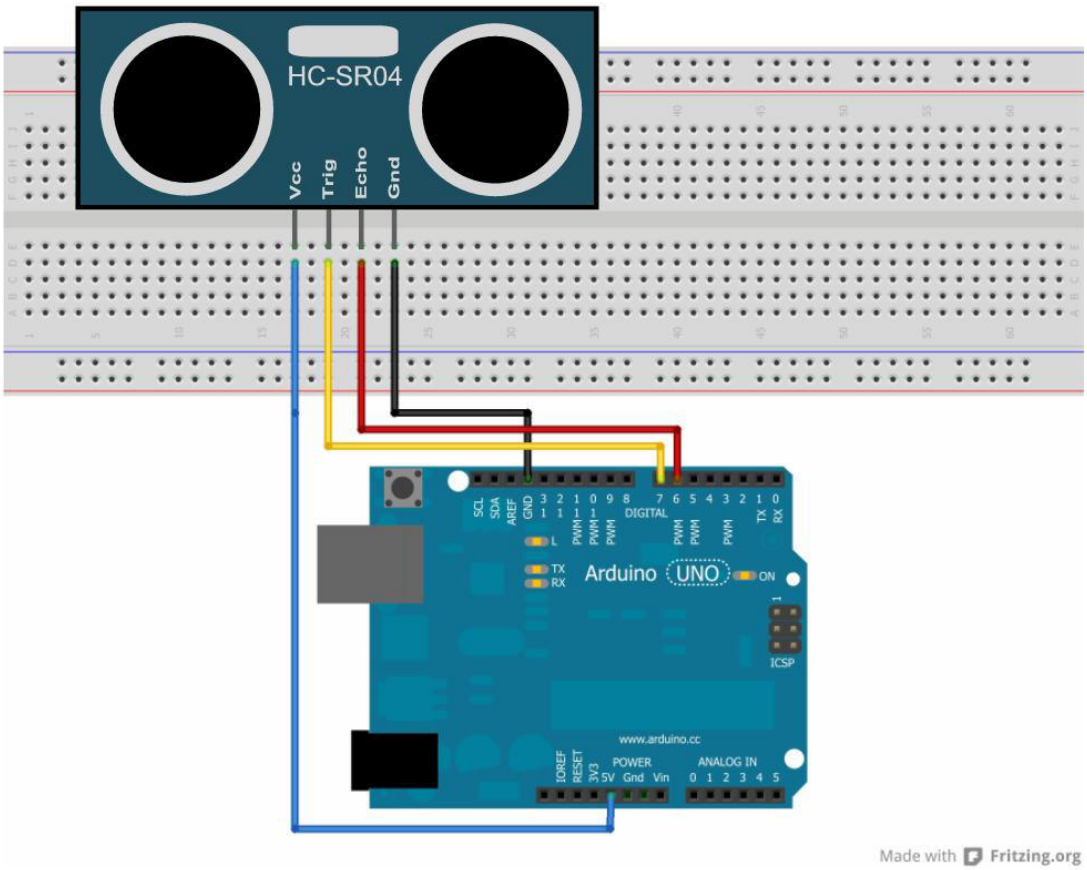
Task 1.11 *Set up the circuit as shown in Fig. 1.18 and upload the code shown in Listing 1.5. The piezo-speaker will sound as soon as it detects a motion.*

1.10 Ultrasonic distance sensor

Learn how to use the basic HC-SR04 distance sensor. The module has 2 ultrasonic transducers (speaker and microphone) and a controller that makes the measurement. It works by sending a series of sound waves (40kHz) and recording the time it takes for them to return after bouncing off an object. The controller then outputs this time interval encoded as the width of a voltage

pulse (that can be read by Arduino and interpreted as distance). More information about the ultrasonic distance sensor can be found here.

Now assemble the circuit:



Use the sample code:

```
const int trig = 7;
const int echo = 6;

void setup()
{
  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);

  digitalWrite(trig, LOW);

  Serial.begin(9600);
}

void loop()
{
  //send an impulse to trigger the sensor start the measurement
  digitalWrite(trig, HIGH);
  delayMicroseconds(15);  //minimum impulse width required by HC-SR4 sensor
  digitalWrite(trig, LOW);

  long duration = pulseIn(echo, HIGH);
        //this function waits for the pin to go HIGH,
        //starts timing, then waits for the pin to go LOW and stops timing.
        //Returns the length of the pulse in microseconds
```

```
  //'duration' is the time it takes sound from the transmitter back
        //to the receiver after it bounces off an obstacle
  const long vsound = 340;  // [m/s]
  long dist = (duration / 2) * vsound / 10000;  // 10000 is just the scaling
    factor to get the result in [cm]

  if (dist > 500 || dist < 2)
  {
    Serial.println("Invalid range!");
  }
  else
  {
    Serial.print(dist);
    Serial.println(" cm");
  }
  delay(1000);
}
```

sourcecodes/Ultrasound/ultrasound.ino

Task 1.12 *Display the distance in the serial monitor.*

1.11 Servomotor

A servomotor is an essential device for automated action of various mechanisms. Small servomotors are widely used for remote controlled airplanes, cars, robots and other toys while bigger servomotors are used for power steering in automobiles, control of valves in plants and many other systems.

Servomotors usually have a shaft that can precisely rotate at a specific angle, but linear servos also exist that have an output rod sliding back and forth. They are compact objects integrating an electric motor and a (fixed) gearbox, a sensing potentiometer on the output shaft and power/control electronics that adjust the motor power and direction such that the output shaft reaches the desired angle.

In this lab we are going to use a micro servo as shown in Fig. 1.19. In the actual servo given to you, the colors of the leads are slightly different. The brown lead is for the GND, red for 5V as usual, and orange for the control-signal (shown as yellow in Fig. 1.19). Extend the leads using the jumpers.

Warning: Do not restrict the rotation of the motor-shaft in any way: This may lead to high currents. Run the example code in Listing 1.11.

```
#include <Servo.h>
Servo s;  //create servo object;

void setup()
{
  s.attach(8);  //control signal on pin 8
}

void loop()
{
  //now we can control the servo with write(angle)
  //angle of the servo 0-180 degrees
  //90 degrees = servo is centerd

  s.write(0);
  delay(3000);

  s.write(45);
```

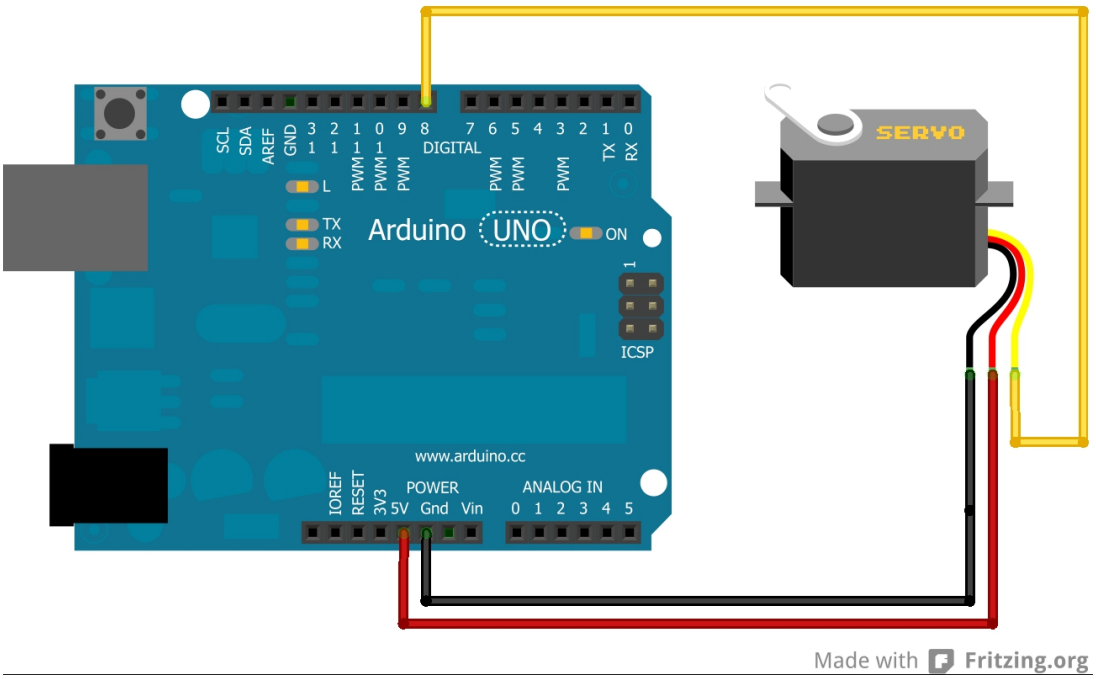


Figure 1.19: A micro-servo.

```
  delay(3000);

  s.write(90);
  delay(3000);

  s.write(135);
  delay(3000);

  s.write(180);
  delay(3000);
}
```

sourcecodes/Servo/servo.ino

Task 1.13 *Which two PWM pins cannot be used on the Uno if you're using the servo library? Refer to the library documentation.*

1.12 RGB LED

In this example, we are going to control the color of an RGB. We will use *Serial.parseInt()* function to locate values separated by a non-alphanumeric character. The values are parsed into integers and used to determine the color of a RGB LED. You'll use the Arduino Software (IDE) serial monitor to send strings like "5,220,70" to the board to change the light color.

Prepare the circuit as shown in Fig. 1.20. An RGB LED consists of three LEDs of red, green, and blue colors together sharing a common anode. The common anode is the second pin which is the longest. On one side of it is the red-LED's cathode, and on the other side of it are the cathodes of the green and blue LEDs. We will use a resistance of 200 Ω for all three LEDs, although you can choose them specifically for color based on the experiment in Lab 1.

