

STEMKIT
4SCHOOLS

USING IR SENSORS IN ALARM SYSTEMS

LESSON PLAN 1



Co-funded by the
Erasmus+ Programme
of the European Union

This project has been funded with support from the European Commission.

This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Table of Contents

1. Using IR sensors in alarm systems	2
1.1 General information	2
1.1.1 Short description	2
1.1.2 Learning objectives	2
1.1.3 Links to curriculum	2
1.1.4 Materials required	3
1.1.5 Duration	3
1.2 Lesson plan	4
1.2.1 Introduction	4
1.2.2 Preparation	4
1.2.3 Investigation.....	9
1.2.4 Conclusion	10
1.2.5 Follow-up exercise (optional)	10
1.3 References or Resources	10



1. Using IR sensors in alarm systems

1.1 General information

1.1.1 Short description

This lesson introduces the use of TCRT5000 reflective optical sensor with transistor output to design a simple circuit that will act as an alarm for windows that might have been opened by an unauthorised person. The positioning of such a sensor can be introduced in real set up next to the window frame due to operating range of the sensor that is from 0.2 mm to 15 mm.

1.1.2 Learning objectives

The main learning objectives of this lesson plan or educational activity are:

- Concept and content understanding of hardware to set up a code in Scratch.
- Familiarising with hands on console building activities to enhance experimentation in STEM related subjects.
- Familiarizing with readings from GPIO pins.
- Understanding the coding steps in Scratch.
- Designing coding in Scratch.
- Using sensors, infrared emitter, phototransistor and other elements to experiment with the STEMKIT console.
- Performing coding on a STEMKIT console on the example of IR sensors in alarm systems.
- Performing basic assembly of sensors on a breadboard.
- Experimenting with connecting sensors to the Raspberry Pi and STEMKIT console.
- Autonomy in the creation of a simple circuit that can serve as a demonstration of a security installation in a building.
- Autonomy in introducing the concepts of coding in the classroom environment.

1.1.3 Links to curriculum

The domains, subdomains, subjects/topics that this lesson plan can be linked to are:



Erasmus+

2019-1-FR01-KA201-062281



STEMKIT
4SCHOOLS

Science (Physics/Chemistry/Biology/Geology): voltage, power, circuits, alarm triggers, scientific method, investigation, experimentation, analysis and interpretation of results

Computer Science/Informatics: processing unit and peripherals, interfaces, programming language and main structures, coding

Technology: electronics, open-source hardware and software, sensors, digital signal, single board computers, console

1.1.4 Materials required

In order to carry out this lesson plan, the STEMKIT console with Raspberry Pi is needed along with the following elements:

- 3 x TCRT5000 sensors
- 1 x buzzer with generator
- 2 x Female-to-Female jumper wires
- 5 x Male-to-Female jumper wires
- 3 x 10k Ω resistors
- 3 x 330 Ω resistors
- 1 x breadboard

1.1.5 Duration

The duration of this lesson plan is estimated to be about 45-60 mins, i.e., one classroom hour.



1.2 Lesson plan

The lesson plan is divided in four phases, which are introduction, preparation, investigation and conclusion. As a follow-up there is also an optional exercise at the end.

1.2.1 Introduction

The TCRT5000 is a sensor that is composed of two elements in one enclosure. The emitter emits the wavelength of 950 nm that is then to be received by the phototransistor working as a detector. The closer the object to the sensor is, the greater reading of the voltage from the phototransistor is going to be. As Raspberry Pi offers GPIO pins, this reading can then be followed to check whether the obstacle (the window frame) is close to the sensor and as a result if the window is closed.

Within this lesson Scratch will be used to demonstrate the sample code that can be used to track this simple circuit.

1.2.2 Preparation

The preparation phase requires to perform a basic assembly of the sensors on a breadboard and setting up the code in Scratch. Let us start with the breadboard first.

Place three TCRT5000 sensors on an empty breadboard by connecting the emitter of the phototransistor and cathode of the infrared emitter to the ground rail. After that, connect the collector of the phototransistor using a 10k Ω resistor to the positive rail on a breadboard. Likewise, connect the anode of the infrared emitter to the positive rail using a 330 Ω resistor. Repeat it for the remaining two TCRT5000 sensors.

Now it is time to connect the sensors to the Raspberry Pi. Connect the jumper wire of an appropriate length to each TCRT5000 sensor. It should connect to the rail where the collector of the phototransistor is connected and powered by a positive voltage rail using a 10k Ω resistor. At this stage there should be three jumper wires – one per each TCRT5000 sensor. These jumper wires should be connected to GPIO pins on Raspberry Pi marked as 35, 33 and 31 (or in GPIO: 19, 13 and 6). Now it is also the time to connect the buzzer directly to Raspberry Pi. You can do so by attaching the GND pin of the buzzer to pin 39 and its positive wire to pin 38 (or GPIO: 26). Finally, the breadboard needs to receive the power from Raspberry Pi. For this purpose, you can use +5V rail from Raspberry Pi (pin 4) and GND (pin 6). Use jumper wires to power up the breadboard. The hardware setup is done, so now we can move to Scratch.



