**Deadline (Challenge 1): Feb 19, 2023 – 11:30 pm**

**Deadline (Challenge 2): Feb 26, 2023 – 11:30 pm**

NOTE: COMPLETE THE STEPS IN THIS DOCUMENT BEFORE WORKING ON CHALLENGES 1 AND 2!

List of steps:

* Step 1: Getting familiar with the Safety Shield
* Step 2: Connect the Safety Shield
* Step 3: Getting familiar with While
* Step 4: Practicing Loop Statement
* Step 5: Controlling a Servo Motor using a Potentiometer
* Step 6: One more application with servo motor
* Step 7: Distance sensor
* Step 8: Go to Section 2

**Possible points:** **110** points (100 pts for this assignment, and 10 bonus points will be added to your previous studio memo 3 in case you missed any points)

**Section 1. Exploring safety systems**

In this studio, each team will implement a camp safety system. Using the sensors and actuators given in class, each team will use PEEB to protect a camp at night.

The sections below outline the deliverables' requirements and steps, including graphical and code elements.

**Parts you need to complete sections 1.1 and 1.2**

* PEEB kit
* USB Cable
* Safety shield (Black printed circuit board inside the box)
* Wires
* Distance sensor
* Touch sensor

Step 1. Getting familiar with the Safety Shield

The safety shield belongs to the family of educational kits for Arduino that form a part of the PEEB project. This shield uses several sensors, actuators, LEDs, and internally connected buttons. Table 1 outlines the connection of each element with its respective pints.

**Table 1.** Internal pinout connections

|  |  |
| --- | --- |
| Components | Pin |
| Button | D10 |
| LED | D13 |
| Touch sensor | D8 |
| Servo motor | D9 |
| Potentiometer | A2 |
| Distance (trig) | D5 |
| Distance (echo) | D6 |

Step 2. Connect the Safety Shield

Before running the experiment today, make sure you complete the steps below:

1. Attach the safety shield to Arduino
2. Plug in the USB cable
3. Open the Arduino IDE

Step 3. Getting familiar with While

* 1. **While statement**

The while loop in Arduino is a programming construct that allows you to repeat a set of instructions as long as a certain condition is true. The basic structure of a while loop looks like this:

**while (condition) {**

**// instructions to repeat**

**}**

Here's how it works:

* First, the program checks whether the condition in the parentheses is true or false. If the condition is false, the program skips over the instructions inside the curly braces and moves on to the next line of code outside of the loop.
* If the condition is true, however, the program executes the instructions inside the curly braces. When it reaches the end of the instructions, it goes back to the beginning of the loop and checks the condition again.
* This process repeats over and over again until the condition becomes false. At that point, the program exits the while loop and moves on to the next line of code outside of the loop.

While loops are useful in Arduino programming when you want to repeat a set of instructions a certain number of times or until a certain condition is met. For example, you might use a while loop to keep a motor running until a sensor detects that a certain position has been reached, or to repeatedly check a button input until it is pressed.

Just be careful to avoid infinite loops, where the condition is never false and the program gets stuck in the loop forever. To prevent this, make sure your condition is properly set up and that the instructions inside the loop eventually change the condition so that it becomes false.

Step 4. Practicing Loop Statement

In this example, you assign a pin for Button 1 (Button 1 is already connected to D10 on the Safety shield) and set Button 1 as INPUT, and a pin for LED 1 (LED 1 is already connected to **D13**) and set it as OUTPUT. You will tell the Arduino to turn the LED **while** the button is being pressed and to keep the LED off when the button is not being pressed. When the button is released, the circuit breaks, and the LED will turn off again.

Similarly to our last studio project, start your code by defining **constants**.

