Applications and Computation for the Internet of Things

2ST Lab Work: Sensing the Physical World

Group: 12		
Student 1	98380	Dominika Florczykowska
Student 2	97144	Pedro Mendes

1 Goal:

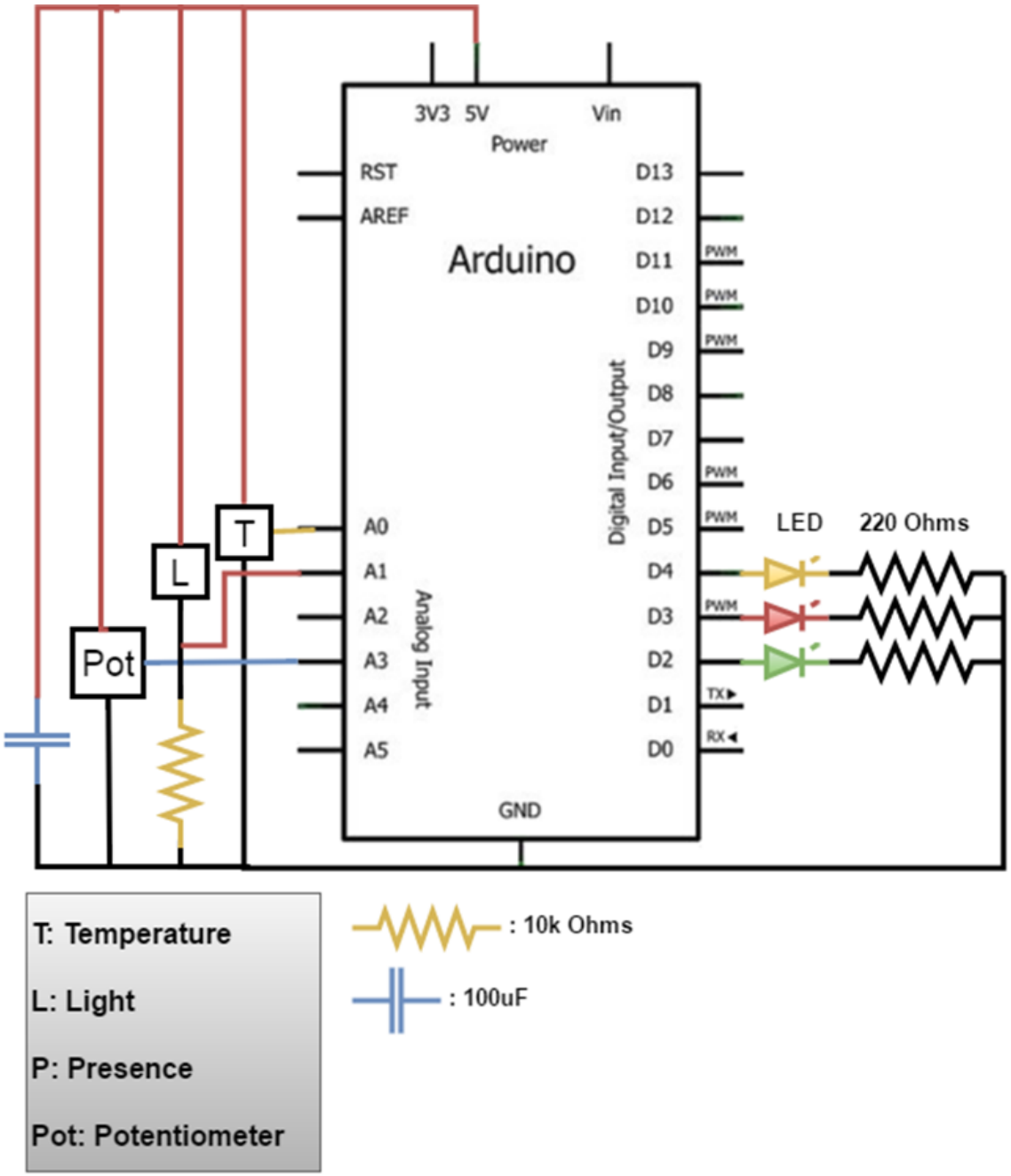
The goal of this work is to sense physical quantities and to control actuators according to the measured environment.