Inside Scratch, you can select any backdrop that has at least three windows visible. For this lesson we are going to use urban1 backdrop. Set it for the entire scene. At the same time, you need to add three objects that will change their costumes once the alarm is triggered. Here the suggestion is to use a sprite that combines button4-a and button5-b. The code will change the costumes based on the reading from the sensors. As you need to have three sprites (one per each window), please duplicate them to have the following setup (the middle sprite has the other costume active to show the difference):



Image 1. Scratch environment with all required elements positioned on the screen
Source: STEMKIT4Schools project

Before we go to the main code, let us work on the three sprites that are placed on the windows.

To simulate the alarm, we are going to use Scratch's broadcasting messages to react accordingly to the readings from the sensors. When you switch over to the *Scripts* tab, you will need to add two reactions for the incoming messages. Let us assume that for the first sensor we will emit messages *window1-open* and *window1-closed*. When the message received is *window1-open*, then we need to trigger the alarm and change the costume from a green checkmark to a red cross. Similarly, when the window is closed, we need to have the green checkmark costume again. The sample code is presented below:



Image 2. Script for sprites reacting to the readings from GPIO pins
Source: STEMKIT4Schools project

Replicate this setup for the remaining two sprites placed over the windows and remember to change the titles of the messages (*window2-open*, *window2-closed*, *window3-open*, *window3-closed*).

The main code is where it all gets interesting. Once you select the cat, switch over to the Scripts tab and start adding the code. The first thing we would like to set up is let Scratch know that GPIO pins 19, 13 and 6 should be read as input pins.



Image 3. Beginning of the code – setting relevant GPIO pins as input ones
Source: STEMKIT4Schools project

Next, you can add some messages before we launch the main code.

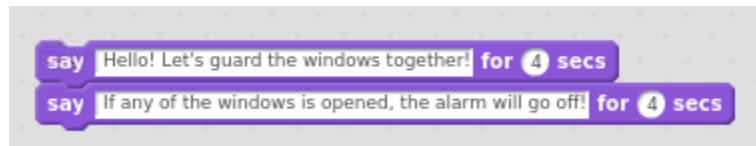


Image 4. Introduction messages executed by the code
Source: STEMKIT4Schools project

In the next step, we are going to add a *repeat* loop that will do 30 cycles, understood as “checks” of the sensors.



Image 5. Main repeat loop

Source: STEMKIT4Schools project

Within each loop, we are interested to reach all three sensors to see if the windows are closed or not. The structure is very simple, as we can check with an *if* statement if the reading from respective GPIO pins is high and act accordingly. Below you will find an example for GPIO pin number 19.



Image 6. An if statement reading the GPIO value and broadcasting messages

Source: STEMKIT4Schools project

Note that the only action we are taking here is to emit either window1-open or window1-closed message based on the value of GPIO pin number 19. Similar *if* blocks should be added for the remaining GPIO: 13 and 6. Again, do not forget to change the number of GPIO pin and also the message to be broadcasted!

In the next step we also want to control our buzzer. The condition here is that if all windows are closed, the alarm is silent. When at least one window is opened, we will emit the alarm sound and will also let the cat say *Alarm!* to have also a visual notice. For this, we will use an *if* statement with three conditions combined by a logical *or* instruction.



Image 7. An if statement to trigger the alarm if needed
Source: STEMKIT4Schools project

Just before completing the current loop, we will pause the execution for 1 second. Doing so defines the frequency of our checks to be executed once per each second.