**Note:** *You must declare it before void setup.*

const int ledPin = 13;

const int buttonPin = 10;

In the **setup( )**, you are going to set the LED pin as output and the button pin as input.

pinMode(ledPin,OUTPUT); //set LED pin as output

pinMode(buttonPin,INPUT); //set button pin as input

In the **loop( )**, you will use the variables defined earlier in your code. First, you read the button state by using the function **digitalRead**.

int buttonState = digitalRead(buttonPin);

In this section, you are going to use the statement **while**. More specifically, while the button is pressed, turn the LED On and wait.

while (buttonState == HIGH) {

digitalWrite(ledPin, HIGH); // turn LED on

buttonState = digitalRead(buttonPin); // update button state

}

If the button is not pressed, turn the LED off.

digitalWrite(ledPin, LOW);

Your final code should look like this:

Text

Description automatically generated

**Use it**

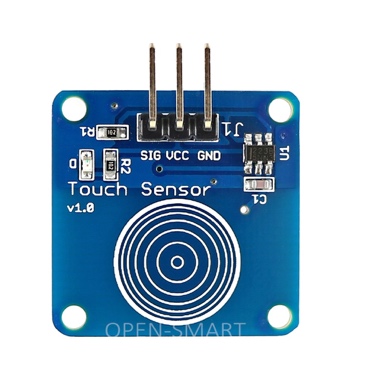
After writing your code, time to test your first while statement. First, verify your code by pressing **Verify** . Then, press UPLOAD .

**NOTE:** *If you see any error message, read the error message and check for any missing element of your code, such as brackets or semi-colons.*

**Your turn!**

In this practice, you will familiarize yourself with a touch sensor (see Figure 1). A touch sensor is an electronic device that detects when something touches or comes into contact with its surface. When you touch the sensor, it sends a signal to a microcontroller, which can then use that information to trigger an action or response.

The most common type of touch sensor works by using a conductive material, like a conductive foam or metal plate, to detect changes in electrical capacitance. When you touch the sensor, you change the electrical capacitance at that point, and the sensor can detect that change. Some touch sensors use a different method, called resistive sensing. In this case, the sensor is made up of two layers of material that are separated by a small gap. When you touch the sensor, the two layers come into contact, and the resistance between them changes. The sensor can detect this change in resistance and use it to trigger an action.



**Figure 1.** Touch sensor

In our class, we will use the touch sensor illustrated in Figure 1. Before connecting the sensor to the Safety Shield, triple-check the connections. In this case, SIG will be connected to D8, GND will be connected to GND, and VCC will be connected to +5V.

**Your task: Create one application where you turn a LED on while you press the touch sensor AND button 1 at the same time.**

Step 5. Controlling a Servo Motor using a Potentiometer

The Potentiometer (Figure 2) will be used to control the position of the servo motor (Figure 3).



Figure 2. Potentiometer



Figure 3. Servo motor

The project allows us to control the shaft at angles between 0 and 180 degrees. We can also set the rotation of the shaft at different speeds. Servo motor has three terminals signal **(D9)**, power **(+5V)**, and ground **(GND)**. The power pin of the servo motor is connected to the PWM pin of the Arduino board. Here, we have connected the power terminal to pin 9 of the Arduino board.

You could try to control the servo motor through the Arduino digitalWrite() functions but that would require us figuring out the right values to write, and the timing for writing those values. That’s too much work.

We are lucky, though, because with the Arduino IDE we get the Servo library, which contains functions that allow us to easily work with servo motors. First, we'll write the sketch using the Servo library.

You first include the Servo library (“#include <Servo.h>”), and create the variable myservo that you can use as a handle to the Servo object (“Servo myservo;”).

#include <Servo.h>

Servo myservo;

You also need to declare the variables used to connect the potentiometer and read the value from the analog pin.

int potpin = A2;

int val;

In the **setup** function, you tell the Arduino that the control wire from our servo motor is attached to digital pin 9 (“myservo.attach(9);”).

myservo.attach(9);

The work is done in the loop() function, where you use a method to read the current position of the potentiometer and then return the value. In the loop section of the code, we read the value from the analog pin A2 with the function analogRead. Arduino boards contain a 10-bit analog to digital converter (ADC), so this gives us a value between 0 and 1023 depending on the position of the potentiometer. Because the servo motor can only rotate between 0 and 180 degrees, we need to scale the values down with the map() function. This function re-maps a number from one range to another.

