Program Control with Hand Gestures

Garpozis Fivos

INDEX

INTRODUCTION

I developed a project, using the capabilities of Arduino. I decided upon motion-controlling a media player, where the user is able to control a program (in this case, VLC Media Player) with hand gestures.

The technique I used to detect hand gestures is called leap motion, which is included in some laptops and desktop computers, or Leap Motion controllers. This technique consists mainly in using hand or finger gestures as an input much like a mouse or a keyboard.

In this report I will explain in detail how that came into fruition between the user interaction, the hardware configuration and the code.

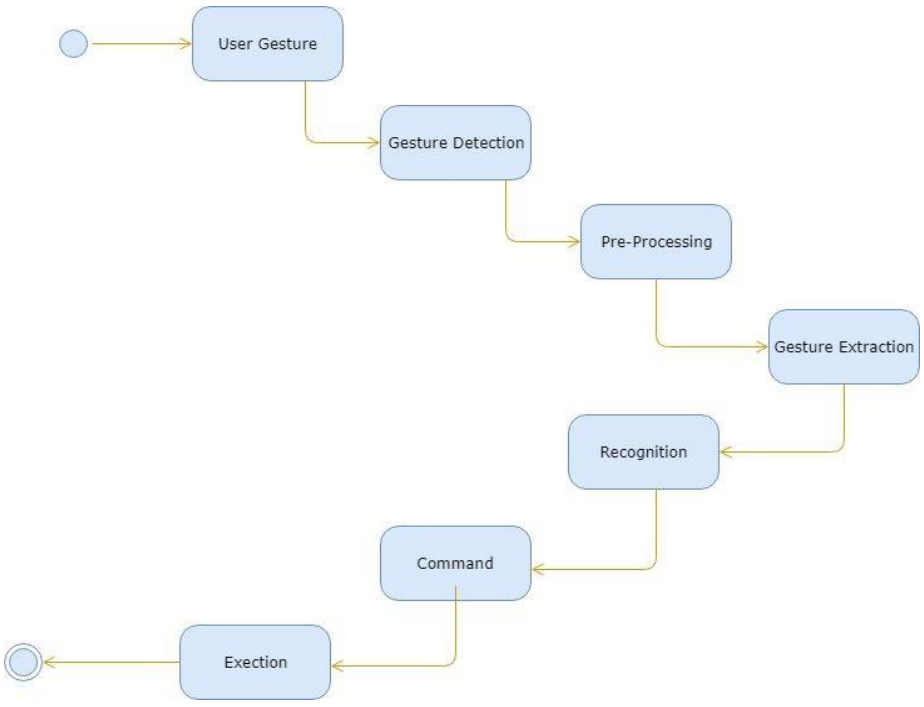

Fig.1 - Leap Motion State Diagram

OBJECTIVES

The main objective is for the program (VLC) to perform the action of the corresponding gesture the user is making.

I set out initially to program the first 5 gestures, I coded the last 2 as an afterthought. I will list the function it triggers, followed by the description of its' corresponding gesture:

1. Play/Pause - to perform this action the user must place both his hands at a specific distance from the sensors. I have decided that this specific distance A (between 25 and 40 cm) will be reused in 3 actions (1, 2 and 3).

2. Forward - to trigger this function the user has to place his right hand to the sensor at the distance of A, doing this will fast forward the video one step

3. Rewind - to rewind the video one step, the user must place his left hand to the left most sensor at the distance of A.

4. Forward/Rewind - both these actions have been programmed in the previous function, but only to forward or rewind one single step in the video. I encountered a problem: what if instead of a single step, I want to forward/rewind multiple steps? It becomes rather inconvenient, so I devised another gesture so the user can do just that. Raising the users' right hand at a nearer distance B (between 5 and 25) and moving it towards the sensor (less than 15 cm) will perform the forward action, moving it away (than 15 cm) would rewind it.

5. Volume Increase/Decrease - the user can increase and decrease the volume, raising his left hand at a distance B and move towards the sensor to increase the volume. To decrease simply put the left hand at the distance of B and move it backward.

6. Toggle Fullscreen – the user is able to Fullscreen by covering the left sensor leaving little to no distance between the sensor and the hand (for better results cover the sensor with the thumb).

7. Mute/Unmute – to mute simply do the same as the gesture above but, this time, with the right-side sensor.

After these were implemented, my secondary goal was to make the interaction less clunky, adjusting distances, slightly altering gestures as to make its detection easier, altering delay times, experimenting with the sensors positioning, etc.

