Arduino Motion Alarm System

n3oxzz

Physics 11

June 18, 2025

Introduction

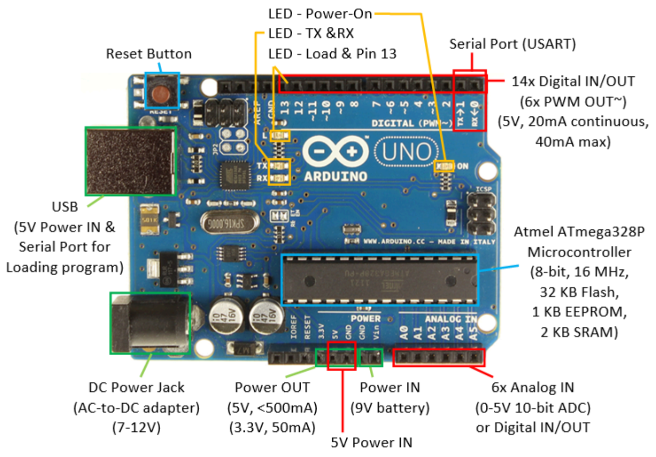
Most people experience frustration when someone enters their room without knocking or announcing themselves. For many, a personal room is a private space, and crossing this boundary, especially with no permission, often irritates. As someone who finds this issue to be especially bothersome (even though I don't have younger siblings), I found a way of dealing with it. I decided to design a simple device that could at least alert me when someone enters the room, and came up with the idea of creating the motion alarm system.

While the motion system is neither new nor groundbreaking, it usually proves itself practical with applications in various fields. Motion trackers are used in automating lights, home alarm systems, and even security systems in restricted areas. The combination of motion tracking and alarm systems allows even more versatility and application in other fields. The goal of this project is to utilize the Arduino microcontroller and its software, along with the 6-axis Inertial Measurement Unit MPU-6050, which incorporates both a gyroscope and accelerometer, and apply my knowledge of physics, math, and software development in creating a motion alarm system.

Background Information

What is Arduino?

According to the official Arduino website, Arduino is an open-source electronics platform based on easy-to-use hardware and software. Essentially, it is a small computer or microcontroller that can be programmed to perform a variety of tasks, such as turning on LEDs, reading data from sensors, and using motors to create the movement of an object.


LED - Power-On
LED - TX &RX
LED - Load & Pin 13
Reset Button
Serial Port (USART)
14x Digital IN/OUT (6x PWM OUT~) (5V, 20mA continuous, 40mA max)
USB (5V Power IN & Serial Port for Loading program)
Atmel ATmega328P Microcontroller (8-bit, 16 MHz, 32 KB Flash, 1 KB EEPROM, 2 KB SRAM)
DC Power Jack (AC-to-DC adapter) (7-12V)
Power OUT (5V, <500mA) (3.3V, 50mA)
Power IN (9V battery)
5V Power IN
6x Analog IN (0-5V 10-bit ADC) or Digital IN/OUT

The image above provides a diagram of the key components of the Arduino UNO R3 board. Elements to look out for:

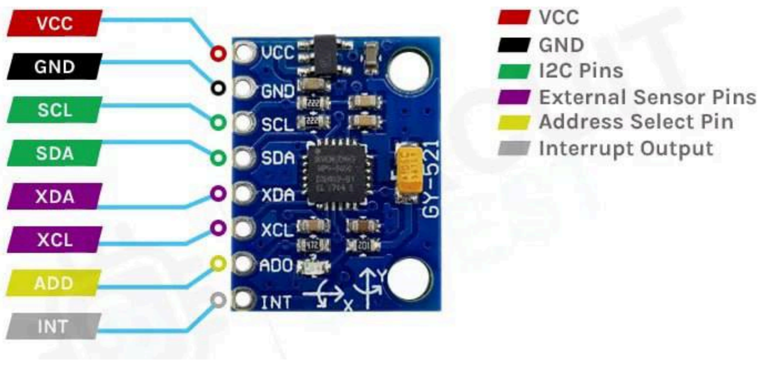
- The power jacks allow the board to be connected to a PC, laptop, or any other power supply via a USB cable or through the DC Power Jack, which utilizes a battery pack as its power source.
- Power out pins — 5V and 3.3V — allow the delivery of power to the external circuit.
- Digital IN/OUT pins allow to write digital data — discrete data that represents distinct values, often in a binary form (Wikipedia contributors, 2025), for example, connecting