The code is as follows:

```
/*
```

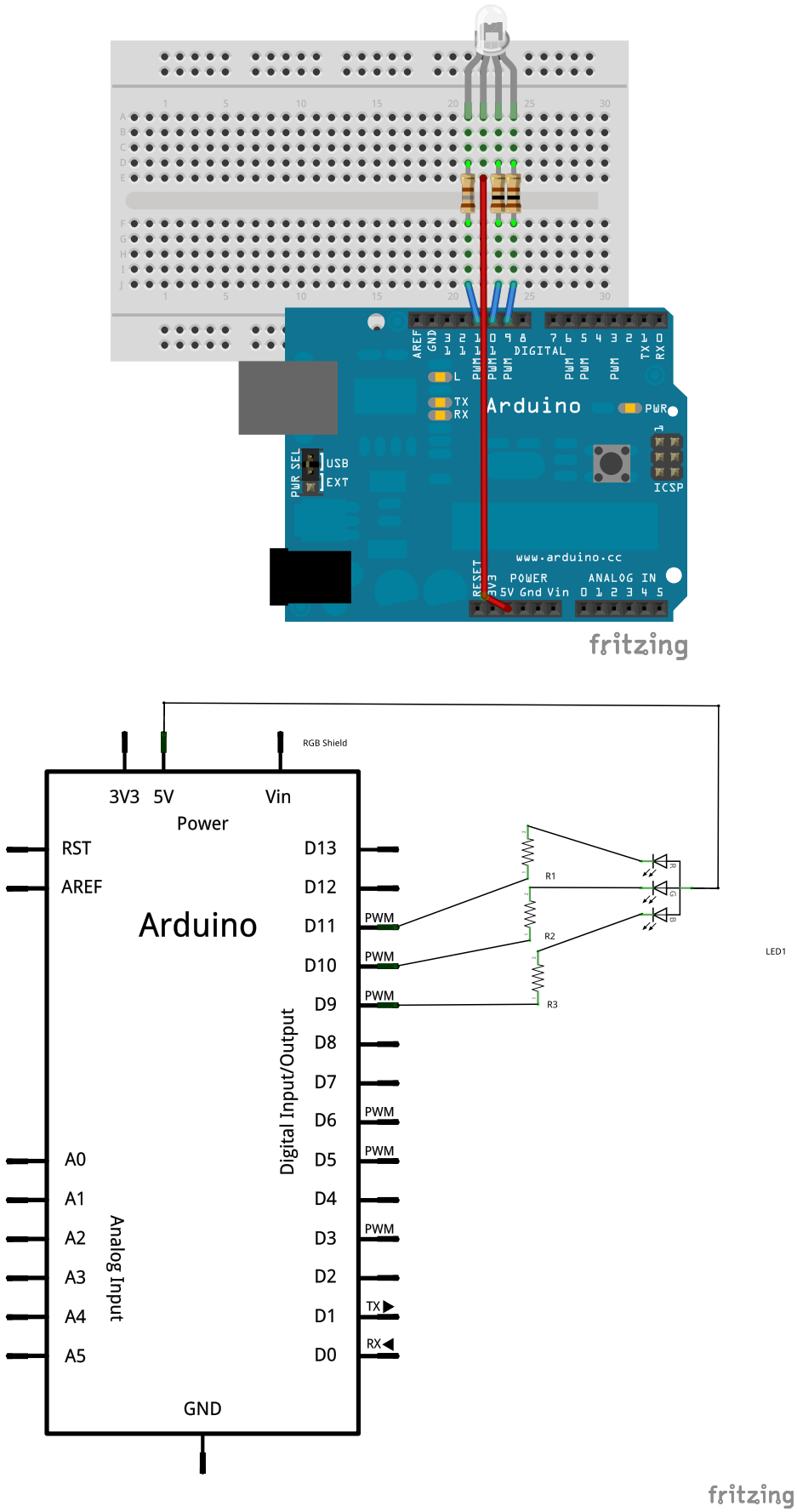


Figure 1.20: The circuit for a common-anode RGB LED.

```
  Reading a serial ASCII-encoded string.

 This sketch demonstrates the Serial parseInt() function.
 It looks for an ASCII string of comma-separated values.
 It parses them into ints, and uses those to fade an RGB LED.

 Circuit: Common-anode RGB LED wired like so:
 * Blue cathode: digital pin 9
 * Red cathode: digital pin 11
 * Green cathode: digital pin 10
 * anode: +5V

 This example code is in the public domain.
 */

// pins for the LEDs:
const int redPin = 11;
const int greenPin = 10;
const int bluePin = 9;

void setup() {
  // initialize serial:
  Serial.begin(9600);
  // make the pins outputs:
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

}

void loop() {
  // if there's any serial available, read it:
  while (Serial.available() > 0) {

    // look for the next valid integer in the incoming serial stream:
    int red = Serial.parseInt();
    // do it again:
    int green = Serial.parseInt();
    // do it again:
    int blue = Serial.parseInt();

    // look for the newline. That's the end of your
    // sentence:
    if (Serial.read() == '\n') {
      // constrain the values to 0 - 255 and invert
      // if you're using a common-cathode LED, just use "constrain(color, 0,
    255);"
      red = 255 - constrain(red, 0, 255);
      green = 255 - constrain(green, 0, 255);
      blue = 255 - constrain(blue, 0, 255);

      // fade the red, green, and blue legs of the LED:
      analogWrite(redPin, red);
      analogWrite(greenPin, green);
      analogWrite(bluePin, blue);

      // print the three numbers in one string as hexadecimal:
      Serial.print(red, HEX);
      Serial.print(green, HEX);
      Serial.println(blue, HEX);
    }
  }
}
```

Listing 1.6: Control the Color of an RGB LED.

Chapter 2

Visualize the Data by Processing Programming

In this lab, we are going to become familiar with the "Processing" programming language, its usage, and how we can make it communicate with Arduino.

2.1 Get Started with Processing

Processing is an open-source JAVA-like programming language mainly for visualizing data. Now please open the IDE of Processing and you will see a window similar in Fig. 2.1. It looks very similar to the Arduino IDE – this is because the Arduino IDE was originally inspired by Processing.

2.1.1 Your First Program

Note: If you want to know more details about a function, go to the online reference: https://processing.org/reference/

You're now running the Processing Development Environment (or PDE). There's not much to it; the large area is the Text Editor, and there's a row of buttons across the top; this is the toolbar. Below the editor is the Message Area, and below that is the Console. The Message Area is used for one line messages, and the Console is used for more technical details.

In the editor, type the following:

```
ellipse(50, 50, 80, 80);
```

This line of code means "draw an ellipse, with the center 50 pixels over from the left and 50 pixels down from the top, with a width and height of 80 pixels." Click the Run button, which looks like as shown in Fig. 2.2(a). If you've typed everything correctly, you'll see the display-window shown in Fig. 2.2(b).

If you didn't type it correctly, the message area will turn red and complain about an error. If this happens, make sure that you've copied the example code exactly: the numbers should be contained within parentheses and have commas between each of them, and the line should end with a semicolon. The code that you are typing is actually Java: if you know C/C++, it will not look too unfamiliar.

2.1.2 Draw

The computer screen (window) is composed of pixels. To create a new window, use the function *size(width, height)*. For example, `size(400,200)` creates a window with 400 pixels wide and 200 pixels high.

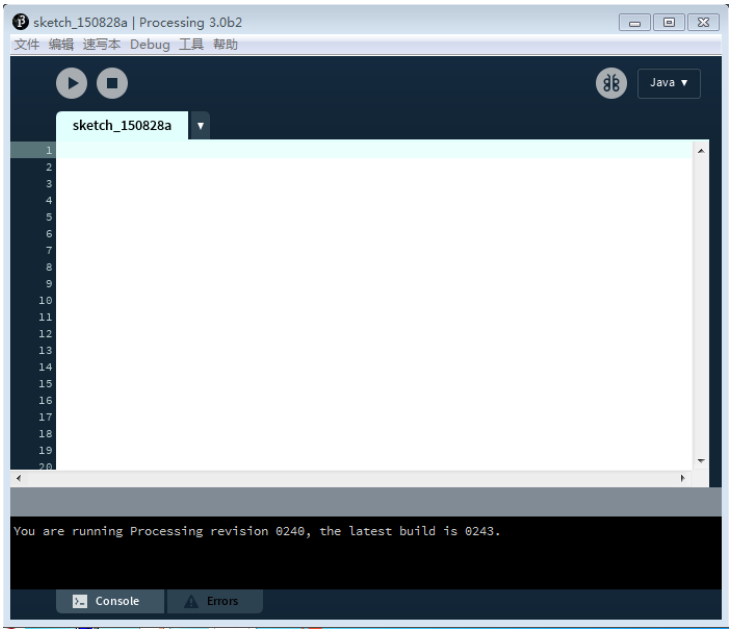
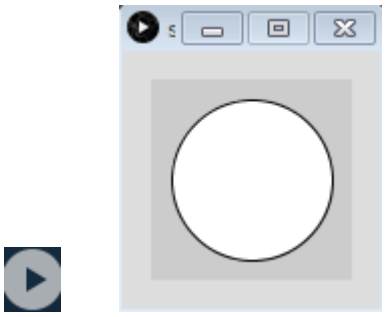


Figure 2.1: The Processing IDE Window.



(a) Run (b) A circle

Figure 2.2: Your first program.

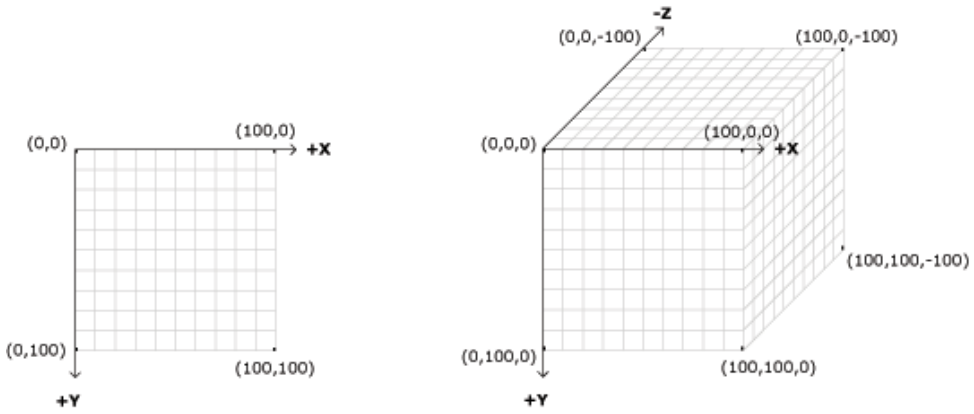


Figure 2.3: The Processing Coordinate-System.

Processing uses a Cartesian coordinate system with the origin in the upper-left corner. If your sketch is 320 pixels wide and 240 pixels high, coordinate $(0, 0)$ is the upper-left pixel and coordinate $(320, 240)$ is in the lower-right. The last visible pixel in the lower-right corner of the screen is at position $(319, 239)$ because pixels are drawn to the right and below the coordinate.

Draw a Point:

The following codes draw a point at coordinate $(250, 60)$. Try it your self and feel free to change the value inside the functions.

```
size(400,200); point(250,100);
```

Basic Shapes:

Processing includes a group of functions to draw different geometric graphs:

Syntax	Description
line$(x1, y1, x2, y2)$	$(x1, y1)$: start point; $(x2, y2)$: end point
triangle$(x1, y1, x2, y2, x3, y3)$	points of a triangle
quad$(x1, y1, x2, y2, x3, y3, x4, y4)$	points of a four sided polygon
rect$(x, y, width, height)$	(x, y) is the top-left point of a rectangle with the given sides.
ellipse$(x, y, width, height)$	(x, y) is the center of a circle
arc$(x, y, width, height, start, stop)$	*start* and *stop* are angles to start and stop an arc

More information can be found at here. Try to draw different shapes by yourself. For the angle parameters in the *arc* function, you can use *radians(180)* to denote the 180-degree angle.

The Order:

The computer runs the code line by line. If you want to put one object on top of others, you need to draw it after them.

Try the following codes and observe the difference.