val = map(val, 0, 1023, 0, 180);

Lastly, we write the angle to the servo motor:

myservo.write(val);

delay(15);

Your final code should look like this:

#include <Servo.h>

Servo myservo; // create servo object to control a servo

int potpin = A2; // analog pin used to connect the potentiometer

int val; // variable to read the value from the analog pin

void setup() {

myservo.attach(9); // attaches the servo on pin 9 to the servo object

}

void loop() {

val = analogRead(potpin); // reads the value of the potentiometer (value between 0 and 1023)

val = map(val, 0, 1023, 0, 180); // scale it to use it with the servo (value between 0 and 180)

myservo.write(val); // sets the servo position according to the scaled value

delay(15); // waits for the servo to get there

}

DID YOU KNOW? Controlling multiple servos is just as easy as controlling only one but you need to use an external power source because Arduino cannot provide enough power to all servo motors. Here is one example in Figure 4.

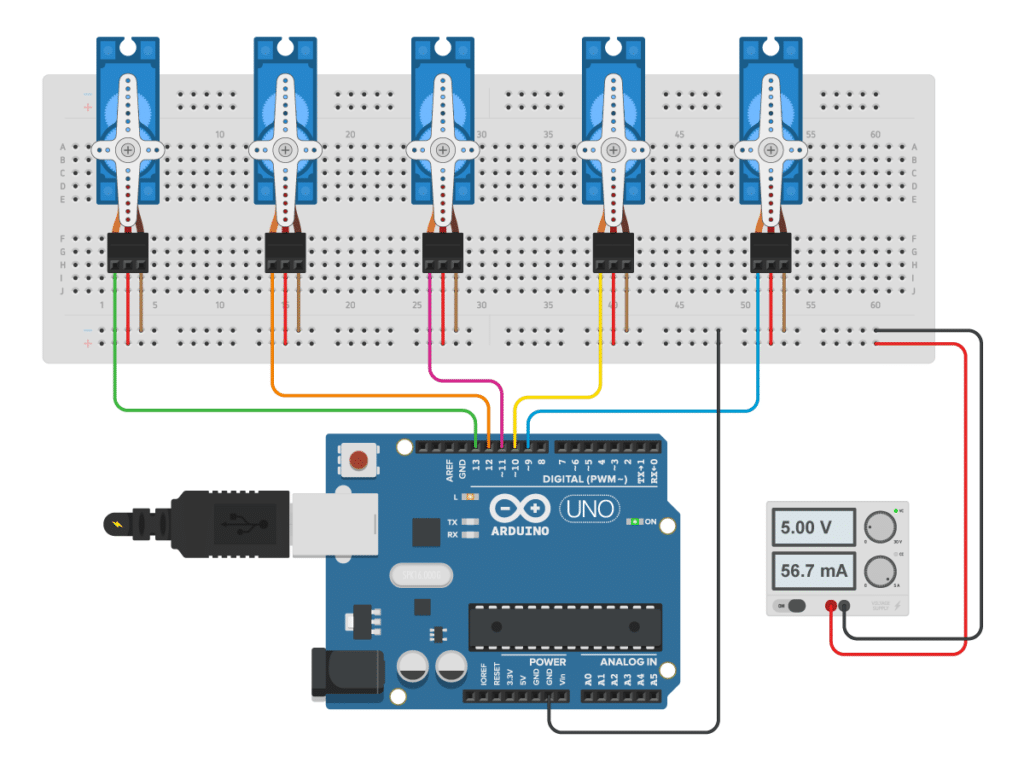


Figure 4. Multiple servo motors connected.

Step 6. One more application with servo motor

In this example, let’s explore one more servo motor functionality. In this example, we will move the servo position to 90 degrees while the touch sensor is pressed.

This code starts by including the Servo library, which allows us to control the servo motor. The code then sets up the push button pin as input with an internal pull-up resistor, and attaches the servo motor to pin 9 using the attach function.