the LED to the one of the pins would enable the board to “communicate” with the LED and therefore let it turn on and off; read digital data, e.g., detecting the state of the button (i.e., pressed or not pressed).

- Analog IN pins are primarily used for reading analog data — information with a continuous range of values, such as the hands of a clock or a sound wave (Wikipedia contributors, 2025). In this situation, signals like voltage levels from sensors are used. They are designed to convert this continuous voltage signal into a digital value, allowing a microcontroller to process the information.

What is MPU-6050?

The sensor used in this project is the MPU-6050. This sensor is an IMU, which stands for Inertial Measurement Unit, a device used to measure and report an object’s acceleration, angular rate, and sometimes orientation. Some examples of IMUs are gyroscopes, accelerometers, and magnometers. The MPU-6050 incorporates a 3-axis accelerometer that operates on the piezoelectric principle — the ability of certain materials to generate an electric charge in response to applied mechanical stress. The generated current can be measured and then used to identify how strongly and in what direction the sensor accelerates or tilts (Bera & Sarkar, 2016). It also includes a 3-axis gyroscope that works on the principle of Coriolis acceleration, which occurs when the moving internal part of the device experiences lateral displacement while turning. This displacement creates a force that pushes on the piezoelectric crystals, holding the structure. In response, these crystals produce an electric current, which can be used to determine the sensor’s angular velocity (Coriolis acceleration, n.d.) Some use it in their DIY projects, such as balancing robots, multicopters, and game controllers.



The image above is a diagram of the MPU-6050 sensor. Out of the eight pins on it, this project only requires the use of four: VCC, GND, SCL, and SDA. VCC pin is providing power to the sensor, it’s connected to the 5V power out pin on the microcontroller, GND is the ground pin, which provides a return path for electrical current, allowing a circuit to function properly and safely, SCL (Serial Clock Line) and SDA (Serial Data Line) are the parts of the I2C communication system. I2C, or I^2C is the Inter-Integrated Circuit, it is a two-wire serial communication protocol that enables digital integrated circuits (ICs) to communicate with each other (Wikipedia contributors, 2025). The simple way to interpret this is as a way for electronic devices to communicate with each other, using only two wires. For instance, in this project, the MPU-6050 transfers data in and out of the Arduino board using the SDA pin, while SCL tells both sides when to send or read each bit.

Noises

Noises in data are the data that was corrupted or distorted; it is also sometimes considered to be irrelevant or unnecessary (Wikipedia contributors, 2025). It is an inherent part of data; however, one of the most significant challenges for data scientists. Noisy data can originate from various sources, but most commonly it appears due to measurement errors, mistakes in calculations, and external factors (Gomede E, 2023). In our particular case, we sometimes receive noisy data from the sensor. Due to its low-cost MEMS design, the sensor is inherently imprecise and sometimes too sensitive, so it is prone to noise. However, there is a way to improve this data or in other words — filter.

Filtering

Data filtering is the process of refining a dataset by removing the noisy information, which allows for focusing on specific data points for analysis (Cribl, 2024).

In this project, I was considering among these three methods of filtering: low-pass filtering, complementary filtering, and Kalman filtering.
The low-pass method is the easiest of the three, and essentially, it works by blending the previously measured angle and the new one using a simple formula to smooth out sudden jumps or noise. The idea is to trust the previous value more and slowly adjust toward the new one. The formula used is $\theta_{filtered} = x \cdot \theta_{previous} + (1 - x) \cdot \theta_{current}$

Here, x represents a trust factor or the weight of the value before.

