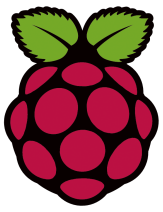

# Theremin Pi

# Building a Theremin using the Raspberry Pi and a 3D capacitive touch sensor.


Difficulty level: 8/10;
Language: "C++";
Opensource: true;
Cost: 60;

## Preface

In this project you will learn how to build a theremin using a Raspberry Pi. This project is for people who already have got some experience with c++, Makefiles and working with the terminal. The project uses a 3D capacitive touch sensor to register hand movement over the xyz axis above the sensor. It is produces sound using the internal headphone jack and the Jack 2 connection library. All the sounds are controlled and being generated by using c++. The hand gestures are being tracked by the sensor which is being read with the c++ library provided by the sensor developers.

## Parts List

| Part Name | Price | Where to buy |
|---|---|---|
| Raspberry Pi | 30 euro | kiwi-electronics.nl |
| Flick3D Hat | 22,5 euro | kiwi-electronics.nl |
| 16GB SD card | 5 euro | dataio.nl |

# Stepwise Plan

## 1 Get the parts

Buy the necessary parts from the parts list

## 2 Setting up the sd card

1. Get Raspbian Lite from the official website
2. Download and install balenaEtcher
3. Insert the SD card into your computer and open balenaEtcher.
4. Select the downloaded Rasbian image or zip file. Select your SD card and click flash.
5. Now browse to the sd card and add a file called "SSH". Just in file format, not .txt or any other extension.

## 3 Connect your Raspberry to your Computer

Insert the SD Card into your Raspberry and hook it up to a power.
From now on we are going to program the Pi using our external computer.

1. Power the Pi again by using a micro usb cable so the device will restart.
2. Connect a UTP or Ethernet cable to the Pi and your main Computer.
3. Download and install the following program: Angry Ip Scanner
4. Share your network with the one of the raspberry pi.
5. Open angry Ip scanner. Set your starting Ip with your DNS Ip and end with .0 and set your end Ip the same way but end with .255:
   start:
   ```
   192.168.137.0
   ```
   end:
   ```
   192.168.137.255
   ```
6. Press start and wait till there is a name in the list called Raspberry Pi Home. Remember that Ip address
7. Open your terminal of preference. For this project I have used Bash and will be referring using the UNIX syntax.
8. Use the folling command to connect to your Pi using the Pi's Ip address:
   ```
   ssh pi@[your-ip-goes-here]
   ```
9. Enter you password and now you will see you are connected to your raspberry!
10. Now that we are here, enter the following command:
    ```
    sudo raspi-config
    ```
    Browse with the arrow keys to interfaces and press enter.
    From here we can enable I2C for the flick3d hat, browse there with the arrow keys press enter and select enable. Now you can safely exit this menu by selecting finish.

## 4 Downloading the right packages

Before we can start writing any program we need the right packages installed.

1. First update and upgrade the os, get the newest version
   (This may take a while):

```
sudo apt-get update
sudo apt-get upgrade
```

2. Now install the right library for the 3D touch sensor:

```
curl -sSL https://pisupp.ly/flickcode | sudo bash
```

   And the following line so the sensor can work

```
sudo apt-get install wiringpi
```

3. Install git and download and install the right files and cpp files from my githubpage:

```
sudo apt-get install git
```

   Download the package:

```
git clone https://github.com/Revess/ThereminPi
```

4. Install make:

```
sudo apt-get install make
```

5. Install Jack without X11 so we can actually run it. Solution found here Run the following commands for everything to work:

```
sudo apt-get install -y libasound2-dev libsndfile1-dev libreadline-dev libreadline6-dev
libtinfo-dev
git clone https://github.com/jackaudio/jack2.git --depth 1
cd jack2
./waf configure
./waf build
sudo ./waf install
sudo ldconfig
sudo sh -c "echo @audio - memlock 256000 >> /etc/security/limits.conf"
sudo sh -c "echo @audio - rtprio 75 >> /etc/security/limits.conf"
```

   Now reboot the system for everything to work:

```
sudo shutdown -r now
```

## 5 Adding the Flick3D sensor

1. The installation of the sensor hat itself is pretty straight forward. Screw the standoffs in place and click the hat on top of the GPIO pins. Then screw the left over screws in place.
2. Now to check if everything is working correctly type the following code in your terminal:

```
flick-demo
```

   It will jump to a screen where you can read out the data of the sensor. Check if everything works and quit by pressing: "ctrl+c"

## 6 Writing the program

1. For this project we will be using C++ to write the software. Not only because it will be faster then Python or any other software. Also because there is no good synthesizer library for Python. Of course using PureData as the synthesizer is an option, but I want to stay away from for this project. All you need for program to talk to Jack is the following code from my GitHub repository (already downloaded at step 4):
   here