2 Description:

Build an embedded system using an Arduino UNO board to simultaneously control 3 LEDs (the actuators) depending on the state of 3 different sensors – temperature, rotation angle (potentiometer) and light intensity.

- Temperature — The yellow LED associated with the temperature sensor must be turned on when the temperature read is greater than 26 oC (value to be eventually redefined at the laboratory according to the environment conditions).
- Rotation — The green LED controlled by the potentiometer must blink with a period of time varying continuously between 0.2 and 2 seconds, depending of the rotation angle applied to the potentiometer.
- Light — The red LED for the light intensity function must change continuously its own light intensity based on the light intensity sensed in the surrounding environment from
  - Darkness → LED fully ON, to
  - Brightness → LED OFF

A diagram of the circuit is represented in the figure.



3 References

1. https://www.arduino.cc/en/Reference/digitalWrite
2. https://www.arduino.cc/en/Reference/AnalogRead
3. https://www.arduino.cc/en/Reference/Serial
4. https://www.arduino.cc/en/Tutorial/Calibration
5. https://www.arduino.cc/en/Tutorial/PWM
6. https://www.arduino.cc/en/Reference/Delay

4 Mapping analog measurements

Usually the digital readings retrieved from sensors do not correspond directly to the value of the physical quantity, but rather to values between 0 and a maximum binary value (such as, for a 10 bits word, $1023 = 210 - 1$). Therefore some mapping may be required.

When the value is relative just to an offset a simple mapping is adequate, but in general a more complex scale conversion will be needed. The map function, available in the Arduino IDE, simplifies these types of conversions. For example, when rotating a Servo motor with input from a potentiometer we know that when the value of the potentiometer is 0 then the servo angle must be 0° as well. Logically, if the value of the potentiometer is 1023 (its maximum reading) then the servo angle must be 180° (its maximum position):

$$map(value, 0, 1023, 0, 180)$$

Besides this some sensors require a linearization of its transfer function (physical quantity → electrical quantity, such as voltage). For example, to convert the reading from a circuit with a temperature sensor (in our case a temperature dependent resistor) to the real temperature several transformations may be required

- to linearize the transfer function of the sensing device $RT = f(T)$, and
- to linearize the transfer function of the circuit in which the sensor is included.

For the device and configuration to be used (based on a TMP35 component) check the function

$$T = (((sensorvalue/1024.0) \times 5.0) - 0.5) \times 100$$

5 Programming with analog sensors:

To access an external analog sensor you must attach it to an Arduino analog pin. The software allocation of a sensor to an analog pin is done using the following code:

```
int const tempSensor = A0;
```

Where `A0` is the physical pin where the sensor is attached.

To read the value assigned to a specific pin use the Arduino function `analogRead(PIN)`, as follows:

```
int temperatureValue = analogRead(tempSensor);
```

6 Debug:

In order to control some variables and debug your program it is possible to print them to the Serial Monitor available in the Arduino IDE. This kind of tool is useful, for example, to keep track of the temperature variation or to help the calibration of the system.

To use this feature, first start a serial communication with the PC:

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

Then, just `Serial.print` your variables and/or strings:

```
Serial.println(temperatureValue);
```

7 Program the application:

Structure the program modularly clearly segmenting the program in separate code blocks, one for each function to be implemented (e.g. read temperature sensor, control temperature alarm). In the next laboratory work you will have to distribute several functions across separate Arduino controllers.

Avoid, or use very carefully, the delay function provided in the Arduino IDE since it just stops the execution flow during the specified time interval.