The complementary method of filtering data is slightly more challenging than the previous one, although the concept behind it is straightforward. Instead of just using data from the gyroscope, we could also use the value from the accelerometer. The gyroscope on its own provides smooth values; however, it gets worse over time. On the other hand, an accelerometer provides rougher data but always gives an angle close to reality. What we could do with the complementary filtering is to trust the gyroscope in the short term, but trust the accelerometer in the long term. The formula used there is $\theta_{filtered} = x \cdot (\theta_{prev} + gyro \cdot dt) + (1 - x) * \theta_{accel}$

Here, x is again the trust factor to the gyroscope, and the $gyro \cdot dt$ is the angle calculated from the gyroscope (derived from the formula for the angular velocity $\omega = \frac{\Delta\theta}{dt}$)

The Kalman method is considered the most challenging on my list due to its concept. The idea is that the model predicts the resulting value based on physics and the previous value, and then compares it with the actual values. Later, the model corrects its value with an account for the possibility of the error. It has not just one formula, but a system that includes matrices, covariance, statistical weight, etc. I could have used the built-in library that already has all the calculations for the Kalman filtering method and tried to explain how it works; however, I felt it would be less fair and less engaging. This project can later be improved using this method, once I gain more knowledge.

That is why my choice is the happy medium — the complementary filtering method.

Materials

1x Arduino UNO R3 microcontroller (I am using a replica made by the company ELEGOO)

1x MPU6050 sensor

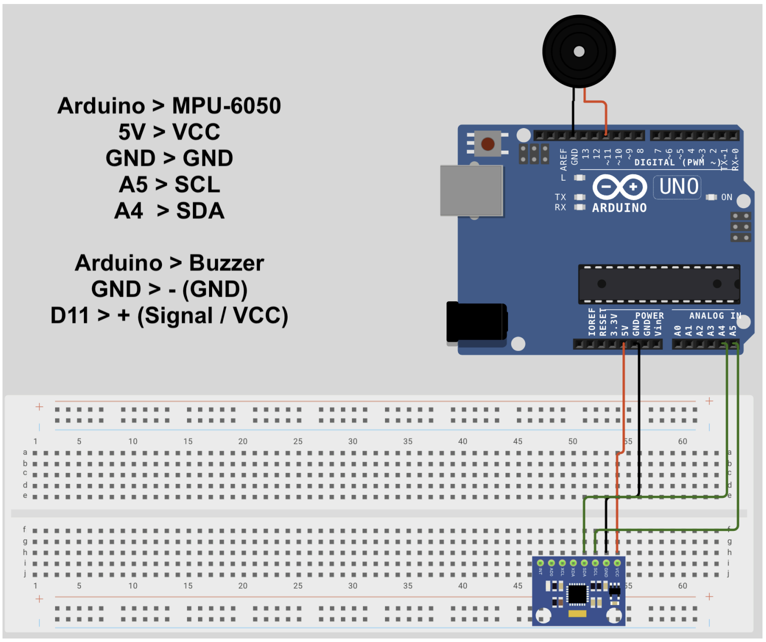
1x Arduino breadboard (the solderless construction base, used for developing electronic circuits and wiring)

1x Active buzzer (the source of sound in the project)

4x Breadboard Jumper Wires

Methods

First, we need to create the circuit and write the software in the simulation. I am using Wokwi to do that. On this website, I made a circuit connection diagram on the right and then used the MPU6050_light library by rfetick to write software that collects data from the sensor. After that, I had to filter the data, as I have already mentioned, using the complementary filter. The code I use for it:



```
float dt = (currentTime - prevTime) / 1000.0;
gyroAngleX += gyroX * dt;
float phi = atan2(accel[1], accel[2]);
float finalAngle = (weight * gyroAngleX) + ((1-weight) * phi);
```

In this part of the code:

- dt is the time between the previous gyroscope measurement and the current one.
- $gyroAngleX$ is the sum of all the rotations given by the gyroscope over time. Essentially, it is the value of an angle according to the gyroscope.
- phi is another angle now from the accelerometer, it is an angle between the gravitational vector (z-axis) and the plane (y-axis).
- $finalAngle$ is the filtered angle using the complementary filtering.

