Instituto Superior Técnico

Applications and Computation for the Internet of Things

2022-2023

2nd Lab work: Network of Sensors and Actuators[1]

Goal

The goal of this work is to sense physical quantities and to control actuators according to the measured environment. Sensors and actuators will be driven by controllers communicating over a wired link based on the I2C protocol.

Description

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

- The RED LED associated with the temperature sensor must be turned on when the temperature read is greater than 26 ºC (to be eventually redefined at the laboratory).
- 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 applied to the potentiometer (0.2 s <- turn left, turn right -> 2 s).
- The YELLOW LED for the light intensity function must continuously and "linearly" change its own light intensity based on the light intensity sensed in the surrounding environment from
    - Darkness → LED fully ON, to
    - Brightness → LED OFF

  (See Reference 5 about Pulse Width Modulation.)

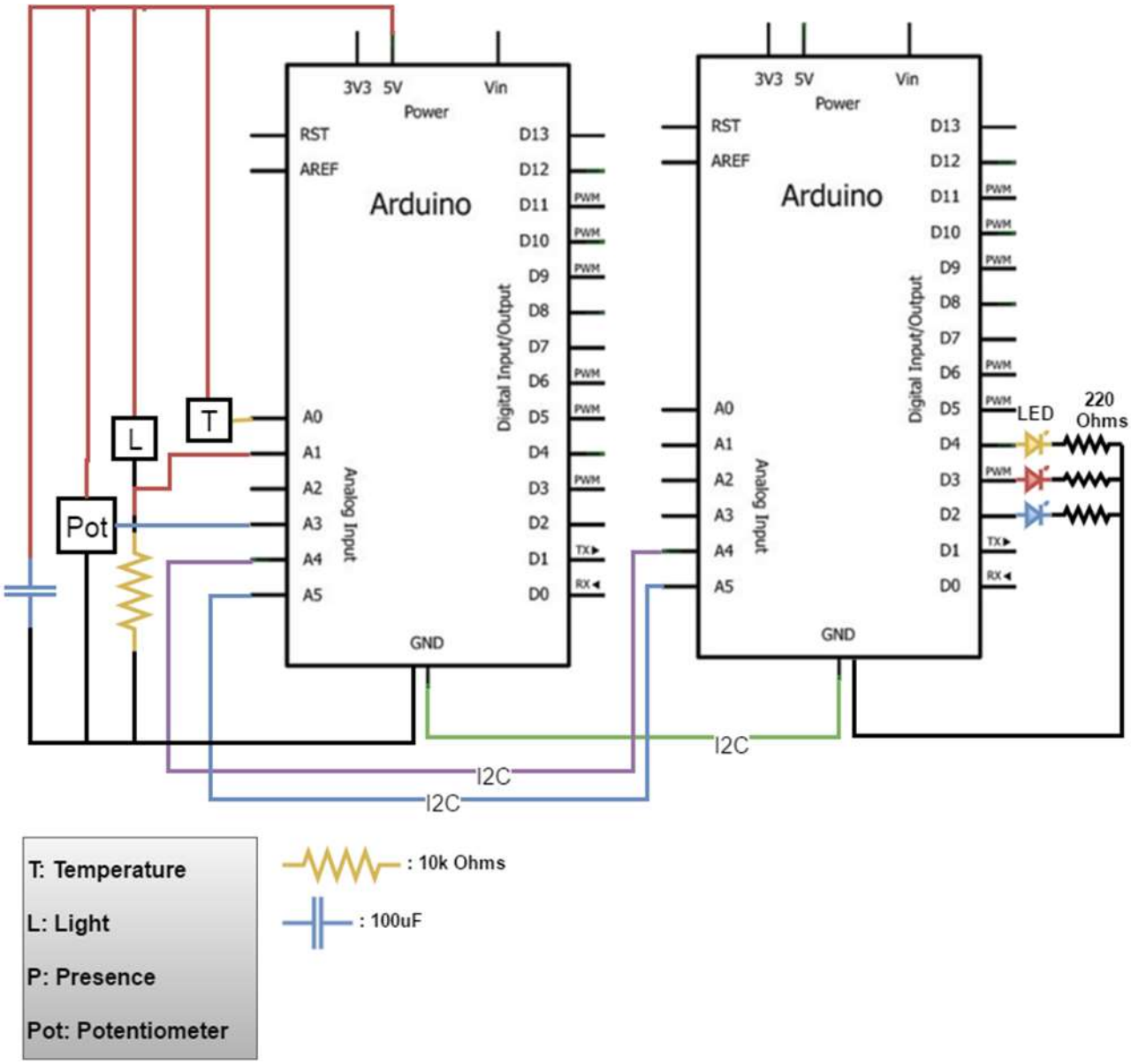
  "Darkness" and "brightness" are reference conditions:
    - Darkness – sensor fully covered;
    - Brightness – sensor under a ceiling light.[2]

In the first part of the work

- Sensors will be connected to an Arduino controller;
- Actuators will be driven by another Arduino controller.

A diagram of the circuit to be assembled is presented in the figure.

[1] V2.0, 28 Nov 2022. V1.0, Nov 2021.
[2] Not a smartphone flashlight.



(in this work ignore “P: Presence”)

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
7. https://www.arduino.cc/reference/en/language/functions/time/millis/
8. https://www.arduino.cc/en/Reference/Wire

Recommendations

In order to fulfill your work with security and not damaging the hardware involved, remember to carry out the recommendations below. As you are working fill the boxes to be certain that you fulfill all security measures.