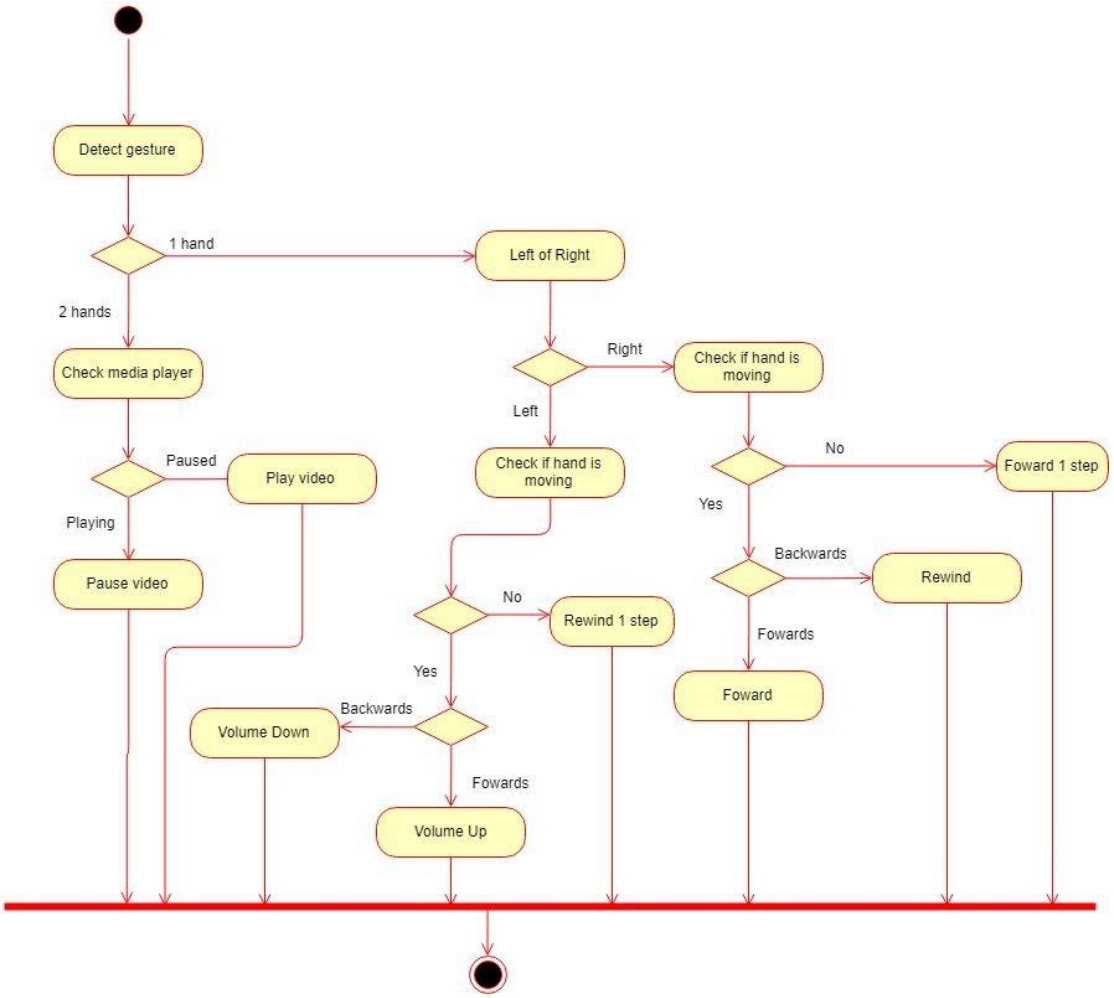

Fig.2 - Old Graphic of User Gestures

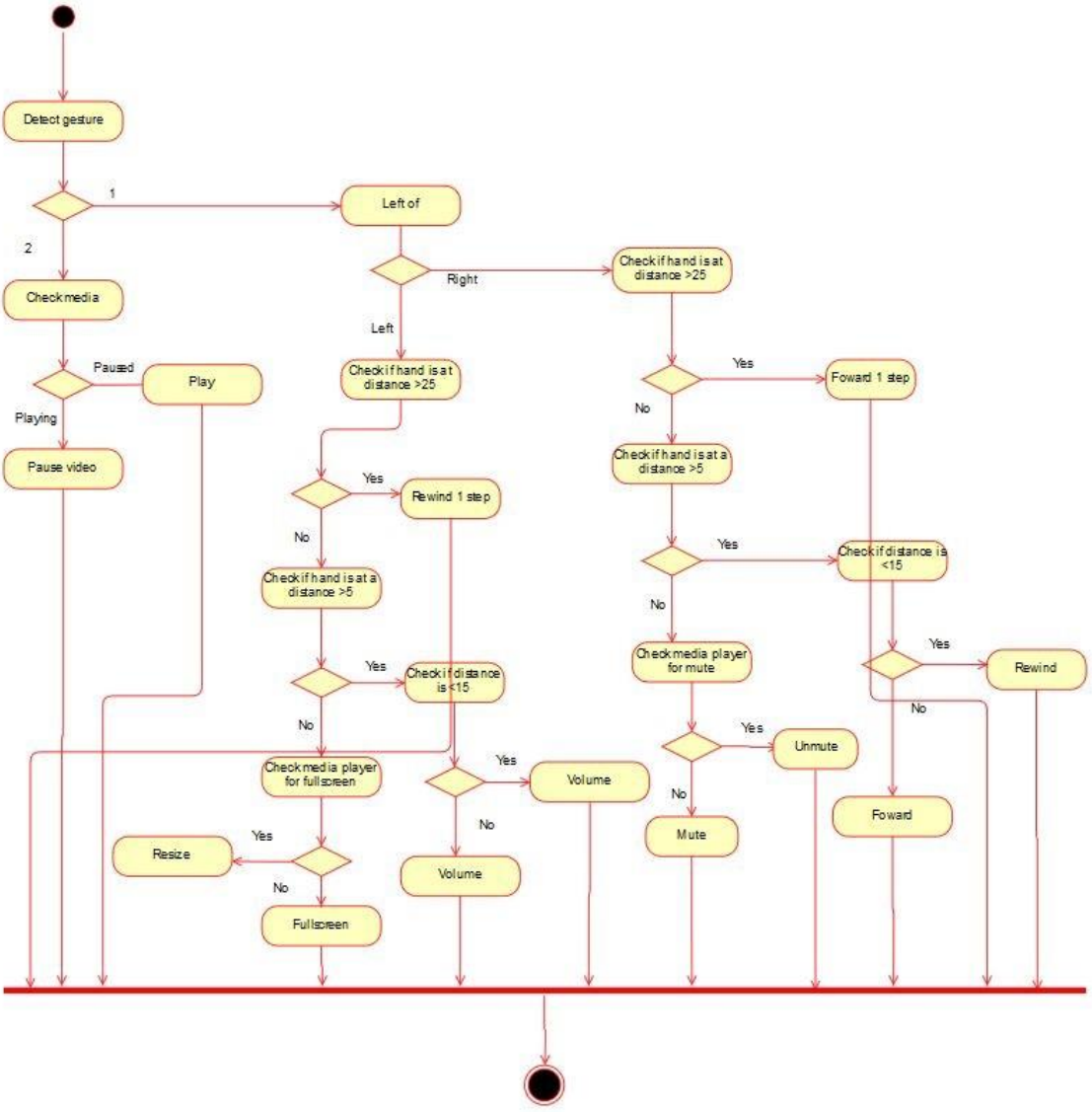

Fig.3 - New Graphic of User Gestures

PROJECT OVERVIEW

There are 3 main components to this project, 2 ultrasonic sensors, an Arduino Uno and the computer. The sensors were connected to the Arduino (The pin configuration is shown in Fig.4). The Arduino is connected to the computer which provides power to the module and also establishes communication through a serial port (USB).

The sensors measure the distance by emitting ultrasonic waves and receiving the reflected wave back from the target (in our case the target is the hand), then measure the time between the emission and reception.

We place 2 ultrasonic sensors on top of the laptop one on either side: these will measure the distance from the hand to the sensor with the help of Arduino.

Afterwards the Arduino will determine which gesture was made by the user and match it to the action command. Following that, the commands from Arduino are sent to the computer through USB.

This data will then be read by python which is running on the computer and based on the read data an action will be performed.