7.1 Code:

7.1.1 File: sensor.hpp

```
class Sensor {

  public:
    using Feedback = void (*)(Sensor const&, u16);
    int const sensor;
    int const led;

    constexpr explicit Sensor(int const sensor, int const led, Feedback const f)
        : sensor(sensor), led(led), feedback(f) {}

    void setup() const { pinMode(led, OUTPUT); }

    void update() const { (feedback)(*this, get_sensor_voltage()); }

  private:
    const Feedback feedback;
    u16 get_sensor_voltage() const { return analogRead(sensor); }
};
```

7.1.2 File: lab2.ino

```
#include "sensor.hpp"

enum { TemperatureThreashold = 28 };

enum LEDS {
    Yellow = 5,
    Red = 6,
    Green = 9,
};

enum Sensors {
    Temperature = A2,
    Light = A1,
    Potentiometer = A0,
};

const Sensor SENSORS[] = {
    Sensor(
        Temperature,
        Yellow,
        [](Sensor const& s, u16 voltage) {
            auto degreesC = (((voltage / 1024.0) * 5.0) - 0.5) * 100.0;
            if (degreesC > TemperatureThreashold) {
                digitalWrite(s.led, HIGH);
            } else {
                digitalWrite(s.led, LOW);
            }
        }),
    Sensor(
        Light,
        Red,
        [](Sensor const& s, u16 value) {
            analogWrite(s.led, map(value, 1, 1021, 255, 0));
        }),
    Sensor(Potentiometer, Green, [](Sensor const& s, u16 value) {
        static u32 last_blink_time = 0UL;
        static bool led_on = false;
```

```
        u32 const half_blink_interval = map(value, 0, 1023, 100, 1000);
        u32 const now = millis();
        if (last_blink_time + half_blink_interval < now) {
            digitalWrite(s.led, led_on);
            led_on = !led_on;
            last_blink_time = now;
        }
    })};

void setup() {
    for (auto& s : SENSORS) s.setup();
    pinMode(LED_BUILTIN, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    for (auto& s : SENSORS) s.update();
}
```

7.2 Questions:

Question: For each of the three pairs sensor-actuator describe:

1. the mapping process implemented;
2. the calibration setup;
3. the process, or technique, used to modulate the behaviour of the actuator;
4. the setup prepared to demonstrate the functionality of the system.

Answer: The Arduino has an `analogRead` range from 0 to 1023, and an `analogWrite` range from 0 to 255, therefore the data from any analogRead needs to be mapped to fit into the smaller range.

- **Temperature:** This sensor's value had to be linearized as explained in Section 4. Using the function described there,

  $$T = (((sensorvalue/1024.0) \times 5.0) - 0.5) \times 100$$

  we where able to obtain the temperature in degrees.

  To demonstrate the functionality simply touch the sensor with both fingers to increase the temperature above 25 degrees, this constant was picked for being above room temperature, but low enough that body heat can be enough to light up the LED.

- **Light:** To calibrate the sensor 2 experiments were performed to obtain the 2 extremes, first the sensor was placed in a dark room with an opaque piece of plastic on top of it to obtain the lowest possible reading, which was '1', then the sensor was exposed to a bright white light (phone flash light) to obtains the highest possible reading which was 1021.

  These values were then mapped to the accepted range of the `analogWrite` function $[0, 255]$. This interval is actually inverted when passed to the map function since the LED will light up at its highest intensity when the sensor reads lowest level of light in the environment.

  $$map(value, 1, 1021, 255, 0)$$

  The modulation of the behaviour is just a direct call to `analogWrite` with the value returned from `map`.

- **Rotation:** The calibration of this sensor was simple, we measured the value read from `analogRead` when it was at the minimum and maximum rotation, these mapped exactly to 0 and 1023. Since we work in milliseconds, we then map this range to the range $[100, 1000]$ (0.1 seconds to 1 seconds) which will be stored as the current half blink period (in the code named `half_blink_period`).

Then to modulate the behaviour of the actuator, the program checks the `millis` function and the last time the led state has been changed, and sees if `half_blink_period` has passed, if so, then the led state is toggled.

Question: What is the system software pattern of the application?

Answer: Round Robin pattern. All the tasks are executed continuously in the main loop in the same order (one at a time). Each task checks for the value read by a corresponding sensor and responds by showing correct output with the use of LED.

Question: What are the timing constraints of the system?

Answer: The system must complete each round of the round robin loop in less than 0.1 seconds to make sure the blink period of the green led is maintained in the worst case