Example: `size(400,200); ellipse(200,100,60,60); rect(50,50,200,50);`

Example: `size(400,200); rect(50,50,200,50); ellipse(200,100,60,60);`

Task 2.1 *Draw a 2D car.*

The Attributes of a Stroke:

Some most frequently used functions are listed below:

Syntax	Description
smooth()	smooth the line
strokeWeight(weight)	set the weight (in pixel) of a stroke
strokeJoin(join)	join ={MITER,BEVEL,ROUND}
strokeCap(cap)	cap ={SQUARE, PROJECT, ROUND}
rectMode(mode)	mode ={CORNER, CORNERS, CENTER, RADIUS}
ellipseMode(mode)	mode ={CORNER, CORNERS, CENTER, RADIUS}

Try the above functions to observe the different attributes of a stroke.

Example:

```
size(400,200); smooth(); strokeWeight(12); strokeJoin(ROUND); rect(50,50,150,150);
```

Color:

We can use *background(),fill()* and *stroke()* to change the color of a background, of an object, of a stroke.

For a black-white picture, only one value between $0 - 255$ is necessary. For example, 255 denotes white, 128 denotes Gray and 0 denotes black. For a color picture. One use three values to denote the combination of a RGB color.

Example:

```
size(400,200);
noStroke(); // no stroke
smooth(); //smooth
background(0,26,51); //set background to be dark blue
fill(255,0,0); // set color to be red
ellipse(132,50,150,150); // draw a circle
```

Listing 2.1: Color Example in Processing.

Task 2.2 *Improve your car from the last task by using different strokes and colors.*

2.2 Moving Object

In this section, we are going to learn how to use the mouse to interact with our drawing. In order to interact with the mouse, we need our code to run continuously. There is a function called *draw()* in Processing which runs repeatedly until you click the stop button. Every iteration of *draw()* constitutes one *frame*. The default frame rate is 60 frames per second.

Usually *draw()* is used together with an initialization function *setup()*. *setup()* only runs once in the beginning. We set up the frame rate inside *setup()* by function *frameRate(rate)*. Note how this is very similar to an Arduino sketch.

Run the following example and try to understand it.

```
float x = 60;
float y = 60;

int diameter = 60;

void setup() {
 size(480, 360);
 frameRate(30);
```

```
}

void draw()
{
 background(102);
 x = x+2.8;
 y = y+2.2;
 ellipse(x, y, diameter, diameter);
}
```

Listing 2.2: Moving Object Example in Processing.

In this example, x, y are global variables and can be seen inside both functions *setup()* and *draw()*.

2.3 Interact with the Mouse

In order to interact with the mouse, we need variables to store the coordinate of the current mouse position. In Processing, *mouseX* and *mouseY* store the coordinate of the current mouse position.

Try the following example.

```
void setup() {
 size(480, 360);
 smooth();
 noStroke();
 fill(102);
}

void draw()
{
 ellipse(mouseX, mouseY, 9, 9);
}
```

Listing 2.3: Following Mouse Example in Processing.

Add the function *background(204)* inside *draw()* and observe the difference.

Next, we use another example to see how to detect mouse pressed by a boolean variable *mousePressed*.

```
void setup() {
  size(480, 120);
}

void draw() {
  if (mousePressed) {
    fill(0);
  } else {
    fill(255);
  }
  ellipse(mouseX, mouseY, 80, 80);
}
```

Listing 2.4: Mouse Interaction Example in Processing.

This program creates a window that is 480 pixels wide and 120 pixels high, and then starts drawing white circles at the position of the mouse. When a mouse button is pressed, the circle color changes to black.

The next example will show you how to detect which buttons of the mouse is pressed.

```
void setup() {
  size(480, 120);
}

void draw() {
  if (mousePressed) {
    fill(0);
    if (mouseButton == LEFT)
       println("Left mouse pressed.");
    else
       println("Right mouse pressed.");
  }
  else {
    fill(255);
  }
  ellipse(mouseX, mouseY, 80, 80);
}
```

Listing 2.5: Mouse Detection Example in Processing.

More examples related to mouse and keyboard can be found at https://processing.org/reference/ − > Input

Task 2.3 *Now, with all you have learned, you are already able to program a small animation. Modify Listing 2.2 so that ball bounces off the "walls". The bouncing off angle should be realistic.*

2.4 More Examples

Try the following examples and try to understand the program:

Width Height
Map
Integer Float
Char String
DatatypeConversion
Image

2.5 Communication from Processing to Arduino

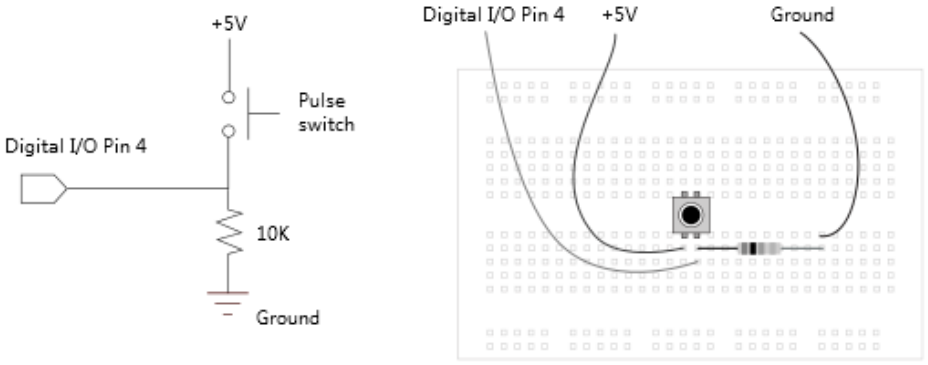
In this lab, we are going to connect Processing to Arduino and control an LED from Processing. In the first lab session, we already know that the data from the sensor is read in via USB cable from Arduino to the computer via serial communication. Arduino has a serial monitor to print out the value of the data. Processing also offers library to deal with data via serial communication. In order to use serial communication in Processing, we need to import a library in *setup()*:

```
import processing.serial.*;
```

2.5.1 Directly Using the Serial Communication

In the first example, we are going to read in a digital signal, i.e., the button (switch) status from Arduino and change the color of a rectangular in our Processing display window.

Please build up your circuit as the following diagram shows:



Read in the switch status via Arduino Pin4:

```
// Code for sensing a switch status and writing the value to the serial port.

int switchPin = 4;                        // Switch connected to pin 4
char temp = 0;
void setup() {
  pinMode(switchPin, INPUT);              // Set pin 0 as an input
  Serial.begin(9600);                     // Start serial communication at 9600
    bps
}

void loop() {
  if (digitalRead(switchPin) == HIGH) {  // If switch is ON,
    temp = 1;                 // send 1 to Processing
  } else {        // If the switch is not ON,
    temp = 0;               // send 0 to Processing
  }
  Serial.print(temp);     //send the value of temp to serial port
  delay(100);                             // Wait 100 milliseconds
}
```

Listing 2.6: Using serial communication in Processing to visualize a switch event via Arduino (The code on the Arduino Side)

The Processing program will create an object *port* from Serial class which is already defined in the imported library. In the *draw()* function, it keep checking whether there is new data available from the serial port.

The data will be stored in a variable called *val*. In this example, *val* is defined as an integer number and the switch status is either a HIGH voltage which corresponds to '1' or a LOW voltage (value '0'). Processing will display a grey rectangular if the switch status is HIGH or a black rectangular if the status is LOW.

```
// Read data from the serial port and change the color of a rectangle
// when a switch connected to the board is pressed and released

import processing.serial.*;

Serial port;                              // Create object from Serial class
int val;                                  // Data received from the serial port

void setup() {
  size(200, 200);
  // Open the port that the board is connected to and use the same speed (9600
    bps)
  port = new Serial(this, "COM3", 9600);
  // The port name may be different on your computer from "COM3",
```

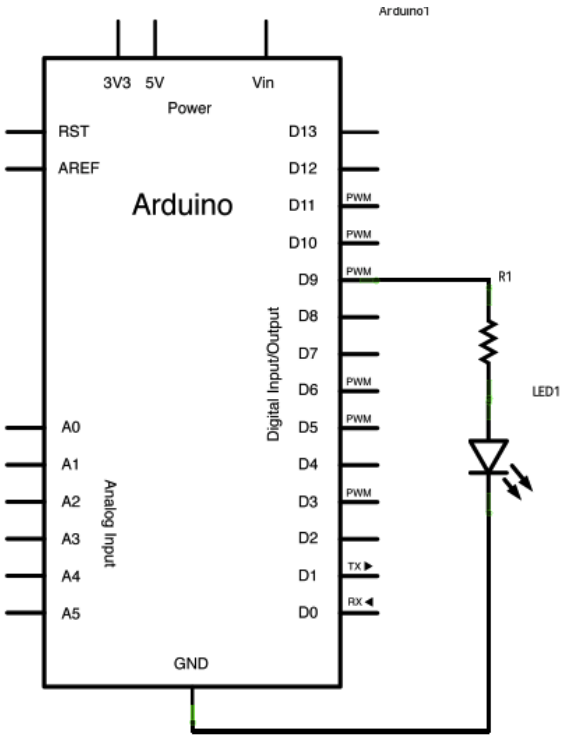
```
  // you can use println(Serial.list()); to check the port name
}

void draw() {
    if (0 < port.available()) {             // If data is available,
    val = port.read();                      // read it and store it in val
    println(val);                          // if the type of val is char, use port.
    readChar()
  }

  background(255);                          // Set background to white
  if (val == 0)  {                          // If the serial value val is 0
    fill(0);                                // set fill to black
  }
  else {                                   // If the serial value is not 0,
    fill(204);                              // set fill to light gray
  }
  rect(50, 50, 100, 100);
}
```

Listing 2.7: Using serial communication in Processing to visualize a switch event via Arduino (The code on the Processing side)

Task 2.4 *Set up the circuit and successfully run the program.*

In the second example, we are going to learn how to use the X coordinate of the mouse to control the brightness of an LED. In Processing display window, we can visualize the brightness of the LED depend on the X coordinate of the mouse.

