WPI

RBE 1001: Introduction to Robotics
C-Term 2019-20
HW 5.2: Alarm system as a state machine

Last modification: February 23, 2020

This is a team assignment. **You must complete the assignment as a team and you must all work together**. However, you may not begin to work on this assignment until each team member has submitted HW5.1.

Home alarm system

Your challenge is to build and code a minimalist home alarm system. The system will need to allow activation and deactivation; sense an intruder; and sound an alarm if someone is detected. For each of these functions, you'll use the following hardware and methods:

Activation. To activate your system, you will merely press a button. When activated, your system will turn a servo motor (to represent locking a door) and turn on an LED that is pointed towards a photoresistor, which will be used for detecting an intruder – a low-tech version of the laser systems seen in the movies.

Deactivation. Normally, deactivation would require a code or a key or similar. Due to time constraints, however, you'll deactivate your alarm using a second button. Pretend it's well hidden! When deactivated, the "laser" will turn off and the servo will move back to the "unlocked" position.

Intrusion detection. When the system is activated, if something comes between the LED and the photoresistor, occluding the laser, the system will alarm.

Alarming. Alarming will consist a piezo buzzer that blares at 3800 Hz. You must be able to deactivate the system with the button while it's alarming.

A skeleton code will be provided that takes care of a lot of the tedious parts – we want you to focus on implementation of the state machine.

Procedure

1. Consulting the LED/photoresisor circuits in Figure 2, do the following:

   (a) Insert the photoresistor and LED into your breadboard about 3 - 4 cm apart. Gently bend the two so that the LED will shine directly on the photoresistor element – they should be about 1 cm from each other when you're all done, as shown in Figure 1. You'll want them close enough to get a good signal while still being able to pass a pencil or a finger between them without bumping either. Once you've arranged them, try not to move either component so you will get consistent results.

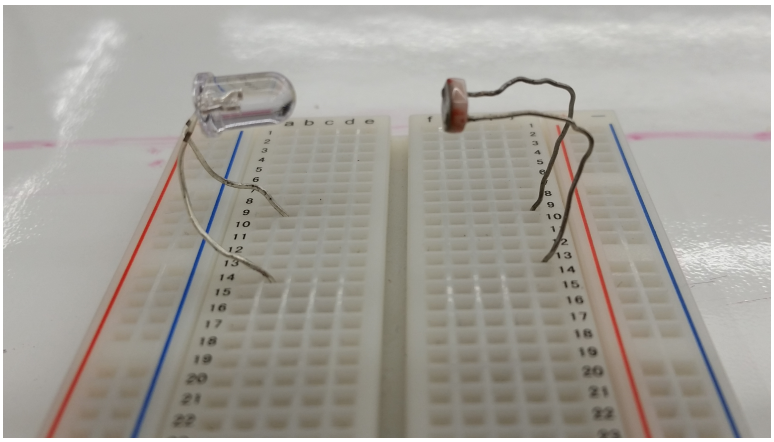

Figure 1: “Laser” setup in your breadboard. Note how the two components are bent to point towards each other. If you have a clear LED, that will work best, but a yellow one also works.

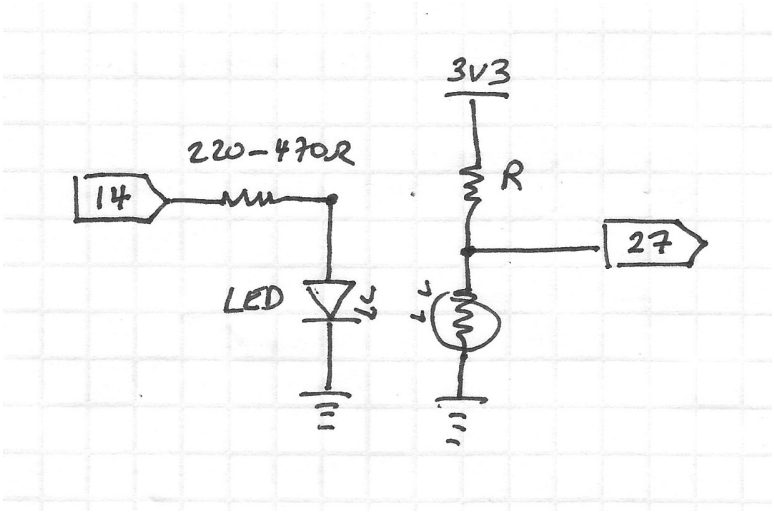

Figure 2: Circuits for the laser. The resistor for the LED can be anywhere between 220Ω and 470Ω, but lower resistance will give you better results (why is that?).