#include <Servo.h>

Servo myservo; // create servo object to control a servo

int buttonPin = 8; // push button pin

int servoPos = 0; // variable to store the servo position

int buttonState = 0; // variable to store the button state

void setup() {

myservo.attach(9); // attaches the servo on pin 9 to the servo object

pinMode(buttonPin, INPUT\_PULLUP); // sets the push button pin as input with internal pull-up resistor

}

In the main loop, the code reads the state of the push button using digitalRead. If the button is pressed (i.e. buttonState is LOW because we set up the pin with an internal pull-up resistor), the code sets the servoPos variable to 90 degrees. If the button is not pressed, the servoPos variable is set to 0 degrees. The code then tells the servo motor to move to the position specified by servoPos using the write function, and waits for 15ms for the servo to reach the position using the delay function. This code should move the servo motor to 90 degrees when the button is pressed, and return it to its original position (0 degrees) when the button is released. If the servo motor does not move to the correct position, you may need to adjust the value of servoPos to match the actual position of your servo motor.

void loop() {

buttonState = digitalRead(buttonPin); // read the state of the button

if (buttonState == LOW) { // if button is pressed

servoPos = 90; // set servo position to 90 degrees

}

else { // if button is not pressed

servoPos = 0; // set servo position to 0 degrees

}

myservo.write(servoPos); // tell servo to go to position in variable 'servoPos'

delay(15); // waits 15ms for the servo to reach the position

}

Test your code now!

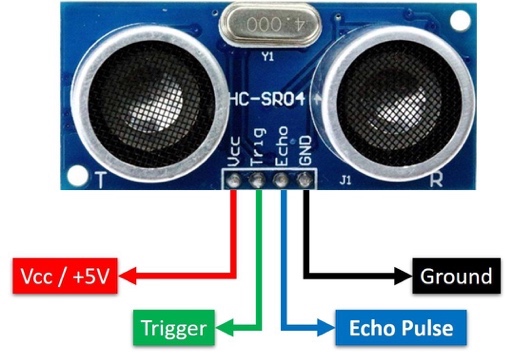
Step 7. Distance sensor

Here is the last application to test your Safety shield. But let’s see the entire code first. Then, let’s understand each part of the code.

Text

Description automatically generated

Now let me explain each part of the code. First, take a look at the sensor pinout in Figure 5.



**Figure 5.** Ultrasonic sensor pinout

const int trigPin = 5; and const int echoPin = 6; are variables that store the pin numbers where the ultrasonic sensor is connected. In this example, the trig pin is connected to pin 5 and the echo pin is connected to pin 6.

void setup() is a function that runs once when the Arduino is turned on or reset. In this function, we initialize the serial communication with the computer (using Serial.begin(9600)), and set the pins of the trig and echo pins as output and input respectively.

void loop() is a function that runs repeatedly as long as the Arduino is turned on. In this function, we first set the trig pin to LOW to ensure that it is not transmitting any signal. Then, we wait for 2 microseconds to let the pin settle.

Next, we set the trig pin to HIGH for 10 microseconds. This sends a sound wave from the ultrasonic sensor. The sound wave will bounce off any object in front of the sensor and then return to the sensor.

After the 10 microseconds, we set the trig pin back to LOW. This tells the sensor that we have finished sending the sound wave.

We then use the pulseIn() function to measure the amount of time it takes for the sound wave to travel from the sensor, bounce off an object, and return to the sensor. We store this duration in the variable duration.

We then use a simple formula to convert the duration into distance. The speed of sound in air is about 343 meters per second. Since the sound wave has to travel to the object and back, we divide the duration by 2. We then multiply this by 0.034, which is the distance that sound travels in one microsecond. This gives us the distance in centimeters, which we store in the variable distance.

We then use Serial.print() and Serial.println() to display the distance on the serial monitor.

Finally, we add a delay of 500 milliseconds to give some time for the next measurement.

**Use it**

After writing your code, time to test your distance sensor. First, verify your code by pressing **Verify** . Then, press UPLOAD .

**Section 2. Assignment**

Assignment goal: Create a safety system to protect your camp. No physical prototype of your camp is needed to build the safety system in this assignment.