Please set up the circuit as the diagram shows:



Arduino reads data from the serial port and stores the data into a variable called *brightness* (ranging from 0 to 255) and uses them to set the brightness of the LED.

```
const int ledPin = 9;      // the pin that the LED is attached to

void setup()
```

```
{
  // initialize the serial communication:
  Serial.begin(9600);
  // initialize the ledPin as an output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  int brightness;

  // check if data has been sent from the computer:
  if (Serial.available()) {
    // read the most recent byte (which will be from 0 to 255):
    brightness = Serial.read();
    // set the brightness of the LED:
    analogWrite(ledPin, brightness);
  }
}
```

Listing 2.8: Using serial communication in Processing to control and visualize the brightness of an LED via Arduino (The code on the Arduino Side)

The data is the X coordinate of the mouse sent by Processing.

```
import processing.serial.*;

Serial port;                              // Create object from Serial class
int val;                                  // Data received from the serial port

void setup() {
  size(200, 200);
// Open the port that the board is connected to and use the same speed (9600 bps
    )
  port = new Serial(this, "COM3", 9600);
  // The port name may be different on your computer from "COM3",
  // you can use println(Arduino.list()); to check the port name
}

 void draw() {
 // draw a gradient from black to white
 for (int i = 0; i < 256; i++) {
 stroke(i);
 line(i, 0, i, 150);
 }

 // write the current X-position of the mouse to the serial port as
 // a single byte
 port.write(mouseX);
 }
```

Listing 2.9: Using serial communication in Processing to control and visualize the brightness of an LED via Arduino (The code on the Processing Side)

Task 2.5 *Understand the code and successfully control the brightness of an LED by the mouse.*

2.5.2 More on the Serial Communication

You might already notice that Arduino use "Serial.print" or "Serial.println" to send data through serial port. It will send the data as a String of characters, each character has one byte. The following example tries to send an integer from Arduino to Processing.

```
int incomingByte = 23;   // for incoming serial data
```

```
void setup() {
        Serial.begin(9600);     // opens serial port, sets data rate to 9600 bps
        Serial.println(incomingByte);
}

void loop() {

 }
```

Listing 2.10: Send an Integer from Arduino to Processing (The code on the Arduino Side)

On the Processing side, "port.read()" will read the incoming data from the serial port one byte each time.

```
import processing.serial.*;

Serial port;                                // Create object from Serial class
int val;                                    // Data received from the serial port

void setup() {
  size(200, 200);
  // Open the port that the board is connected to and use the same speed (9600
    bps)
  port = new Serial(this, "COM3", 9600);
  // The port name may be different on your computer from "COM3",
  // you can use println(Serial.list()); to check the port name
}

void draw() {
    if (0 < port.available()) {             // If data is available,
    val = port.read();                      // read one byte from the serial port
    and store it in val
    println(val);
  }
}
```

Listing 2.11: Receive a Byte from Arduino by Processing (The code on the Processing Side)

After running the code, you will notice that the Arduino send the integer 23, but the Processing received four numbers 50, 51, 13, 10. The reason is that "Serial.println" treats the integer "23" as a String of characters '2' and '3'. If you check the ASCII table http://www.asciitable.com/, you will find decimal representation of characters '2' and '3' are 50 and 51. "Serial.println" will add a carriage return '\r' (13) and a line feed '\n' (10) at the end of the string. In order to correctly receive the integer, you need to change the data type of "incomingByte" from "int" to "char" and use "Serial.print". This will allow the Arudino program treat the data "incomingByte" as one byte. That is the case in Example 2.6.

Another way to correctly receive an integer is to keep the code on Arduino side unchanged as in 2.10 but change on the Processing side. Instead of using "port.read()" which only receive one byte at one time, we use "port.readStringUntil('\n')" which will receive the data as a string of character until it reach the line feed '\n'. And then use functions "float()" and "int()" to convert a string into a float and integer. The modified Processing code is as follow: On the Processing side, "port.read()" will read the incoming data from the serial port one byte each time.

```
import processing.serial.*;

Serial port;                                // Create object from Serial class
String valstring;
int val;                                    // Data received from the serial port
```

```
void setup() {
  size(200, 200);
  // Open the port that the board is connected to and use the same speed (9600
    bps)
  port = new Serial(this, "COM3", 9600);
  // The port name may be different on your computer from "COM3",
  // you can use println(Serial.list()); to check the port name
}

void draw() {
    if (0 < port.available()) {             // If data is available,
    valstring = port.readStringUntil('\n');  // read the input until '\n' and
    store it as a string
    val =int(float(valstring));          // convert the string into an integer
    println(val);                       //
  }
}
```

Listing 2.12: Receive a String from Arduino by Processing (The code on the Processing Side)

Task 2.6 *Correctly receive the integer* 23 *on the Processing side.*

2.5.3 An Example – Temperature Visualization

Please set up the circuit as in Fig. 1.16. The following code is on the Arduino side to collect temperature data and send it to the serial port.

```
int temperature;

void setup(){
  Serial.begin(9600);
}

void loop(){

    temperature= map(analogRead(A0), 0, 410, -50, 150);
    Serial.println(temperature);

}
```

Listing 2.13: Collect Temperature Valud and Send it via Serial Port (The code on the Arduino Side)

On the Processing side, temperature value will be read in as a String from the serial port and then converted into float. A visualization codes will be provided for you to display the temperature in graph.

```
import processing.serial.*;

Serial port;
String output="0.00";
float temp;

void setup() {
 size(330,306);
 port = new Serial(this, "COM8", 9600);
}

void draw() {
  int i;
  int ypos;
```

```
    while (port.available()>0) {
        output = port.readStringUntil('\n');
        if (output != null) {
          temp = float(output);
          println(temp);
        }
    }

    ypos = int((100 - temp) / 100 * 256);
    background(32);
    for(i=0; i<256; i++) {
      stroke(255 - i, 128, i);
      line(25, 25+i, 175, 25+i);
    }

    stroke(255);
    for ( i = 0 ; i < 11 ; i++) {
    int ypos2 = round (25 + 25.6 * i ) ;
    line (173 , ypos2 , 177 , ypos2 ) ;
    textSize (12) ;
    text ( ( ( 10 -i ) * 10) , 178 , ypos2 + 6) ;
    }

    stroke(255, 128);
    line(20,25+ypos, 205, 25+ypos);

    textSize(32);
    text(temp, 205, ypos+38);
}
```

Listing 2.14: Receive Temperature Value and Visualize it in Processing (The code on the Processing Side)

2.5.4 Send Multiple Data from Arduino to Processing

Now, you might wish to send multiple data from Arduino to Processing at one time. In the next example, you will learn how to receive multiple data from Arduino by Processing. The idea is to send multiple data as a string and separated by some certain characters such as ':' from Arduino. Processing will receive the string until '\n' and split the data by ':'.

On the Arduino Side:

```
int a=0;
void setup() {
  // initialize serial:
  Serial.begin(9600);
}

void loop() {
  // if there's any serial available, read it:
  //while (Serial.available() > 0) {
   while (a < 10) {
    // look for the next valid integer in the incoming serial stream:
      a = a+1;

      Serial.print("The value is:");
      Serial.print(a);
      Serial.println();
    }
 // }
}
```

Listing 2.15: Demonstrate How to Process a String (The code on the Arduino Side)

On the Processing side:

```
import processing.serial.*;

int strtoint;
float strtofloat;
String myString = null;
Serial myPort;  // The serial port

void setup() {
  // List all the available serial ports
  //printArray(Serial.list());
  // Open the port you are using at the rate you want:
  myPort = new Serial(this, "COM7", 9600);
  myPort.clear();
  // Throw out the first reading, in case we started reading
  // in the middle of a string from the sender.
  myString = myPort.readStringUntil('\n');
  myString = null;
}

void draw() {
  while (myPort.available() > 0) {
    myString = myPort.readStringUntil('\n');
    if (myString != null){  // if received string is non empty
    String[] myStringList = split(myString, ':'); // split the String into two
    substrings separated by ':'
    if (myStringList.length == 2) { //  if there are 2 Strings in myStringList
      println(myString);
      print(myStringList[0]);
      println(myStringList[1]);
      strtofloat = float(myStringList[1]);  // convert string to float
      strtoint = int(strtofloat);           // convert float to int
      println(strtofloat);
      println(strtoint);
    }
    }
  }
}
```

Listing 2.16: Demonstrate How to Process a String (The code on the Processing Side)

More information on how to process string can be found at website https://processing.org/reference/String.html.

Task 2.7 *Understand the codes, explain the codes to the instructor/TAs.*

2.5.5 Send Multiple Data from Processing to Arduino

In this section, we will learn how to send multiple data from Processing to Arduino. The following code will send three integers from Processing to Arduino. These three integers are R,G,B pixel values of your mouse clicking location on an image file "color-wheel.jpg". The Processing will send these three values as a string separated by comma and end with '\n'.

```
import processing.serial.*;

Serial myPort;

// Declaring a variable of type PImage
PImage img;

void setup(){
  //println(Serial.list());
```

```
  myPort = new Serial(this, "COM7", 9600);
  // If the LED color does not change, the Arduino port is wrong.

   size(800 , 800) ;
  // The image file should exist in the data subfolder of the sketch folder
  img= loadImage("color-wheel.jpg");
  background(0);
  // Draw the image to the screen at coordinate (0,0)
  image(img, 0, 0);
  loadPixels(); // tell Processing, we want to access pixel values
}

void draw(){
  if (mousePressed == true) { // if the mouse is pressed
    // Get the color at mouseX, mouseY
    // The pixels are stored in row-major format
    int loc= mouseY*width + mouseX;
    int r = int( red(pixels[loc]) ); // get the pixel value of red and convert
   float to int
    int g = int( green(pixels[loc]) );
    int b = int( blue(pixels[loc]) );

  myPort.write(str(r));  // send r as a string
  myPort.write(',');
  myPort.write(str(g));
  myPort.write(',');
  myPort.write(str(b));
  myPort.write('\n');   // send line feed
  }
}
```

Listing 2.17: Control An RGB LED by Processing (The code on the Processing Side)

Connect the circuit as showed in Fig. 1.20. On the Arduino side, you will receive the string from Processing through the Serial port. You can use the List 1.6 to parse the the receiving string back into the three integers and turn on the RGB LED accordingly.

Next, I would like introduce another more general way to receive a string by Arduino which allow the Arduino receive a mixture of Char-type data, String-type data and integer-type data. The idea is to first split the string into substrings (Char-type data is a string with length 1) and then convert the string into integer correspondingly. The following is the code on the Arduino side.

```
/*
  Reading a RGB value from Processing throug the serial port.

  and uses RGB data to fade an RGB LED.

 Circuit: Common-anode RGB LED wired like so:
 * Blue cathode: digital pin 11
 * Red cathode: digital pin 10
 * Green cathode: digital pin 9
 * anode: +5V

 created 17 June 2016
 by Fangning Hu

 This example code is in the public domain.
 */
#include <string.h>

// pins for the RGB LEDs:
const int greenPin = 10;
const int redPin = 11;
```

```
const int bluePin = 9;

void setup() {
  // initialize serial:
  Serial.begin(9600);
  // make the pins outputs:
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

}

void loop() {
    char *first,*second, *third;
    int data1,data2,data3;
    int red, green, blue;
    String serialResponse;
    int len;

    if ( Serial.available()) { // if there's any serial available, read it:
    serialResponse = Serial.readStringUntil('\n'); //read data into a String
    object serialRespons
    //Serial.println(serialResponse);
    len = serialResponse.length();           // get the length of the String
    char buf[len+1];
    serialResponse.toCharArray(buf, sizeof(buf)); // convert a String object to
    char *buf
    //  split buf by comma
    first = strtok(buf,",");   //get the first CharArray splitted by comma
    data1 = atoi(first);       // convert CharArray to integer
    Serial.println(data1);

    //  Second strtok iteration
    second = strtok(NULL,",");   //get the second CharArray splitted by comma
    data2 = atoi(second);
    Serial.println(data2);

    third = strtok(NULL,",");   //get the third CharArray splitted by comma
    data3 = atoi(third);
    Serial.println(data3);


      // constrain the values to 0 - 255 and invert
      // if you're using a common-cathode LED, just use "constrain(color, 0,
    255);"
      red = 255 - constrain(data1, 0, 255);
      green = 255- constrain(data2, 0, 255);
      blue = 255 - constrain(data3, 0, 255);

      // fade the red, green, and blue legs of the LED:
      analogWrite(redPin, red);
      analogWrite(greenPin, green);
      analogWrite(bluePin, blue);

      // print the three numbers in one string as hexadecimal:
      Serial.print(red, HEX);
      Serial.print(green, HEX);
      Serial.println(blue, HEX);

  }
}
```

Listing 2.18: Control An RGB LED by Processing (The code on the Arduino Side)

Task 2.8 *Control the RGB LED by Processing.*

Chapter 3

Wireless Sensor Network

In this lab, we are going to set up networks allow wireless communication among sensors/actuators on different devices. First, we will learn how to make Arduino communicate with the computer via Bluetooth. Then we will set up a wireless sensor network based on ZigBee wireless networking protocol and xBee radio modules.