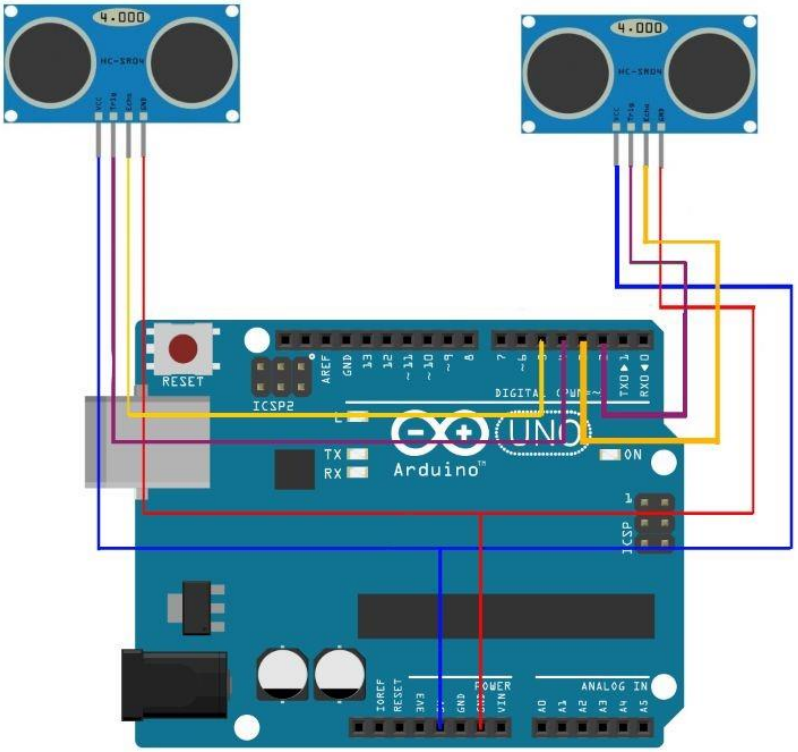

Fig. 4 - Assembly Schematic

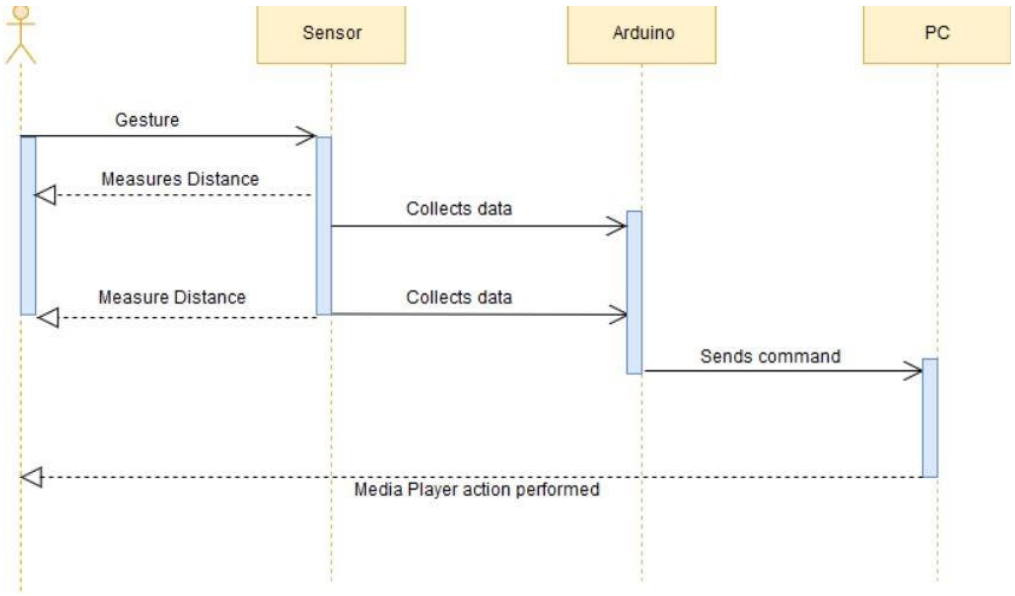

Fig.5 - Sequence diagram

As for programming, it will be necessary for the Arduino to have a code that reads the distance between the hand and the sensor as well as determining which action is to perform according to it. On the computers part, we will have to establish a serial communication with Arduino through the correct baud rate and it will perform hotkey actions. Python will be used for this last part, including the pyautogui and pyserial modules.