The code was almost done at this point — the only thing left was to make the buzzer produce the sound when the movement is detected.

I used this snippet to do that:

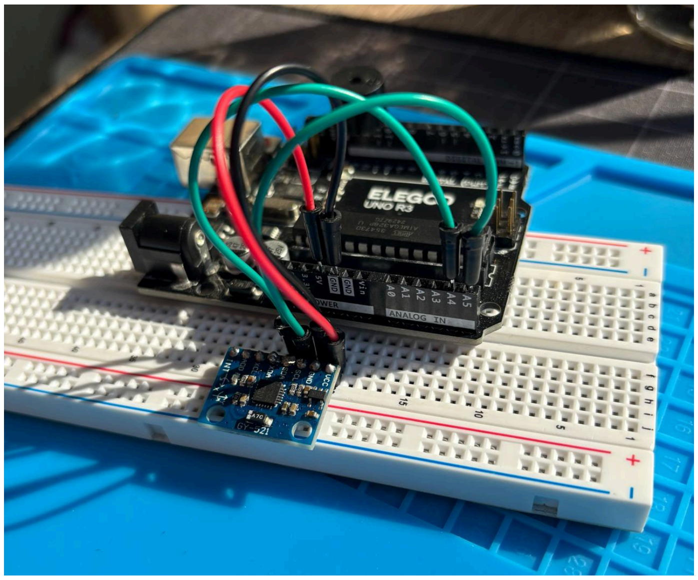
```
if (abs(finalAngle) > threshold) {
  if(!isAlarmOn){
   digitalWrite(buzzerPin, HIGH);
   isAlarmOn = true;
  }else{
   if(isAlarmOn){
     digitalWrite(buzzerPin, LOW);
     isAlarmOn = false;
}}}
```

Here, we check if the absolute value of the final angle exceeds a certain threshold (set to 5°) If it does, that means the sensor has detected noticeable movement. In response, current flows to the buzzer, causing it to produce sound. This check happens every loop (i.e., every 10 milliseconds), and when the value of the final angle turns smaller than the threshold, the buzzer turns off.

Now the code was done. Before moving the circuit onto an actual Arduino circuit board, I had to solder the MPU6050 sensor to the pin headers. Next, I built the circuit on the Arduino board and transferred the software to the Arduino IDE. Lastly, I had to test my work.

Testing and Results

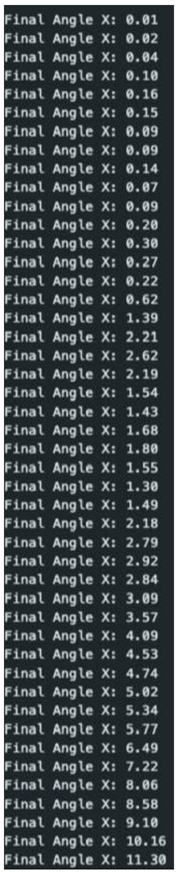
The soldered sensor connected to an Arduino circuit board is shown in the picture.



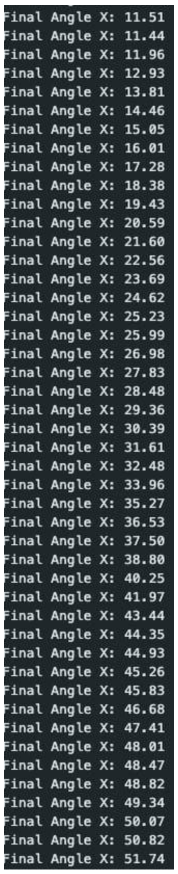
The next thing was to check whether it works. I connected the whole system to the door using tape and tested it. The results turned out to be better than expected. The system worked pretty well, without abrupt changes in values. The only problem was that sometimes, when you reach an angle that is too high compared to the threshold, the buzzer could become unstoppable, and the only way to turn it off was to unplug the Arduino from the port.