3.1 Bluethooth Connection

Before starting the Bluethooth connection, please first try the following example designed by one of our students:

```
// Code by Anuraag Shakya
char value ; // Value sent over serrial port
char lastValue = 'f'; // Stores last state of LED
int ledPin = 13 ; // Output LED pin

void setup ( )
{
   Serial.begin (9600) ; // Baud rate set to 9600
   pinMode ( ledPin ,OUTPUT) ; // Set l edPin to output
}
void loop ( )
{
   if ( Serial.available( ) ) // if there is data being recieved
  {
    value = Serial.read ( ) ; // Store read value to value
    //Serial.write ( value ) ;
  }

   if ( value == 'n' && lastValue != 'n' ) // if value from serialis'n ' and
    current state is not 'n '
  {
      digitalWrite (13 , HIGH) ; // switch on LED
      Serial.println ( "LED is on"  ) ; // print LED is on for debugging
      lastValue=value ; // save current state of LED
  }
  else if (value=='f' && lastValue != 'f')
  {
       digitalWrite(13, LOW);
       Serial.println("LED is off");
       lastValue = value;
  }

  else if (value =='b')
  {
    if (lastValue !='b') {
```

```
        Serial.println("Blinking LED");
        lastValue = value;
    }
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(500);
  }
  delay(500);
}
```

Listing 3.1: Control the State of an LED

Connect an LED to pin 13 and open the serial monitor. Based on the input in your serial monitor, the LED will change its state. Input 'n' will turn on the LED; input 'f' will turn off the LED; input 'b' will blink the LED once.

Task 3.1 *Correctly run the program.*

You will notice that the computer can send/receive data from the serial monitor through a USB cable which uses serial port named "COM3" on my computer (Your connection might have different port name). Instead of using a USB cable, we can attach a Bluethooth module to the Arduino and attach a Bluetooth adapter to the computer to set up a Bluetooth connection between the Arduino and computer. The data exchange between the Bluetooth module and Arduino is through TX and RX pins based on the UART (Universal Asynchronous Receiver/Transmitter). Now, please connect the Bluetooth module to your Arduino as in Table 3.1.

Bluethooth	Arduino
VCC	5V
GND	GND
TX	RX (pin 0)
RX	TX (pin 1)

Table 3.1: Pin Connection Between Bluetooth Module and Arduino.

After the connection, a red LED on the Bluetooth module will start to blink.

Plug a Bluetooth adapter to the computer to allow the computer to communicate in Bluetooth. The computer may automatically install the driver for Bluetooth adapter. It may take some time. After the driver is installed, you will find a Bluetooth icon at the right corner of your computer. Right click the Bluetooth icon and select "Add Device", you should be able to find the Bluetooth module named "HC-05" or "HC-06". Click "Next" and select "Enter the Device's Pairing Code", by default the pairing code is "1234". Enter "1234" and click "Next", the Bluetooth module will be successfully installed on your computer. Right click the Bluetooth icon and select "Show Bluetooth Devices", you should see a device icon "HC-05" or "HC-06". Right click the icon and select "Property", then "Hardware", you will find out on which serial port the Bluetooth module is connected to your computer. In my case, it is "COM10". Now in your Arduino IDE, select "Tool-¿Port", you will find a new port "COM10" listed there. Now select this new port, open the serial monitor again. The input in the serial monitor will be send to Arduino through this new port "COM10", namely through the Bluetooth channel over air instead of the USB cable. Try to input 'n' to see whether it will turn on the LED. Now use a battery or power adapter to provide power to your Arduino and disconnect your USB cable from the Arduino, you should still be able to control the state of the LED by inputting in the serial monitor on "COM10".

Serial.print will send data through Arudino TX pin (pin 1) and Serial.read will read in data through Arduino RX pin (pin 0) by default. Bluetooth module will receive data through its RX pin from Arduino TX pin and send the data to its pairing device (the computer) over the air. The computer also send data wirelessly to its pairing device (the Bluetooth module) and the Bluetooth module will forward the data through its TX pin to the Arduino RX pin.

Task 3.2 *Use the serial monitor to control the LED via Bluetooth.*

3.2 SoftwareSerial

However, when we use the Arudino pin 0 and pin 1 to do the serial communication, we can not upload codes through USB cable to Arduino at the same time. Only after you disconnect the Arduino pin 0 and pin 1 from the Bluetooth module, you can start to upload codes through the USB cable. In order to avoid this trouble, we would like to use other Arduino pins to do the serial communication which requires a library called "SoftwareSerial.h".

Now add the following code into your previous program. This will define a new serial communication at the baud rate of 9600 named "bluetoothSerial" with pin 6 as its RX pin and pin 7 as its TX pin on Arduino.

```
#include <SoftwareSerial.h>

short bluetoothTx = 7;   //use pin 7 as Arduino TX pin
short bluetoothRx = 6;   //use pin 6 as Arduino RX pin

SoftwareSerial bluetoothSerial(bluetoothRx, bluetoothTx); //define a Serial
    Communication

void setup ( )
{
   bluetoothSerial.begin(9600);    // Baud rate set to 9600
}
```

Listing 3.2: Set Up a New Serial Communication by SoftwareSerial Library.

Connect the Bluetooth module TX pin to Arduino pin 6 (the new defined RX pin for Arduino to receive data from the Bluetooth module). Connect the Bluetooth module RX pin to Arduino pin 7 (the new defined TX pin for Arduino to send data to the Bluetooth module). Now the data exchanging between Bluetooth module and Arduino will be handled by Arduino pin 6 and pin 7. You need to use bluetoothSerial.read and bluetoothSerial.print to read data through Arduino pin 6 and send data through Arduino pin 7, respectively. Use bluetoothSerial.available to check the availability of the incoming data.

In Arduino IDE, switch the port to the USB port ("COM3" in my case) and upload your new code to Arduino. Then switch the port back to the Bluetooth port ("COM10") and use the serial monitor to control the LED state.

HINT: Note that Serial.print and Serial.read will exchange data through your USB channel. bluetoothSerial.read and bluetoothSerial.print will exchange data through the Bluetooth channel. Your computer use different ports (different "COM") to send/receive data over different channels.

Task 3.3 *Understand the meaning of different port and the usage of SoftwareSerial library. Use the serial monitor to control the LED via Bluetooth.*

3.3 xBee Modules

xBee modules are designed for data communication over-the-air. Compared to Bluetooth, it can transmit data for longer distance. Fig. 3.1 shows an xBee module and its pin layout.

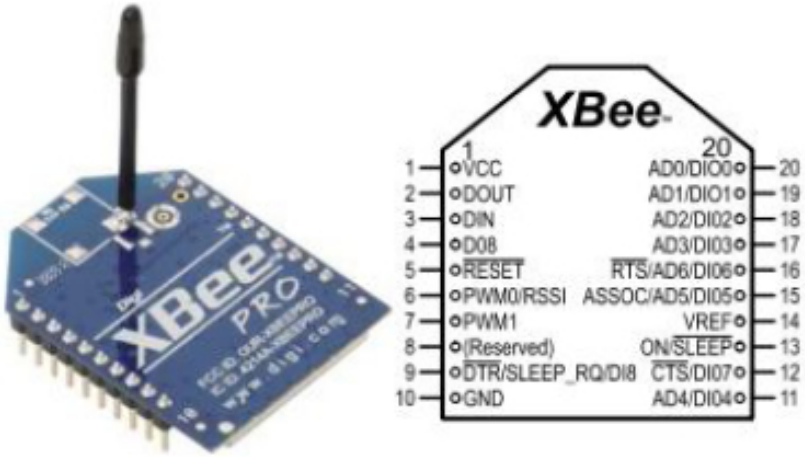


Figure 3.1: An xBee Module and its Pin Layout.

Similar to the Bluetooth module, xBee module also use UART to communicate with Arduino. If we use Arduino pin 0 and pin 1 as its RX and TX, xBee module should connect to Arduino as in Table 3.2.

xBee	Arduino/xBee Shield
VCC (pin 1)	5V
GND (pin 10)	GND
DOUT (pin 2)	RX (pin 0)
DIN (pin 3)	TX (pin 1)

Table 3.2: Pin Connection Between xBee Module and Arduino/xBee Shield

However, the pins on xBee module are too thick and they can not be plugged into the breadboard. We need to use an xBee shied (Fig. 3.2) to serve as an interface.

Plug the xBee Module to the xBee shield. On the shield, the xBee module connects to the xBee shield internally according to Table 3.2.

If we use Arduino pin 11 and pin 12 as its TX and RX, then we can achieve UART data communication between Arduino and xBee module by a connection as showed in Table 3.3.

xBee Shield	Arduino
5V	5V
GND	GND
RX (pin 0)	pin 12 (SoftwareSerial pin RX)
TX (pin 1)	pin 11 (SoftwareSerial pin TX)

Table 3.3: Pin Connection Between the xBee Shield and Arduino

3.3.1 Wireless Communication Between Two Arduinos

Now connect two xBee shields with two Arduinos as showed in Table 3.3. Then these two Arduino can exchange data through xBee modules over the air if these two xBee modules are configured to send data with the same radio frequency. Connect a button to one Arduino as showed in Fig. 1.8 and connect an LED to another Arduino's pin 13. The following codes

Figure 3.2: An xBee shield.

will allow the button to wireless control the LED. When you press the button, the voltage on input pin 2 will change from LOW to HIGH, the transmitter (the Arudino which connects to the button) will send the button state 'H' via xBee module. The receiver (the Arudino which connects to the LED) will receive this state and turn on the LED. When you release the button, the button state 'L' will transmitted and the LED will turn off.

The Arduino code in List 3.3 should be uploaded to the Arduino which connects to the button.

```
#include <SoftwareSerial.h>

short button1 = 2;
short xBeeTx = 11;
short xBeeRx = 12;

char buttonstate = 'L';

SoftwareSerial xBeeSerial(xBeeRx, xBeeTx);

void setup(){
  Serial.begin(9600);
  xBeeSerial.begin(9600);
  pinMode(button1, INPUT);
  }

void loop(){

  if (digitalRead(button1) == HIGH) {  // if button is pressed
         buttonstate = 'H';  }
  else {buttonstate = 'L';}

  xBeeSerial.print(buttonstate);
  Serial.print(buttonstate);
    }
```

Listing 3.3: Transmit the Button State via xBee

The Arduino code in List 3.4 should be uploaded to the Arduino which connects to the LED.

```
#include <SoftwareSerial.h>

char LEDstate;

short LED = 13;
short xBeeTx = 11;
short xBeeRx = 12;

SoftwareSerial xBeeSerial(xBeeRx, xBeeTx);


void setup(){
  Serial.begin(9600);
  xBeeSerial.begin(9600);
  pinMode(LED, OUTPUT);
  }

void loop(){
  if (xBeeSerial.available() > 0){
      LEDstate = xBeeSerial.read();
      Serial.println(LEDstate);
      if (LEDstate == 'H'){
         digitalWrite(LED,HIGH);
       }
       else {
         digitalWrite(LED,LOW);
       }
  }
}
```

Listing 3.4: Receive Button State via xBee and Control the LED

Note: Be careful that two Arduinos use different "COM"s to communicate with the computer.