Project requirements:

* Your project must contain at least two different sensors.
* Your project must contain at least one actuator (servo motor) playing an important role in your design.
* Your project must have one output (e.g., light, sound, etc.) to indicate any approximation. You should have two minimum output states to detect when someone is detected near the camp, or no one is near the camp.

**Deliverable 1 - Deadline (Preliminary draft): Feb 19, 2023 – 11:30 pm**

**Challenge 1.** **a)** Sketch a detailed safety system to protect your camp, including all dimensions, sensor and actuator locations, and any additional information that will aid in understanding the proposed design and expected outputs. **Be creative!!** Assume that your camp has the following configuration in Figure 6. You can decide your camp dimensions.



**Figure 6.** Your camp

You can use hand drawing or any other software in this first draft. Also, include **b)** the expected programming logic behind your circuit and how you intend to implement that according to the project requirements described above. In this initial draft, **DO NOT** include any code. You must develop the code in the classroom. You can use flowcharts ([More information here](https://www.edrawsoft.com/explain-algorithm-flowchart.html)) or any other diagram. **One page max!**

**IMPORTANT:** Keep in mind that no substantial changes to the project and code logic will be allowed once the draft is submitted. Discuss a realistic plan with your team before the deadline. If you implement a solution different than what you proposed in your draft, you must provide a clear justification behind your changes and why you did change it. During next class, you **CANNOT** use any document besides your draft and the [Arduino documentation](https://www.arduino.cc/reference/en/) to develop your code.

**Deliverable 2 - Deadline (Final Memo): Feb 26, 2023 – 11:30 pm**

**Challenge 2.** The final memo **must** contain a professional representation of your design and steps to build and test your prototype. You must implement the project described in Challenge 1 and write a technical memo describing the process. For this project, the ***theory*** will include **Pulse-width modulation (PWM)** and **description of each sensor used in your project**. You must do your research to include additional information about each of these theories. Do not forget to cite your references. Your ***method*** must include the planning process described in your draft and the implementation process to execute your plan. Your final memo must be written in a way that any person will be able to replicate your process.

**Grading Rubric**

|  |  |  |
| --- | --- | --- |
| Description | Expectation | Met? |
| Formatting – Final Memo (90 points) | | |
| General | All instructions and formatting standards were followed. | /2 |
| Writing is in past tense and third person. | /2 |
| Code includes meaningful comments, and it is included as text (**NOT** screenshot) – Code must be placed in Appendix | /3 |
| Figures, labels, and equations formatted correctly | /3 |
| Front cover properly formatted | /3 |
| Abstract is clear and complete | /3 |
| Contents include every single section in your document | /2 |
| Introduction | Clear and concise introduction to paper | /2 |
| A brief section describing individual roles and contribution to each challenge | /2 |
| Theory | Ties together information from class. Author’s introduction demonstrates a theoretical understanding of PWM | /7 |
| Ties together information from class. Author’s introduction demonstrates a practical understanding of each sensor used in the project | /7 |
| Authors used at least **two** external references for each theory | /5 |
| Methods | The paper communicates the methods well. Circuit schematics, tables, and equations provide important details to the reader and are properly labeled and described | /12 |
| The report clearly outlines the tools and procedures used for project design | /8 |
| Discussion | Robust discussion of your results and decisions for each challenge, including potential limitations | /13 |
| Conclusion | Meaningful conclusion, and includes ideas not already discussed in paper | /10 |
| References | References properly organized and cited | /3 |
| Appendix | Appendix properly organized and cited | /3 |
| Challenges (20 points) | | |
| Challenge 1  (Submit as a simple memo) | Project design includes all dimensions, sensors, actuators, and expected outputs | /4 |
| Students demonstrated systematic thinking behind the sensor choice and implementation | /2 |
| Clear code planning | /2 |
| Organization is clear and professional | /2 |
| Challenge 2  (Submit as a final memo) | Students followed the initial plan presented in Challenge 1 | /3 |
| Project is functional | /4 |
| Project followed the requirements | /3 |
| Total score = | | /110 |