Analysis

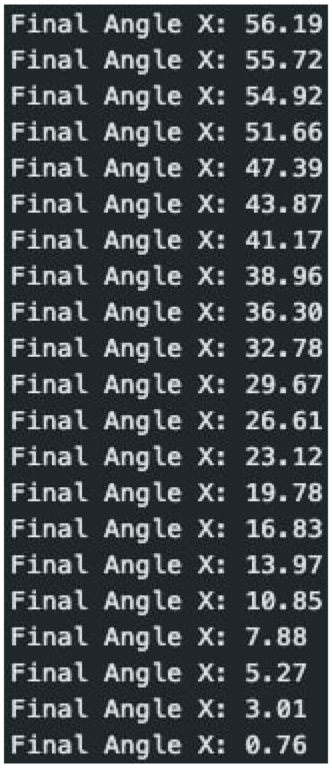
After the testing, I collected the logs I had. These were the values of the final angle every 10 ms. When the angle reached the value of 5°, the buzzer turned on, then the door kept moving until approximately 56°, and then went back to 0°. We can see that the values are increasing and decreasing consistently, which means that the filtering method we used works — otherwise the values would have been heavily oscillating, and would not be consistent.



```
Final Angle X: 0.01
Final Angle X: 0.02
Final Angle X: 0.04
Final Angle X: 0.10
Final Angle X: 0.16
Final Angle X: 0.15
Final Angle X: 0.09
Final Angle X: 0.09
Final Angle X: 0.14
Final Angle X: 0.07
Final Angle X: 0.09
Final Angle X: 0.20
Final Angle X: 0.30
Final Angle X: 0.27
Final Angle X: 0.22
Final Angle X: 0.62
Final Angle X: 1.39
Final Angle X: 2.21
Final Angle X: 2.62
Final Angle X: 2.19
Final Angle X: 1.54
Final Angle X: 1.43
Final Angle X: 1.68
Final Angle X: 1.80
Final Angle X: 1.55
Final Angle X: 1.30
Final Angle X: 1.49
Final Angle X: 2.18
Final Angle X: 2.79
Final Angle X: 2.92
Final Angle X: 2.84
Final Angle X: 3.09
Final Angle X: 3.57
Final Angle X: 4.09
Final Angle X: 4.53
Final Angle X: 4.74
Final Angle X: 5.02
Final Angle X: 5.34
Final Angle X: 5.77
Final Angle X: 6.49
Final Angle X: 7.22
Final Angle X: 8.06
Final Angle X: 8.58
Final Angle X: 9.10
Final Angle X: 10.16
Final Angle X: 11.30
```



```
inal Angle X: 11.51
inal Angle X: 11.44
inal Angle X: 11.96
inal Angle X: 12.93
inal Angle X: 13.81
inal Angle X: 14.46
inal Angle X: 15.05
inal Angle X: 16.01
inal Angle X: 17.28
inal Angle X: 18.38
inal Angle X: 19.43
inal Angle X: 20.59
inal Angle X: 21.60
inal Angle X: 22.56
inal Angle X: 23.69
inal Angle X: 24.62
inal Angle X: 25.23
inal Angle X: 25.99
inal Angle X: 26.98
inal Angle X: 27.83
inal Angle X: 28.48
inal Angle X: 29.36
inal Angle X: 30.39
inal Angle X: 31.61
inal Angle X: 32.48
inal Angle X: 33.96
inal Angle X: 35.27
inal Angle X: 36.53
inal Angle X: 37.50
inal Angle X: 38.80
inal Angle X: 40.25
inal Angle X: 41.97
inal Angle X: 43.44
inal Angle X: 44.35
inal Angle X: 44.93
inal Angle X: 45.26
inal Angle X: 45.83
inal Angle X: 46.68
inal Angle X: 47.41
inal Angle X: 48.01
inal Angle X: 48.47
inal Angle X: 48.82
inal Angle X: 49.34
inal Angle X: 50.07
inal Angle X: 50.82
inal Angle X: 51.74
```



```
Final Angle X: 56.19
Final Angle X: 55.72
Final Angle X: 54.92
Final Angle X: 51.66
Final Angle X: 47.39
Final Angle X: 43.87
Final Angle X: 41.17
Final Angle X: 38.96
Final Angle X: 36.30
Final Angle X: 32.78
Final Angle X: 29.67
Final Angle X: 26.61
Final Angle X: 23.12
Final Angle X: 19.78
Final Angle X: 16.83
Final Angle X: 13.97
Final Angle X: 10.85
Final Angle X: 7.88
Final Angle X: 5.27
Final Angle X: 3.01
Final Angle X: 0.76
```