Task 3.4 *Wireless control the LED by the button.*

3.3.2 Handle Integer Value Between Two Arduinos

In this section, we will transmit an integer from one Arduino to another Arduino via the xBee modules. The receiver Arduino will print out the received integer in the serial monitor. Upload the following two codes to the transmitter and the receiver, respectively.

```
#include <SoftwareSerial.h>

int data = 23;

short xBeeTx = 11;
short xBeeRx = 12;


SoftwareSerial xBeeSerial(xBeeRx, xBeeTx);

void setup(){
  xBeeSerial.begin(9600);
 }

void loop(){
```

```
  xBeeSerial.println(data);
  delay(2000);
  }
```

Listing 3.5: Transmit an Integer via xBee

```
#include <SoftwareSerial.h>

int receive;

short xBeeTx = 11;
short xBeeRx = 12;

SoftwareSerial xBeeSerial(xBeeRx, xBeeTx);


void setup(){
  Serial.begin(9600);
  xBeeSerial.begin(9600);
  }

void loop(){
  if (xBeeSerial.available() > 0){
      receive = xBeeSerial.read();   // receive integer from xBee module
      Serial.println(receive);      // print out the received integer in the
    serial monitor
  }
}
```

Listing 3.6: Receive an Integer via xBee (Wrong)

Open the serial monitor corresponding to the receiver Arduino to observe the data. Instead of receiving 23, the serial monitor will print out three values 50, 51 and 10 which correspond to the ASCII code of characters '2', '3' and '\n'. This means that the receiver treat 23 as a string of characters. In order to receive the correct integer, we should change the receiving code as in List 3.7.

```
#include <SoftwareSerial.h>

String receive;

short xBeeTx = 11;
short xBeeRx = 12;

SoftwareSerial xBeeSerial(xBeeRx, xBeeTx);

void setup(){
  Serial.begin(9600);
  xBeeSerial.begin(9600);
  }

void loop(){
  if (xBeeSerial.available() > 0){
      receive = xBeeSerial.readStringUntil('\n');  // read in data as a String
      int receiveInt = receive.toInt();   // convert a String into an integer
      Serial.println(receiveInt);
  }
}
```

Listing 3.7: Receive an Integer via xBee (Correct)

Task 3.5 *Correctly receive the integer* 23.

Task 3.6 *Connect an potentiometer to the transmitter Arduino as in Fig. 1.9. Connect an LED to the receiver Arduino's pin* 9*. Write programs to use the potentiometer to wireless control the brightness of the LED.*

Hint: *You need to use analogRead() and analogWrite(). Their usage can be found in the section 1.5 and the section 1.6.*

3.3.3 Handle Multiple Data Between Two Arduinos

Now you might wish to use the transmitter Arudino to control two LEDs on the receiver Arduino via xBee modules, one by the button and another by the potentiometer. In order to achieve this task, the transmitter need to send two data, one is a character indicating the button state, another is an integer which control the brightness of the LED. These two data will be transmitted as a string of characters separated by a comma and end with '\n'. At the receiver, it will split the string by comma and convert the corresponding string into the integer.

```
#include <SoftwareSerial.h>

short button1 = 2;

short xBeeTx = 11;
short xBeeRx = 12;

char LEDstate1;    // state of the button
int LEDstate2;     // state of the potentiometer

SoftwareSerial xBeeSerial(xBeeRx, xBeeTx);

void setup(){
  xBeeSerial.begin(9600);
  pinMode(button1, INPUT);
  }

void loop(){
  int temp = analogRead(A0);     //read in value from the potentiometer
  LEDstate2 = map(temp,0,1023,0,255);    // map temp from the range (0,1023) to
    the range (0,255)
  if (digitalRead(button1) == HIGH) {
       LEDstate1 = 'H';
  }
  else {LEDstate1 = 'L';}

xBeeSerial.print(LEDstate1);   //transmit the state of the button
xBeeSerial.print(',');    // seperate two data by a comma
xBeeSerial.print(LEDstate2);  //transmit the state of the potentiometer
xBeeSerial.println();
delay(1000);

  }
```

Listing 3.8: Transmit Multiple Data via xBee

```
#include <SoftwareSerial.h>
#include <string.h>

short LED1 = 13; // LED1 is controlled by a remote button
short LED2 = 9;  // LED2 is controlled by a remote potentialmeter

char LEDstate1;   //store the LED1 state  ('H' or 'L')
int LEDstate2;   // store the LED2 brightness (0 ~ 255)

short xBeeTx = 11;
```

```
short xBeeRx = 12;

SoftwareSerial xBeeSerial(xBeeRx, xBeeTx);

void setup() {

   Serial.begin(9600);
   xBeeSerial.begin(9600);
   Serial.println("Starting..");
   pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);

}

void loop () {
    char *first,*second, *third;
    String serialResponse;
    int len;

    if ( xBeeSerial.available()>0) {
    serialResponse = xBeeSerial.readStringUntil('\n'); //read data into a String
     object serialRespons
    Serial.println(serialResponse);
    len = serialResponse.length();           // get the length of the String
    char buf[len+1];
    serialResponse.toCharArray(buf, sizeof(buf)); // convert a String object to
    char *buf
    //  split buf by comma
    first = strtok(buf,",");   //get the first CharArray splitted by comma
    LEDstate1 = first[0];     //get the LED1 state
    Serial.println(LEDstate1);
    if (LEDstate == 'H') {    // if the LED1 state is 'H', turn on LED1
       digitalWrite(LED1,HIGH);
      }
    else {  digitalWrite(LED1,LOW);}

    second = strtok(NULL,",");   //get the second CharArray splitted by comma,
    NULL means continue processing the previous buffer
    LEDstate2 = atoi(second);    // convert a CharArray to an integer
    Serial.println(LEDstate2);
    analogWrite(LED2,LEDstate2);   //

   // third = strtok(NULL,",");   //get the third CharArray splitted by comma

   }
}
```

Listing 3.9: Receive Multiple Data via xBee and Control the LEDs Correspondingly

Task 3.7 *Control two LEDs remotely via xBee modules.*

3.3.4 Communication Among Three Arduinos

Every xBee modules can join the same personal area network (PAN) by configuring the parameter "PAN ID" to the same value. Configuration can be made by a software X-CTU. Each xBee module has a unique 64-bits serial number printed on the back side of the module. In order to send message to a particular xBee receiver, we need to configure the destination address of the transmitter module to the receiver's serial number. This destination address are stored in "DH (Destination Address High)" and "DL (Destination Address LOW)", each with 32 bits. In this experiment, we set DH= 0 and DL= $0xFFFF$ which configures the transmitter xBee module to broadcast the message in the same personal area network.

Task 3.8 *Now you get three xBee modules configured to the same PAN and set to broadcast messages. Connect them to three Arduinos. Test whether these three xBee modules join the same PAN, i.e., when you use one Arduino to send message, whether both of them receive it.*

Task 3.9 *Use one Arduino (connects to two buttons and two potentiometer) to remotely control* 4 *LEDs on the other two Arduinos (each receiver Arduino connects to two LEDs).*

HINT: *You can assign an ID number to the receivers. The transmitter will send an ID number together with the corresponding data to the receivers. The receiver checks whether the received ID number fits to its own ID number, if it is, then continue to process the following data.*

Chapter 4

Web Site and Database

In this lab, we are going to learn how to acquire data from a web browser from a local computer and store the data into a database at the server computer. Each of you will get user name 'userxx' (xx is a number) from your instructor in order to access to the server and the database. The instructor will guide you to set up two passwords on the server computer. One password allow you to access to your personal directory on the server where you can put your personal files. Another password allow you to access to your personal database. Both your personal directory and database are already created by the instructor with the name of 'userxx' on the server. The instructor will also tell you the IP address of the server, please write it down.

4.1 Get Started with HTML

HTML is the standard markup language for creating Web pages. Now let us try a simple example. Open a text editor, it can be Notepad or Sublime if it is installed on your computer. Save the following codes into a file called "fisrtpage.html".

```
<html>
<head>
<title>My First Page</title>
</head>
<body>

<h1>Here is My Heading</h1>
<p>Here is my paragraph.</p>

</body>
</html>
```

Listing 4.1: The First HTML Page (firstpage.html).

Now use your web browser to open this file. Here <html>,<body>,<p>,etc are called tags. The HTML elements are surrounded by pairs of tags. Only the content between <body> and < /body> are visible in the browser.

Now, this page is only accessible from your own computer. In order to let others to browse this page, you need to put this page on a server. Open WinSCP. This is a software allow you to transfer your files from your local computer to a server. Click 'New Site' on the left window. Choose 'FTP' and 'No encryption' on the right window. The 'Host name' is the IP address of the server. The Port number is 21. Enter your User name and Password and click 'Login' button. You will then enter an interface where the left side is your local directories and the right side is your personal directory on the server. You can transfer the files between your local computer and the server by dragging the files into the corresponding windows. Now transfer the file "firstpage.html" from your local computer to the server. Now anyone can browse this page by enter the URL 'http://xx.xx.xx.xx/user/userxx/firstpage.html' where xx.xx.xx.xx is the IP

address of the server, userxx is your personal directory name on the server which is set to be the same as your user name.

Let's try another example where the page display a table.

```
<html>
<head>
<title>My First Table</title>
</head>
<body>
<table  border='1'>
  <tr>
    <td>ID</td>
     <td>Name</td>
    <td>Gender</td>
   </tr>
  <tr>
    <td>1</td>
     <td>Mary</td>
    <td>Female</td>
   </tr>
  <tr>
    <td>2</td>
     <td>Smith</td>
    <td>Male</td>
   </tr>
</table>
<body>
</html>
```

Listing 4.2: An HTML Table (firsttable.html).

An HTML table is defined with the <table> tag. Each table row is defined with the <tr> tag. A table data/cell is defined with the <td> tag. More examples about tables can be found https://www.w3schools.com/html/html_tables.asp

In HTML, links are defined with the $< a >$ tag:

```
<a href="url">link text</a>
```

Here, "url" is the address of the page your want to link to.

Task 4.1 *Add a link in your first HTML page which links to your table page.*

4.2 Get Started with PHP

PHP is an acronym for "Hypertext Preprocessor". PHP scripts are executed on the server. Now save the following code into a file named 'myfirstphp.php' and copy the file to your server directory. You can browse this page by entering the address of the page: http://xx.xx.xx.xx/user/userxx/myfirstphp.ph

```
<html>
<body>

<?php echo "<h1>My first PHP page</h1>";  ?>


</body>
</html>
```

Listing 4.3: A Simple PHP Page (myfirstphp.php).