CODE

ARDUINO

For programming Arduino I start with defining the I/O pins. The two US sensors are connected to Digital pins 2, 3, 4 and 5 and are powered by +5V pin. The trigger pins are output pins and Echo pins are input pins. The Serial communication between Arduino and Python takes places at a baud rate of 9600.

```
const int trigger1 = 2; //Trigger pin of 1st sensor
const int echo1 = 3; //Echo pin of 1st Sensor
const int trigger2 = 4; //Trigger pin of 2nd sensor
const int echo2 = 5;  //Echo pin of 2nd sensor
```

I need to calculate the distance between the Sensor and the hand each time before concluding on any action. I have written a function named calculate_distance() which will return us the distance between the sensor and the hand:

```
void calculate_distance(int trigger, int echo)
{
  digitalWrite(trigger, LOW);
  delayMicroseconds(2);
  digitalWrite(trigger, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigger, LOW);
  time_taken = pulseIn(echo, HIGH);
  dist = time_taken * 0.034 / 2;
  if (dist > 50)
    dist = 50;
}
```

Inside my main loop, I check for the value of distance and perform the actions. Before that I use two variables: distL and distR, which get updated with current distance value. The code to determine the action to perform is quite simple so we will forgo explanation, as the complete code in the annex.

Since I know the distance between both the sensors, I can now compare it with predefined values and arrive at certain actions.

(I introduced large delays (1000) mainly in toggle actions and single step Forward/Rewind to prevent the actions to be trigger twice accidentally.)

PYTHON

First I import all the three required modules for this project. They are pyautogui, serial python and time:

```
import serial #Serial imported for Serial communication
import time #Required to use delay functions
import pyautogui
```

Next I establish connection with the Arduino through COM port. In my computer the Arduino is connected to COM3 (hardware that the arduino is connected to):

```
ArduinoSerial = serial.Serial('COM4', 9600)#Create Serial port object called arduinoSerialData
```

Inside the infinite while loop, I repeatedly listen to the COM port and compare the key words with any predefined works and make keyboard presses accordingly.

To press a key I simply have to use the command “pyautogui.typewrite(['space'], 0.2)” which will press the key space for 0.2sec. For hot keys like Ctrl+S I can use the hot key command “pyautogui.hotkey('ctrl', 's')”. I have used these combinations because they work on VLC media player. Again, the complete code is available in the annex.

```
while 1:
    incoming = str(ArduinoSerial.readline())  # read the serial data and print it as line
    #print (incoming)

    if 'Play/Pause' in incoming:
        pyautogui.typewrite(['space'], 0.2)
```

(I could use the same code to control other programs, by simply changing the hotkeys.)

PROBLEMS AND FIXES

When I wrote the first code, the first thing I noticed that the code used for the functions 4 and 5 (forward/rewind and volume up/volume down) were far too complex, therefore, very finicky and very fallible.

The first working code was the following:

```
//Lock Left - Control Mode
if (distL >= 12 && distL <= 18)
{
  delay(50);
  calculate_distance(trigger1, echo1);
  distL = dist;
  if (distL >= 13 && distL <= 17)
  {
    Serial.println("Left Locked");
    while (distL <= 40)
    {
      calculate_distance(trigger1, echo1);
      distL = dist;
      if (distL < 10)
      { Serial.println ("Vup");
        delay (300);
      }
      if (distL > 20)
      {
        Serial.println ("Vdown");
        delay (300);
      }
    }
  }
}
```

(I am showing only one of the functions because they are very similar. I tried various solutions, but the simplest was the most user responsive, and fluid of all.)

As I can infer, I have to place my hand in a certain narrow positioning to 'lock it' and then move it forwards or backwards. It was flawed because the sensors are not very precise, and timing our movements accordingly wasn't always successful. Also, since it was such a small distance to 'lock', it was complicated to get it right. So, between timing, distance and precision problems we decided to try a different approach.

To fix this I decided to simplify the code as much as we could, so I wouldn't have to time the gestures or worry about getting a distance.

```
if (distL > 5 && distL <= 25)
{
  delay(50);
  if (distL < 15)
  {
    Serial.println ("Vup");
    delay (50);
  }
  if (distL > 15)
  {
    Serial.println ("Vdown");
    delay (50);
  }


}
```

As I can see, now I have a larger range that, instead of being divided in 3, it's cut in 2, where there is no buffer space and no extra measurements. This makes it easy for the user to find the right place and execute the gesture with ease.

The user could mistakenly place the hand too far or too near, but since the response is immediate it is very quickly corrected.