(b) Connect the LED as shown. In a previous exercise, you found the resistance of the photoresistor in both light and dark. Using those values, choose a resistor for the other leg of the voltage divider. Connect the junction of the voltage divider to pin 27.

(c) Write a program to read the ADC and print the result to the Serial Monitor every 500 ms. Run some tests to determine the ADC reading with the LED on and with it blocked. You may want to shield the photoresistor from ambient light to make it more effective. Choose a numeric threshold that will serve to indicate that the beam is blocked – that there is an intruder! Record your values below.

ADC reading (LED on): _____

ADC reading (LED off): _____

Threshold ADC value: _____

2. Build two button circuits, as shown in Figure 3. To save space, we'll use a nice feature of the ESP32, namely internal pullups. To do so, you declare the pin mode as follows:

```
pinMode(buttonPin, INPUT_PULLUP);
```

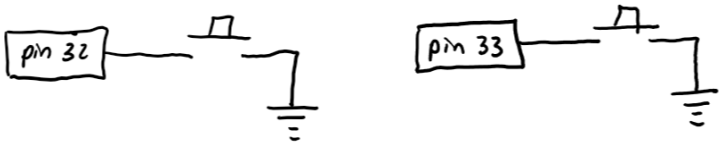

Figure 3: Circuits for the buttons using internal pullups.

This internal pullup has the same effect as the resistor you put in your breadboard, but without the hassle of extra wiring.

3. Build the piezo buzzer circuit in Figure 4.

4. Add the servo motor to the system on pin 25. You used the servo in a previous homework; consult that if you need a refresher on how to command it.

5. Grab the `alarm_skeleton.ino` code from canvas. Because of time constraints, we're not asking you to write code from scratch.

6. Now you will fill in the missing parts of the state machine in the rest of the code. It may be helpful to put some `Serial.print()` statements to help debug. Specifically, every time there is a transition, you should print something to the Serial Monitor. See `HandleArmingButton()` in the code for an example.

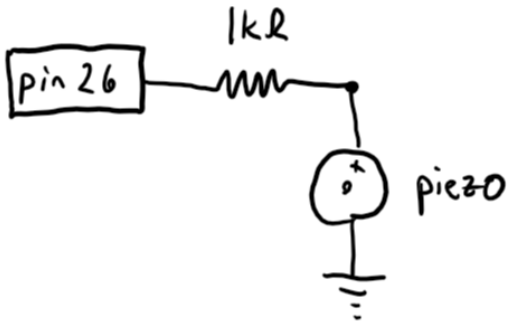

Figure 4: Circuit for the piezo "buzzer."

(a) We've written the button checker function for you, but you need to finish the function to checking if the laser beam is broken using the threshold you found above. Note that you need to write it in **proper event-driven form** – do not just check the state! Use the button checker as a template.

(b) We've started the button handler for you, but you'll have to fill in the actions. Note that we only explicitly address states that are specific to that event.

(c) You'll have to fill in the broken laser beam handler function. Don't forget to change the state when an event is handled! If you've included the `ESP32Servo.h` library, then you have access to the `tone` function:

- `tone(<pin>, <freq>)` will start a tone on the designated pin at the given frequency.
- `noTone(<pin>)` will turn it off.

(d) You'll also have to fill in the deactivation handler. How many states does this handler apply to?

7. Test your system. If it's not working, the first thing you'll want to do is determine if the problem is in hardware or software. To check if it's hardware, use known, working codes to test the components. You can also use a digital multimeter to check voltages, for example, at a pin. If you suspect that software is the problem, try putting `Serial.print()` statements in useful places to help determine if the code is getting to where it should be.

   Of course, you can always ask a TA/SA/instructor for some pointers.

Once you have a working system, demonstrate it to an SA or TA and then submit the code to canvas. The TA/SA has a sign-off sheet to note that you've completed the assignment.