Always work with the circuits disconnect from its power sources.	
Call the teacher, or responsible for the laboratory, before you connect the circuits to its power sources.	
Make sure the circuit is well connected (resistors, capacitors, etc.) to prevent a short circuit, or damage to the hardware.	

To ease the connection of each controller to its specific interface circuits – sensors on one controller, actuators on the other controller – follow the guidelines for breadboard layout presented and check any issues at the beginning of the first lab session with the teacher.

The only signals (lines) common to both controllers are the two I2C lines and GND.

To interconnect both controllers, avoiding electrical hazards and noise, consider the following requirements:

- supply power (VCC) to each Arduino through its own USB cable;
- do not connect in any other way the VCCs of the two Arduino boards; VCCs **must not be directly interconnected**;
- connect both USB cables to the same computer running the IDE;
- connect the grounds (GND = 0 V) of the two Arduino boards (the single green connection on the diagram).

If you want to assemble the circuits of the interfaces of both controllers on the same breadboard, to simplify the test of the system with two controllers consider the following guidelines:

- mount each interface on a separate half of the breadboard – sensors on one side, actuators on the other side;
- each half of the breadboard must have its own power supply – VCC and GND lines
    - GND lines shall be interconnected through the GND line of the I2C bus;
    - VCC lines must not be interconnected.

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 = 2^{10} - 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 $R_T = f(T)$, 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

$$T = (((sensor\ value\ /\ 1024.0) * 5.0) - 0.5) * 100$$

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:

$$int\ temperatureValue = analogRead(tempSensor);$$

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:

$$void\ setup()\{\ \ldots;\ Serial.begin(9600);\ \ldots;\}$$

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

$$Serial.println(temperatureValue);$$

Inter-Integrated Circuit (I2C) Communications

To put the system to work it is necessary to implement the communication link connecting the two controllers. For this we will use I2C communications.

I2C is a serial communications bus implemented with two bidirectional lines – Serial Data Line (SDA), and Serial Clock Line (SCL) – and the GND reference. I2C works in a Master-Slave protocol where one or more Slaves can be connected to one Master on the same bus. There are no fault tolerance or recovery mechanisms included in the protocol. If necessary they must be provided at the application level. There is available an Arduino Library to handle at the application level communications over an I2C bus (see Reference 7 about the Wire library).

To start communications between controllers both Arduinos must start the Wire communication:

```
void setup() {
     Wire.begin(8);
}
```

`Wire.begin` only receives an argument if the board is a Slave. Masters do not need this address.

Subsequent communication transfers may be performed as Master-Writer:

```
void loop() {
     // Master writes for a Slave to read
     Wire.beginTransmission(8);
                          // Transmission to port 8 of the I2C bus
     Wire.write("value:");
                    // Wire.write(string) reads every char as a byte
     Wire.write(80); // Wire.write(int) reads int as byte
     Wire.endTransmission();
     ...
}
```

On the Slave side, when some message is detected, the I2C port triggers an interrupt. The Wire Library uses the `onReceive` function to associate a callback function to the interrupt:

```
void setup() {
     …
     Wire.onReceive(callbackFunction);
}

void loop() {
```

```
      …
}
```

Corresponding to the previous example – on the Master side – the following function – on the Server side – will consume and print the string and will print the integer separately afterwards.

```
void callbackFunction(int i) {
      while (1 < Wire.available()) {
            // make sure there is something to read
            char c = Wire.read(); // read the next byte as a char
            Serial.print(c);      // print the char
      }
      int x = Wire.read();       // read the next byte as an int
      Serial.println(x);         // print the int in a new line
}
```

Program the application

Provide your program listing.

Structure the program modularly clearly segmenting the program in separate code blocks – “basic blocks” –, one for each function to be implemented (e. g. read temperature sensor, control temperature alarm), even if you feel such a program overstructured. In the second part of the work (question 8) you will have to quickly redistribute several functions across different 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.

Instituto Superior Técnico

Applications and Computation for the Internet of Things

2022-2023

2nd Lab work: Network of Sensors and Actuators

Group:		
Student 1:		
Student 2:		
Student 3:		

Results

Fill the following fields or provide the corresponding printed listings.

1. Describe the structure of the programs developed for both controllers.
    a. Describe the design pattern (such as tasks/ modules / basic blocks in the round-robin loop).

    b. Describe other implementation issues (such as implementation of tasks) you consider relevant for the overall app.

2. Controller #1 program – I2C master:

```
void setup() {


}

void loop() {


}
```

3. Controller #2 program – I2C slave:

```
void setup() {


}

void loop() {



}
```

4. For each of the three pairs sensor-actuator describe:
    a. the mapping process implemented;
    b. the calibration setup;
    c. the process, or technique, used to modulate the behavior of the actuator;
    d. the setup prepared to demonstrate the functionality of the system.
5. What are the timing constraints of the system?
6. Evaluate the performance of the I2C bus in your system (data rate, latency).
7. Mount the light sensor and the corresponding LED face to face so that the sensor detects the light emitted by “its” LED. (To reduce the influence of other light sources you can place the sensor and the LED on both extremes of a paper tube, face to face.)
   Do you see any changes in the behavior of the LED? (Analyze the correspondent control loop.)
   If so explain them.
   If not explain why.
8. To demonstrate the modularity of your programs set-up and run the following configuration (to be specified by the teacher at the demo):
    - Controller 1 (original “master”): RED / GREEN / YELLOW LED (actuator);
    - Controller 2 (original “slave”): corresponding sensor.