Other errors were small and easy to fix, such as adjusting distances or not putting enough delays (specially in toggle gestures 1, 6 and 7). This resulted in difficulty in triggering certain actions and multiples of the same action.

CONCLUSION

In the conclusion, I understand that the real world interaction is very different from simulations and digital inputs (keyboard, mouse, etc).

If I redid the project, the one major change I would implement would be to get better (and far more expensive) sensors since the ones I used were inaccurate, imprecise and projected weak waves. Far better sensors would also allow me to include more commands in my project (change zoom mode, toggle subtitles, play next movie from the playlist, change aspect ratio, etc).

ANNEXES

Complete Arduino Code:

```
 const int trigger1 = 2; //Trigger pin of 1st sensor
const int echo1 = 3; //Echo pin of 1st Sensor
const int trigger2 = 4; //Trigger pin of 2nd sensor
const int echo2 = 5;  //Echo pin of 2nd sensor
long time_taken;
int dist, distL, distR;

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

  pinMode(trigger1, OUTPUT);
pinMode(echo1, INPUT);
pinMode(trigger2, OUTPUT);
pinMode(echo2, INPUT);

  Serial.print("Start");
}

void calculate_distance(int trigger, int echo)
{
  digitalWrite(trigger, LOW);
delayMicroseconds(2);
digitalWrite(trigger, HIGH);
delayMicroseconds(10);
digitalWrite(trigger, LOW);
time_taken = pulseIn(echo, HIGH);
dist = time_taken * 0.034 / 2;
if (dist > 50) dist = 50;
}
void loop()
{
```

```
 calculate_distance(trigger1,          echo1);
distL = dist;
 calculate_distance(trigger2,          echo2);
distR = dist;

 Serial.print("L=");
 Serial.println(distL);
 Serial.print("R=");
 Serial.println(distR);

 if ((distL > 20 && distR > 20) && (distL < 40 && distR < 40))
 {
          Serial.println("Play/Pause");
          delay (1000);

 }

 if ((distL > 25) && (distL < 40 ) && (distR == 50))
 {
          Serial.println("Rewind");
           delay (1000);
 }

 if (( distR > 25) && ( distR < 40 ) && (distL == 50))
 {
          Serial.println("Forward");
          delay (1000);
 }

 if (distL >= 0 && distL <= 5)
 {
          delay(50);
          calculate_distance(trigger1, echo1);
          distL = dist;
```

```
            Serial.println("fullscreen");
            delay (1000);
 }
 if (distR >= 0 && distR <= 5)
 {
            delay(50);
            calculate_distance(trigger2, echo2);
            distR = dist;

            Serial.println("mute");
            delay (1000);
 }

 if (distL > 5 && distL <= 25)
 {
            delay(50);
            if (distL < 15)
            {
            Serial.println          ("Vup");
delay (50);
            }
            if (distL > 15)
            {
            Serial.println          ("Vdown");
delay (50);
            }


 }
 if (distR > 5 && distR <= 25)
 {
            delay(50);
```

```
        if (distR < 15)
        {
        Serial.println        ("Rewind");
delay (50);
        }
        if (distR > 15)
        {
        Serial.println        ("Forward");
delay (50);
        }


 }
 delay(200);
}
```

Complete Python Code:

```
import serial #Serial imported for Serial communication
import time #Required to use delay functions import
pyautogui

ArduinoSerial = serial.Serial('COM4', 9600)#Create Serial port
object called arduinoSerialData
time.sleep(2)

while 1:
 incoming = str(ArduinoSerial.readline()) # read the serial data
and print it as line    #print (incoming)
```

```
   if 'Play/Pause' in incoming:
pyautogui.typewrite(['space'], 0.2)

   if 'Rewind' in incoming:
       pyautogui.hotkey('ctrl', 'left')

   if 'Forward' in incoming:
       pyautogui.hotkey('ctrl', 'right')

   if 'Vup' in incoming:
       pyautogui.hotkey('ctrl', 'down')

   if 'Vdown' in incoming:
       pyautogui.hotkey('ctrl', 'up')

   if 'fullscreen' in incoming:
pyautogui.hotkey('f')

   if 'mute' in incoming:
pyautogui.hotkey('m')
    incoming = " "
;
```

REFERENCES

AswinthRaj. (n.d.). *Control your Computer with Hand Gestures using Arduino*. Ανάκτηση από https://circuitdigest.com/microcontroller-projects/control-your-computer-with-hand-gestures