Conclusion

In this project, I have successfully developed a motion alarm system using Arduino and the MPU6050 sensor. It can detect the door motion when opening and sound the alert when a certain angle threshold is reached. The complimentary filtering method used in this project avoided sharp fluctuations in data, making it smoother. While testing, the system has proved its operational reliability; however, with certain limits, such as the impossibility to turn off the signal, when the angle reached is too high.

Future of the project

In the future, a couple of new features could be incorporated, such as remote operation — right now, the Arduino must always be connected to the laptop to transfer the code from it. I could use Bluetooth modules like HC-05, or the Wi-Fi-supported microcontrollers like ESP32. In addition to this improvement, the usage of breadboard in the main system could be eliminated by soldering the sensor directly to wires, making the system more compact. The next major improvement to this project could be the incorporation of push notifications; this would be a great software improvement if I could make the sensor send messages directly to my phone whenever the door is opened. The next equally important improvement to this project could be the use of the Kalman filtering method, which would allow even more accurate data. Lastly, the use of a more reliable sensor like the ICM-20948—the direct successor of the MPU6050 used in this project—or the MPU-9150, which, in addition to the 3-axis accelerometer and gyroscope, has a 3-axis magnetometer, which measures the strength of the magnetic field along the 3 axes, effectively providing a heading reference; that makes this sensor a 9-degree-of-freedom sensor and allows it to determine an absolute position in space (often used for 3D orientation).

References

Arduino. (n.d.). Introduction [Getting Started Guide]. In Arduino – Getting Started. https://www.arduino.cc/en/Guide/Introduction/

Wikipedia contributors. (2025, June 12). Digital data. In *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Digital_data&oldid=1295279949

Wikipedia contributors. (2025, May 19). Analog signal. In *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Analog_signal&oldid=1291213798

Bera, B., & Sarkar, M. D. (2016). Piezoelectric effect, piezotronics and piezophototronics: a review. *Imperial Journal of Interdisciplinary Research (IJIR)*, *2*(11), 1407-1410.

Coriolis acceleration. (n.d.). *ScienceDirect*. https://www.sciencedirect.com/topics/engineering/coriolis-acceleration

Science 4 U. (2024, June 29). Self Balancing Robot using Arduino & MPU 6050 || Step by step Tutorial [Video]. YouTube. https://www.youtube.com/watch?v=pbJbpHjC3v0

Joop Brokking. (2015, April 2). Your Multicopter Flight Controller 3D [YouTube playlist]. YouTube. https://www.youtube.com/playlist?list=PL0K4VDicBzsibZqfa42DVxC8CGCMB7G2G

Wikipedia contributors. (2025, June 5). I²C. In *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=I%C2%B2C&oldid=1294054037

Wikipedia contributors. (2025, May 6). Noisy data. In *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Noisy_data&oldid=1289168414

Gomede, E. (2023, September 27). *The significance of noise and outliers in data science*.

Medium.
https://medium.com/@evertongomede/the-significance-of-noise-and-outliers-in-data-science-e7175fcf397a

Cribl. (2024, March 11). *What is data filtering?* In *Cribl glossary*.
https://cribl.io/glossary/data-filtering/