2. Now that you have the following files from my github: main file, an oscillator class and a sine.cpp subclass etc. These files will form the basics for the Theremin. In the main.cpp you will find code that is responsible for handling the flick sensor input. Let me go ahead and explain how each component works:
    - At the top of the code you will find the gesture handlers, these pick up the data from the sensor and divide them based on the message they get from the "flick.h" file.
      In the main itself you will also find the place where the Flick class will be created. Via this class the program will talk to the sensor. To keep asking the sensor if the data has changed we are using the "flick.poll();" function. As you can see we are using this in a while loop to ensure this keeps updating until the user decides to close the program.
    - In the main.cpp you will also find the audio loop where we connect to Jack. Upon connecting to jack we will send out pieces of audio according to the buffer size set by your audio device. To fill the buffer we are talking to the oscillator class. In the class we are requesting a new calculation for each place in the buffer. With "oscillator->getSample();" we are requesting the current sample value of the oscillator, with "oscillator->tick();" we tell the oscillator to move the phase by one sample. This way the buffer will first read and then write the oscillator.
    - The oscillator.cpp will take care of calculating the current phase. For everything everything to working correctly we need another class that will take care of calculating the sine wave. Luckily the sine class will take care of this, at every phase position the sine will be calculated by using a the "sin()" function of c++.
    - In the while loop where the flick.poll(); gets updated you can now add yourself user control to the sensor. Store each type of data in a variable (x=width, y=depth, z=height). Now you can control the frequency of the sine wave by using the following function; "oscillator->setFrequency(double frequency)", and change the amplitude by using; "oscillator->setAmplitude(double amplitude)".

    All these different parts give the theremin its functionality. Feel free to add your own parts of code of course!
3. For the last step you need to compile the code using make. Type make into your terminal when inside the thereminPi folder. Now the code will compile the program to the bin folder. Run the code by using the following command line:

```
./bin/thereminPi
```

## 7 Creating the startup script

Now that the program has been written it only needs to be added to the startup script of the Pi. For this to work the program needs to be added to systemd.

1. Copy the following file from the theremin folder to the systemd folder:

```
sudo cp /~/thereminPi/theremin.service /etc/systemd/system/theremin.service
```

2. Now restart the deamon service:

```
sudo systemctl daemon-reload
```

3. enable your compiled script:

```
sudo systemctl enable theremin.service
```

4. reboot the system:

```
sudo reboot
```

In the "theremin.service" file you can specify the file location at the command that says "ExecStart"

# Obstacles

## Picking the sensor

Already the first obstacle I came across, what sensor to use for this project? The basis is very simple, measure the distance. After doing some research I settled with two kinds of sensors to use:

- Infrared
- Ultrasonic

Very quickly I came to the conclusion, especially after trying this myself and buying this sensor, that there is a problem with using these sensors. A theremin needs two of them. These sensors emit some kind of waveform at a high frequency. When the waveform hits an object it gets reflected back to the receiver. The problem gets created when you start using two of these sensors at a short distance from eachother, they will start interfering with eachother. The other problem is that the sensors are using analog signals so we need to convert that first and adds extra complexity to our circuit. So from here it was back to the drawing board.

My teacher Hans Leeuw borrowed me a capacitive 3D touch sensor he used for a project years ago. First I tested the device on my Windows 10 machines and the device worked perfectly! The next step was to move to the RPi. The device provided big SDK package written in c++. After using 'make' to create a runnable application for Linux I plugged the device in and nothing. I did not get the device to talk to the Raspberry and went back to the drawing board again. Three time's a charm and I found the right sensor, a hat for the RPi. The sensor just snaps right into place and talks to the Raspberry over an I2C connection.

## Generating audio

This has been a real struggle, which audio library am I going to be using. After a lot of trying and failing I decided Jack is my best option. Portaudio was too difficult to learn. For Python there were no good working synthesizers available.
Using Jack introduced its own problems, Jack needs x11 to run for some reason. This is a problem, because when running the RPi headless (without a monitor) x11 was disabled and I was unable to run Jack. After a lot of research I came across this forum comment which suggested building Jack yourself and installing it by hand. Which I did and it worked! So for Jack to run on a RPi headless I would like to refer to step 4, substep 5, on how to do this yourself.

## C++ and the Pi

Here comes a small list of things that will break your code when using C++ on a RPi:

- Queue; very inconsistent for some reason. Sometimes the queue overloads and will make the code crash. Tried to use this to make a nice data smoother.
- Switch; always use the "default:" at the end of your switch, always.
- Order; Think of the order of things in your code very carefully. Sometimes variables get updated in the wrong ways.

## Startup script

For the startup script on the Pi I would definitely recommend using systemd over rc.local Systemd runs very stable and also just works! Make sure to use the way of referring to the file location at the "ExecStart" command in the .service file.