You will see from the example that a PHP script starts with <?php and ends with ? > and can be placed anywhere in the document. "echo" is a build-in PHP function which outputs the content inside "".

Note: Don't forget the semicolon after each PHP statement.

In PHP, you can also define variables. A variable starts with the $ sign followed by the name of the variable. The data type of the variables can be String, Integer, Float, etc. Unlike C,C++, PHP doesn't need to declare the data types before you can use them. Now try the following code.

```
<html>
<body>

<?php
 $txt = "world";
 $x = 5;
 $y = 10.5;

echo "Hello $txt! <br>";
echo "Hello  " . $txt . "!" . "<br>";
echo $x + $y;
?>


</body>
</html>
```

Listing 4.4: The PHP Page Created by Variables (variables.php).

4.3 HTML and PHP Forms

The HTML <form> element defines a form that is used to collect user input. Form elements have different types of input elements which are defined by <input> element. Please try the following example to understand different input types. Note that the <fieldset> element is used to group related data in a form and the <legend> element defines a caption for the <fieldset> element.

```
<html>
<body>

<form>
 <fieldset>
    <legend>Personal information:</legend>
   First name:<br>
  <input type="text" name="firstname"><br>
   Last name:<br>
  <input type="text" name="lastname"><br>
    User password:<br>
  <input type="password" name="psw">
    </fieldset>
</form>

<form>
 <fieldset>
    <legend>Color information:</legend>
  <input type="radio" name="color" value="red" checked> Red<br>
  <input type="radio" name="color" value="blue"> Blue<br>
  <input type="radio" name="color" value="yellow"> Yellow
    </fieldset>
</form>

</body>
</html>
```

Listing 4.5: The HTML Form Examples (forms.html).

Now we are going to learn an important input type called "submit". <input type="submit"> defines a button for submitting the form data to a server page for processing input data. This server page is usually a .php script specified in the form's action attribute. In the following example, whenever the submit button is pressed, the server page "myfirstphp.php" will be executed. Note that "myfirstphp.php" is already created from the previous section and it should be in the same directory of your current file.

```
<html>
<body>

<form action="myfirstphp.php">
 <fieldset>
    <legend>Personal information:</legend>
   First name:<br>
  <input type="text" name="firstname"><br>
   Last name:<br>
  <input type="text" name="lastname"><br>
    User password:<br>
  <input type="password" name="psw"><br>
  <input type="submit">
    </fieldset>
</form>

</body>
</html>
```

Listing 4.6: The HTML Form with Submit Button (submit.html).

In order to get the data from the form, an attribute called "method" should be defined in the Form Element. There are two types of methods (GET or POST). POST method is more secure than GET. We will always use POST method in our examples. Now we still use the previous example "submit.html", but replace <form action="myfirstphp.php"> by <form action="myaction.php" method="post">. Now create "myaction.php" on your server directory with the following code:

```
<html>
<body>

<?php
$firstname = $_POST['firstname'];
$lastname = $_POST['lastname'];
echo "Hello $firstname $lastname!";
?>

</body>
</html>
```

Listing 4.7: The PHP Page which Processes the Form Data (myaction.php).

You might already noticed that the input data for "firstname" and "lastname" in the HTML form will be stored in a PHP global variable $_POST.

Task 4.2 *Try to create a web page to select your favorite color (maybe from red, blue, yellow) and when you press the submit button, you will get a printout indicating which color is your favorite color.*

4.4 PHP and MySQL Database

MySQL is a very popular database system. On the server, we already installed a MySQL database system. In the following, we are going to learn how input data into the database from the webpage and how to display the data in a database on a webpage.

A database with the same name as your user name is already created for you on the server. You can manipulate your database directly through an interface called "phpmyadmin". Browse to your server "http://xx.xx.xx.xx" which will show you a startpage of the server. Then click "phpmyadmin" which will start a login page. Enter your username and password. You will then enter your database interface. On the left side of the page, you will find your database, namely "userxx". Under "userxx", all the tables will be listed. Currently, you don't have any table. You can create new tables by clicking "New". On the right side of the page, you can use SQL to query your tables.

However, we are going to introduce you how to manipulate the database by PHP since this will alow your database to interact with the browsers.

Before you can access data in your database from the browser, we need to be able to connect to your database on the server:

```
<?php

 $servername ="xx.xx.xx.xx";  // the IP address of your server
 $username = "userxx" ;        // your user name to access the database
 $password = "xxxxx" ;        // your password to access the database
 $dbname = "userxx" ;         // the database which is already created on the
    server

 $conn = new mysqli( $servername , $username , $password , $dbname ) ;  //set up
     connection to the database on the server
  if ( $conn->connect_error ) { // print out the error message if connection
    fails
     die ( "Connection failed : ".$conn->connect_error ) ;
  }

  else{echo "Database connection is successful.";}  // print out the successful
    message if connection is successful.

   $conn->close();      // close the connection

?>
```

Listing 4.8: The PHP Page which Connects the Database on the Server. (dbconnection.php).

Run this file in your browser. A successful message will be displayed if the connection is successful.

Now you can create a table in your database. The following SQL command will first check whether the table named "temperature" exists, if not, it will create this table with five columns: "rid", "did", "date", "time" and "temperature".

```
CREATE TABLE IF NOT EXISTS temperature(rid SERIAL, did INT, date DATE, time TIME, temperature FLOAT
)
```

The data types of them are Serial, Integer, Date, Time and Float. "Serial" is a automatic increasing data type starting with integer number 1. Every time you insert a new record into the table, "rid" will be automatically increased by 1. "did" stores the ID number of your device. "date" and "time" store the date and time when you collect the data. The standard form of "date" is "year-month-day" and the standard form of "time" is "hour:minute:second". "temperature" stores the current temperature value.

In the PHP script, "$conn->query(\$sql)$" will execute the SQL command defined in \$sql on the database defined in \$conn. The complete PHP script to create the table is as follows:

```
<?php

 $servername ="xx.xx.xx.xx";  // the IP address of your server
 $username = "userxx" ;        // your user name to access the database
 $password = "xxxxx" ;        // your password to access the database
```

```
$dbname = "userxx" ;        // the database which is already created on the
    server

$conn = new mysqli( $servername , $username , $password , $dbname ) ;  //set up
    connection to the database on the server
 if ( $conn->connect_error ) { // print out the error message if connection
   fails
     die ( "Connection failed : ".$conn->connect_error ) ;
 }

 else{echo "Database connection is successful.";}  // print out the successful
   message if connection is successful.
 $sql = "CREATE TABLE IF NOT EXISTS temperature(rid SERIAL,  did INT, date DATE
   , time TIME, temperature FLOAT)";  // create a table called temperature

 if ($conn->query($sql) === TRUE) {  //use this SQL to query the databse
    echo "New table created successfully";
 } else {
    echo "Error: " . $sql . "<br>" . $conn->error;  // print out the error
   message if the query fails
 }

  $conn->close();     // close the connection

?>
```

Listing 4.9: Create a Table in the Database on the Server. (createtable.php).

Open this file once in your browser will create the temperature table. You will then find this table under your database "userxx" in the "phpMyAdmin" interface.

Our ultimate goal is to collect data from the sensors or devices, say, the temperature sensors and store the data into the database. But before doing that, we first test our database by webpages.

In the following, we are going to collect data from webpage and insert data into the table. Create a HTML Form page called "temperature.html" where one can submit the device ID and the temperature value to a PHP script called "insertTemp.php" via POST method. The PHP script "insertTemp.php" will store the device ID and the temperature value into two variables, connect to the database and insert the values into the temperature table. You already learned how to store the data from a HTML Form via the global variable $_POST from the previous section and how to connect to the database in this section. In order to insert the data into the temperature table, you need to add the following command after you connect to the database:

```
$sql = "INSERT INTO temperature (did, date, time, temperature) VALUES ('$did',
    CURRENT_DATE(), CURRENT_TIME(), '$temperature')";  // insert data into
    temperature table

 if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
 } else {
    echo "Error: " . $sql . "<br>" . $conn->error;
 }
```

Listing 4.10: The Commands to Insert Data into the Table).

Here, "temperature (did, date, time, temperature)" tells which columns in the temperature table will accept the data. "$did" and "$temperature" should contain the Device ID and the temperature value passed by the global variable $_POST. "CURRENT_DATE()" and "CURRENT_TIME()" are internal PHP functions returning the current date and the current time.

Task 4.3 *After submit the data in "temperature.html", check whether your temperature table get new records.*

Now you can display your table in an HTML page. Create a PHP file named "displayTemp.php". In the PHP script, you first connect to the database as described before. Then add the following codes into the script. Open this file in the browser will display the last 10 records in your temperature table in a decreasing order by "rid".

```
$sql = "SELECT * FROM temperature ORDER BY rid DESC LIMIT 10"; //select the last
    10 records in decreasing order by rid in the temperature table
$result = $conn->query($sql);  // $result contains the above 10 records

  if ($result->num_rows > 0) {
            echo "<html><body><table border='1'>";  // output table in HTML
            echo "<tr><td>Device ID</td><td>Date</td><td>Time</td><td>
   Temperature</td></tr>";
            while($row = $result->fetch_assoc()) { //$row contains the next
   record in $result
                $deviceID = $row["did"];  // $deviceID contains the value in
   the "did" column
                $date = $row["date"]; // $date contains the value in the "date"
    column
                $time = $row["time"]; // $time contains the value in the "time"
    column
                $temp = $row["temperature"];  // $temperature contains the
   value in the "temperature" column

                echo "<tr><td>$deviceID</td><td>$date</td><td>$time</td><td>
   $temp</td></tr>";

            }
     echo "</table></body></html>";  }
   else  { echo "0 result.";}
```

Listing 4.11: The Commands to Display Last 10 Records the Table.

Task 4.4 *Understand the above codes and display the last* 10 *records in your temperature table.*

Suppose you have two devices with the ID 1 and 2. You may interested in only displaying the temperature of device 1.

Task 4.5 *Now add another Form element in your "temperature.html" where you can select between* 1 *and* 2 *and then submit it via POST method. Based on your selection, the corresponding PHP script should display the temperature of device* 1 *or* 2.

Hint: *You can use the following SQL command to select records based on their devices ID stored in $deviceID.*

```
$sql = "SELECT * FROM temperature WHERE did=$deviceID ORDER BY rid DESC LIMIT 10";
```

At last, you may wish to clear all the records in the table. The SQL command for this purpose is:

```
$sql = "TRUNCATE temperature";
```

Task 4.6 *Create a PHP file to clear the table. Note that you should first connect to the database and then execute the SQL command. Add this PHP page as a link in your "temperature.html".*

Chapter 5

Connect Sensors/Actuators to the Database

In this lab, I am are going to show you how to wireless communicate among sensors, actuators and devices.