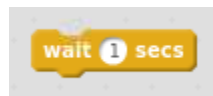


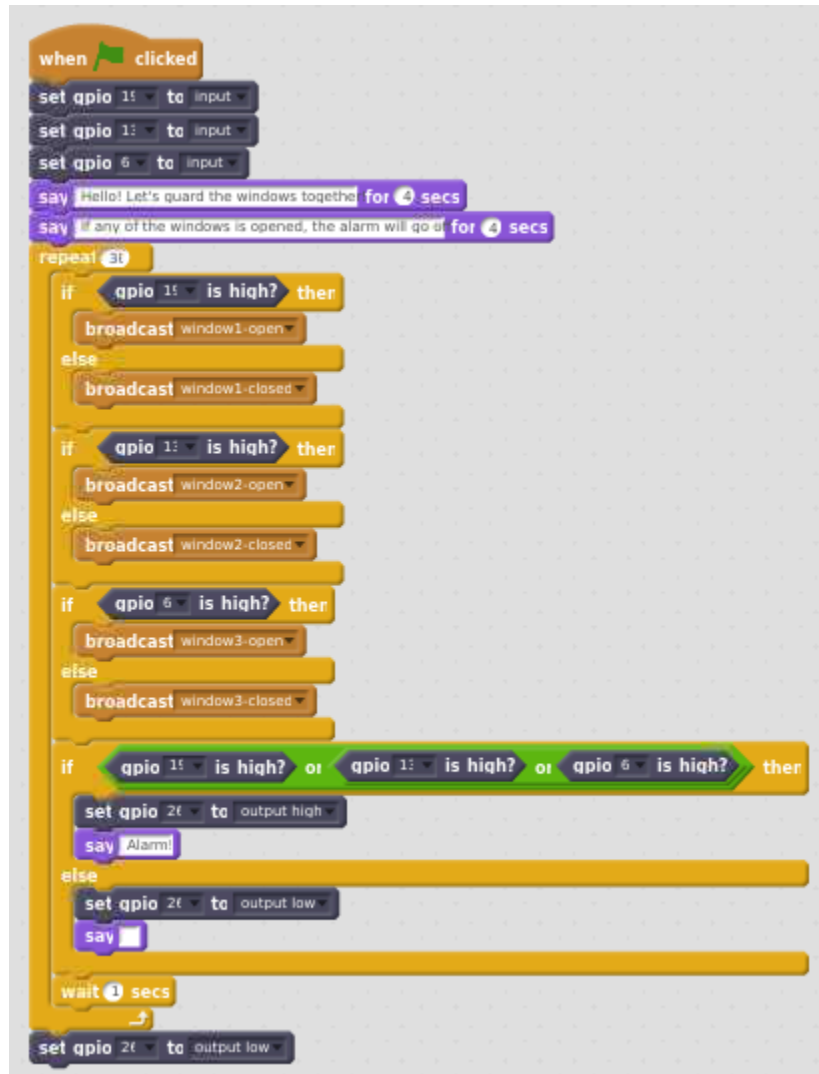
Image 8. Pause execution of the code for 1 second just before leaving the loop
Source: STEMKIT4Schools project

Finally, as we do not want to have the buzzer emitting sound after we finish all our code, right after the *repeat* loop we need to add the instruction to set the GPIO pin number 26 (the one with the buzzer) to remain in the *low* state (which means no sound).



Image 9. Setting buzzer's GPIO to low after the main loop
Source: STEMKIT4Schools project

The final result defining our code is presented below.



```

when green flag clicked
  set gpio 15 to input
  set gpio 13 to input
  set gpio 6 to input
  say Hello! Let's guard the windows together! for 4 secs
  say If any of the windows is opened, the alarm will go off for 4 secs
  repeat (3)
    if gpio 15 is high? then
      broadcast window1-open
    else
      broadcast window1-closed
    if gpio 13 is high? then
      broadcast window2-open
    else
      broadcast window2-closed
    if gpio 6 is high? then
      broadcast window3-open
    else
      broadcast window3-closed
    if gpio 15 is high? or gpio 13 is high? or gpio 6 is high? then
      set gpio 24 to output high
      say Alarm!
    else
      set gpio 24 to output low
      say 
    wait 2 secs
  set gpio 24 to output low
  
```

Image 10. Main code for the sample circuit
Source: STEMKIT4Schools project

1.2.3 Investigation

We can finally run our code! Please follow the next subtasks to learn more about this example. To avoid false readings, make sure that all the elements used to build the circuit (jumper wires or resistors) do not cover any of the TCRT5000 sensors.

Collection of data

For the first run, make sure that the sensors are not covered, or, in other words, that there are no elements that would reflect the emitted infrared beam and therefore turn the GPIO



pins into *high* state. Within the next run, try to cover one sensor with your finger or any other element. Try to observe the behaviour when all three sensors are covered, simulating the situation where all windows are closed.

Analysis of data

Based on the data observed, are you able to tell if the distance of the obstacle from the sensor is in line with the expected working range? Try to notice how close the obstacle needs to be above the sensor to make it turn the GPIO pin into *high* state. Is the behaviour of the code consistent with the expectations? Do the sprites on the screen change in response to specific sensors being covered/uncovered? Does the alarm go off when at least one window is opened and does the cat say *Alarm!* in this situation?

Presentation of results

At this stage we are invited to share the results of our work with other groups. Has everything worked fine? Were there any difficulties in setting up the entire circuit? Were there any changes introduced to the code? If so, what kind of? Were the readings on how close the obstacle needs to be to turn the sensor's GPIO port into *high* state consistent for all the sensors? Was it also the same case in other groups?

1.2.4 Conclusion

We have succeeded in creating a very simple circuit that can serve as a demonstration of a security installation in the building. At this stage we can exchange ideas with other groups, what was done in which way and in what order and to clarify any questions that might appear.

1.2.5 Follow-up exercise (optional)

The follow-up exercise can include a multimeter, measuring the voltage from the sensors in live mode as the obstacle is approaching. This way we will be able to tell at what voltage the Raspberry Pi considers the input to be in the *high* state. We can also try to change the main loop used in the code from *repeat* to *forever*.

1.3 References or Resources

Resource related to this lesson plan: <https://www.vishay.com/docs/83760/tcrt5000.pdf>