5.1 Connect Sensors to the Database

We are going to use the example of temperature sensor in Lab 1 again. You already learned how to read in the temperature from the temperature sensor and print it on the serial monitor.

Task 5.1 *Repeat the experiment of temperature sensor in Lab* 1.

Now we are going to store the temperature value into the temperature table you created in the database. Processing will be the interface to first receive data from the Arduino and then insert the data into the database. We will first let Processing to receive the device ID and the temperature value from the Arduino.

Task 5.2 *Correctly receive the device ID (Integer) and the temperature (Integer) values from the Arduino and print them in the Processing.*

Hint: *Study the Examples 2.15 and 2.16 again and understand how to split data stream on the Processing.*

Processing has a library to deal with the MySQL database. The following code connects the database and insert one record into the temperature table with "did= 1". "temperature= 24" and the date and the time equal to the current date and time.

```
import de.bezier.data.sql.*;

MySQL db;

void setup()
{
  frameRate(1);   // 1 frame per second
  size(200, 200);

  String server = "xx.xx.xx.xx";
  String user = "userxx";
  String pass = "xxxxx";
  String database = "userxx";

  db = new MySQL(this, server, database, user, pass);
  if (db.connect()) {
```

```
      db.query("INSERT INTO temperature(did, date, time, temperature) VALUES(1,
    CURRENT_DATE(), CURRENT_TIME(),24)");
  }

}

void draw(){
}
```

Listing 5.1: Insert a Record from Processing to the Database)

Task 5.3 *Insert one record to the temperature table in the database from Processing.*

Suppose your Processing accept the device ID and the temperature values from the Arduino and store them in the variables "did" and "temperature". You can use the following command to insert these values into the temperature table:

```
db.query("INSERT INTO temperature(did, date, time, temperature) VALUES(%d, CURRENT_DATE(), CURRENT_TIME
(),%d)",did,temperature);
```

To continuously insert the data, please put the above query inside the draw() loop. Set the frame rate to be one frame per second such that the database will not be overloaded by frequent connection.

Task 5.4 *Insert the data collected from the temperature sensor to the temperature table in the database for* 5 *seconds. Display the table from other computer.*

You can also acquire the real-time data from the database by Processing and print or visualize the data in Processing. The following code will get the last temperature value in the database every 1 second and print it out.

```
import de.bezier.data.sql.*;

MySQL db;

int temperature;
int did;

void setup() {
  frameRate(1);   // 1 frame per second
  size(200,200);
  String server = "xx.xx.xx.xx";
  String user = "userxx";
  String pass = "xxxxx";
  String database = "userxx";
  db = new MySQL(this, server, database, user, pass);

}

void draw() {
  if (db.connect()) {
        db.query("SELECT temperature FROM temperature ORDER BY rid DESC LIMIT 1
    ");

        while(db.next()){
        temperature =db.getInt("temperature");  //get the ingeter value of the
    column "temperature"
        println(temperature);
        }
       }

    }
```

Listing 5.2: Acquire Data from the Database and Print it Out in Processing. (displaytemperature.pde)

After getting the data from your database, you can visualize it by Processing. You can reuse the example in 2.14 to visualize the temperature by Processing. However you need to convert your temperature value into data type "float" in order to correctly use the the graphic interface.

```
import de.bezier.data.sql.*;

MySQL db;

int lf = 10;
int temperature;
int did;

void setup() {
  frameRate(1);   // 1 frame per second
  size(330,306);
  String server = "xx.xx.xx.xx";
  String user = "userxx";
  String pass = "xxxxx";
  String database = "userxx";
  db = new MySQL(this, server, database, user, pass);

}

void draw() {
   int i;
  int ypos;
  if (db.connect()) {
         db.query("SELECT temperature FROM temperature ORDER BY rid DESC LIMIT 1
   ");

         while(db.next()){
         temperature =db.getInt("temperature");
         println(temperature);
         }
        }

   ypos = int((100 - float(temperature)) / 100 * 256);
   background(32);
   for(i=0; i<256; i++) {
     stroke(255 - i, 128, i);
     line(25, 25+i, 175, 25+i);
   }

   stroke(255);
   for ( i = 0 ; i < 11 ; i++) {
   int ypos2 = round (25 + 25.6 * i ) ;
   line (173 , ypos2 , 177 , ypos2 ) ;
   textSize (12) ;
   text ( ( ( 10 -i ) * 10) , 178 , ypos2 + 6) ;
   }

   stroke(255, 128);
   line(20,25+ypos, 205, 25+ypos);

   textSize(32);
   text(temperature, 205, ypos+38);

    }
```

Listing 5.3: Visualize the Real-Time Data in Processing. (visualtemperature.pde)

Task 5.5 *Visualize the temperature data from another computer.*

5.2 Connect Actuators to the Database

Now you already learned almost all the necessary steps to connect a sensor to the internet and display the data collected from the sensor from any computers/smartphones connected to the internet. In this section, we will connect an LED to the internet which will allow us to use any computers/smartphones on internet to control the status of the LED. The concept is very similar to connecting the sensor to the internet. We will use a HTML Form webpage to allow user to choose the status of the LED (On, Off, Blink). After submitting the choice from the user, a PHP script running on the server will insert the chosen status to the database. The Processing will get the real-time status from the database and send the status to Arduino which will control the LED based on the status. Now please finish the following task step by step.

Task 5.6 *Reuse the program in 3.1 to control an LED based on your input in the serial monitor. Input 'n' will turn on the LED; input 'f' will turn off the LED; input 'b' will blink the LED once.*

Task 5.7 *Create a table named "LED" in your database similar to temperature table but replace the "temperature" column by "state". "state" will contain the status of the LED with three possible values: 'n','f','b'. It has the data type of "CHAR".*

Task 5.8 *Create a HTML Form allow user to select the LED state among On, Off, and Blink. Create a PHP script to get the LED state from the HTML Form via Post method and insert the corresponding value ('n','f','b') to the LED table in the database.*

Hint: *You can use input type "Radio" in the HTML Form.*

Task 5.9 *Write a Processing program to get the last state from the LED table and send this value ('n','f','b') to Arduino.*

Hint: *Use use "db.getString" to get data of CHAR data type. Check the example in 2.8 and 2.9 to see how to send data from Processing to Arudino.*

Task 5.10 *Control the LED from another computers.*

Chapter 6

Interface with Python

Python is a powerful tool for data science. It would be nature to use it to accept data from the sensors and send them to the database. We will use python3 in the lab.

6.1 Connect Arduino to Python

6.1.1 Transmit Characters from Python to Arduino

Transmit a character to Arduino. Please upload the List 6.1 to Arduino. You need to connect an LED on Arduino pin 9.

```
short LED = 9;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  char state;
  // put your main code here, to run repeatedly:
  if (Serial.available()){
    state = Serial.read();

      if (state == '1'){
         digitalWrite(LED, HIGH);
          delay(1000);
      }
      else{
         digitalWrite(LED, LOW);
         delay(1000);
      }
      }
     Serial.println(state);
    }
```

Listing 6.1: Receive a Character by Arduino

Save the python code in the List 6.2 to a file named "serialtest.py".

```
import serial
import time

ser = serial.Serial('COM20', 9600)

while True:
  ser.write(b'1')
```

```
  time.sleep(1)
```

Listing 6.2: Transmit a Character from Python)

Task 6.1 *Understand the code and run the python code in the command window. Explain the observation. In order to run the python code, you need to type in the command "python serialtest.py" in the command window. Note that you should already be in the directory where you save the python code.*

Now we can try to transmit a string (a sequence of characters) to the Arduino. Try the List 6.1 and the List 6.2.

```
#include <string.h>
short LED = 9;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  String state;
  // put your main code here, to run repeatedly:
  if (Serial.available()){
    state = Serial.readStringUntil('\n');

      if (state == "on"){
         digitalWrite(LED, HIGH);
          delay(1000);
      }
      else{
         digitalWrite(LED, LOW);
         delay(1000);
      }
      }
     Serial.println(state);
    }
```

Listing 6.3: Receive a String by Arduino

```
import serial
import time

ser = serial.Serial('COM3', 9600)

while True:
  ser.write(b'on\n')
  time.sleep(1)
```

Listing 6.4: Transmit a String from Python

Task 6.2 *Understand the code and explain the observation.*

6.1.2 Receiving Multiple Data by Arduino

You can transmit multiple data from Python as a String and separate them by comma. On the Arduino side, we split the String and convert each substring into an integer number. Try the following the codes.

```
#include <string.h>

short LED1 = 9;
short LED2 = 10;
short LED3 = 11;

void setup() {

   Serial.begin(9600);

   pinMode(LED1, OUTPUT);
   pinMode(LED2, OUTPUT);
   pinMode(LED3, OUTPUT);
}

void loop () {
    char *first,*second, *third;
    int data1,data2,data3;
    String serialResponse;
    int len;
    if ( Serial.available()) {
    serialResponse = Serial.readStringUntil('\n'); //read data into a String
    object serialRespons
    len = serialResponse.length();           // get the length of the String
    char buf[len+1];
    serialResponse.toCharArray(buf, sizeof(buf)); // convert a String object to
    char *buf
    //  split buf by comma
    first = strtok(buf,",");   //get the first CharArray splitted by comma
    data1 = atoi(first);       // convert CharArray to integer
    if (data1 == 11)
       digitalWrite(LED1, HIGH);
    //  Second strtok iteration
    second = strtok(NULL,",");   //get the first CharArray splitted by comma
    data2 = atoi(second);
    if (data2 == 12)
       digitalWrite(LED2, HIGH);

    third = strtok(NULL,",");   //get the first CharArray splitted by comma
    data3 = atoi(third);
    if (data3 == 13)
       digitalWrite(LED3, HIGH);
   }
}
```

Listing 6.5: Receive a String of Multiple Integers by Arduino

```
import serial
import time

ser = serial.Serial('COM3', 9600)

while True:
  ser.write(b'11,12,13\n')
  time.sleep(2)
```

Listing 6.6: Transmit a Sting of Multiple Integers from Python)

6.1.3 Receiving Multiple Data by Python

```
int a = 11;
```

```
int b = 12;
int c = 13;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print(a);
  Serial.print(',');
  Serial.print(b);
  Serial.print(',');
  Serial.println(c);
}
```

Listing 6.7: Transmit a String of Multiple Integers to Python

```
import serial
import time

ser = serial.Serial('COM18', 9600)
receivestring = ser.readline()   # ignore the first read in

while True:
  receivestring = ser.readline().decode("utf-8").strip()  # read one line and
    convert bytes to a string
  receivelists = receivestring.split(',')
  print(receivelists)
  time.sleep(1)
```

Listing 6.8: Receive a String of Multiple Integers from Python)

Chapter 7

Final Project

In this lab, you should try to design a wireless sensor network system in a group (2 to 4 persons). After finish the project, your group should demonstrate your work to the instructor/TAs with necessary explanation and write a documentation. The documentation should include the introduction of your project, the motivation and usability of your project; hardwares and softwares you use in the project; circuit design diagram and necessary explanation of the technical aspects. The grading criterion for the final project is as follow:

- The technical complexity (30%);
- The usability (30%);
- The documentation (40%)