Arduino setup guide for Otto robots

Download & install Arduino IDE

If you have Otto Blockly installed and working on your PC **you do not need to install Arduino because you can open .ino files directly from Blockly**

Select the software

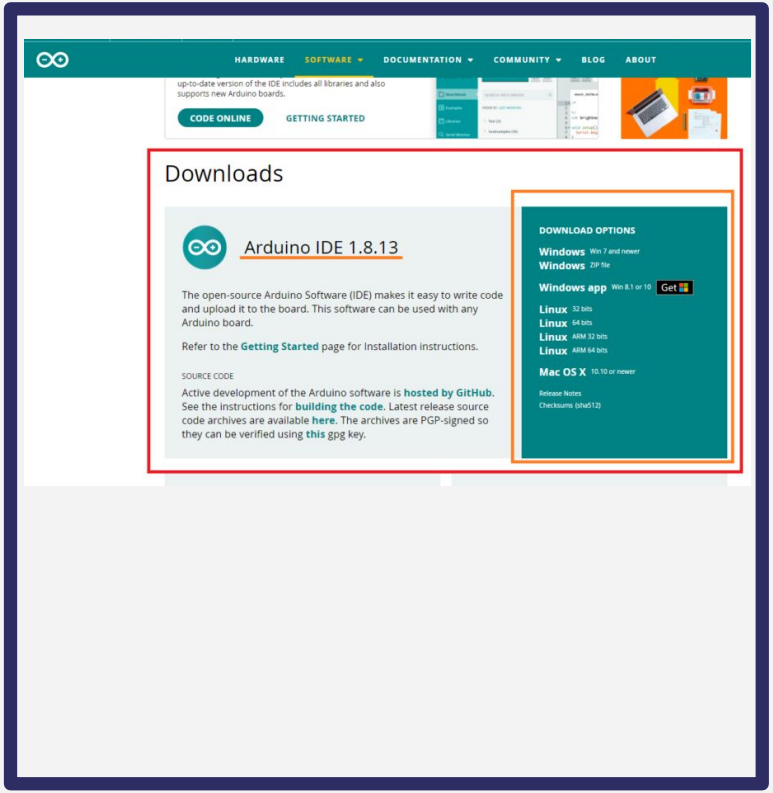
You must select the option that correspond to our operating system.

Not sure the version of your operating system? Check these links.

Windows and how to install

Linux and how to install

Mac OS and how to install

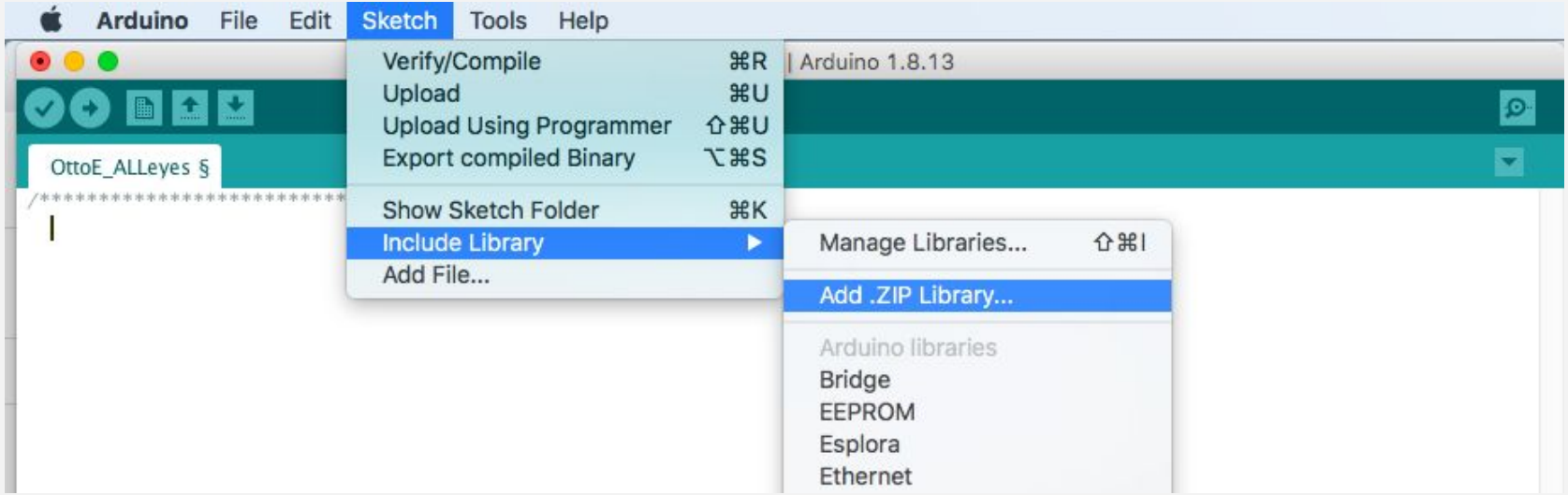


Download Otto DIY Libraries

Link source: https://github.com/OttoDIY/OttoDIYLib/

Install Otto Libraries in Arduino

Open Arduino and select **Sketch** from the menu bar, then **Include Library** followed by selecting **Add .ZIP Library**.

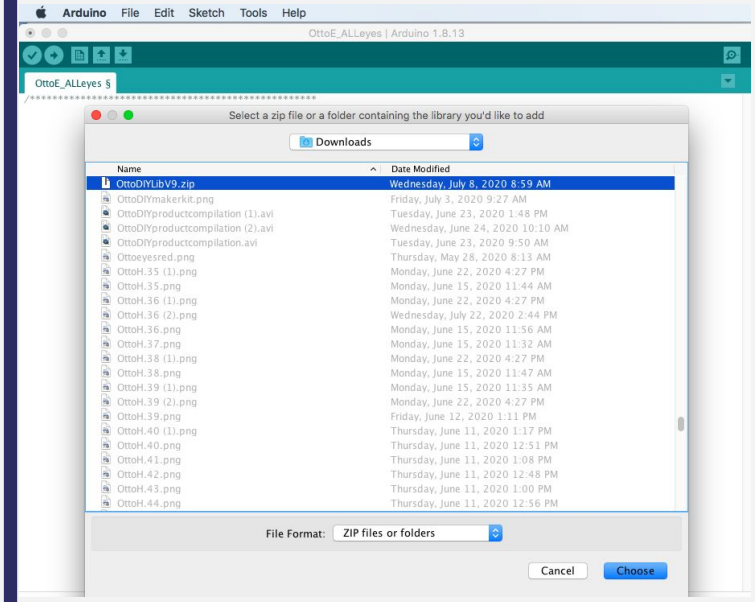


Include the Libraries

You will be prompted to select the library. Navigate to the .zip file's location, that you just downloaded and open as it is.

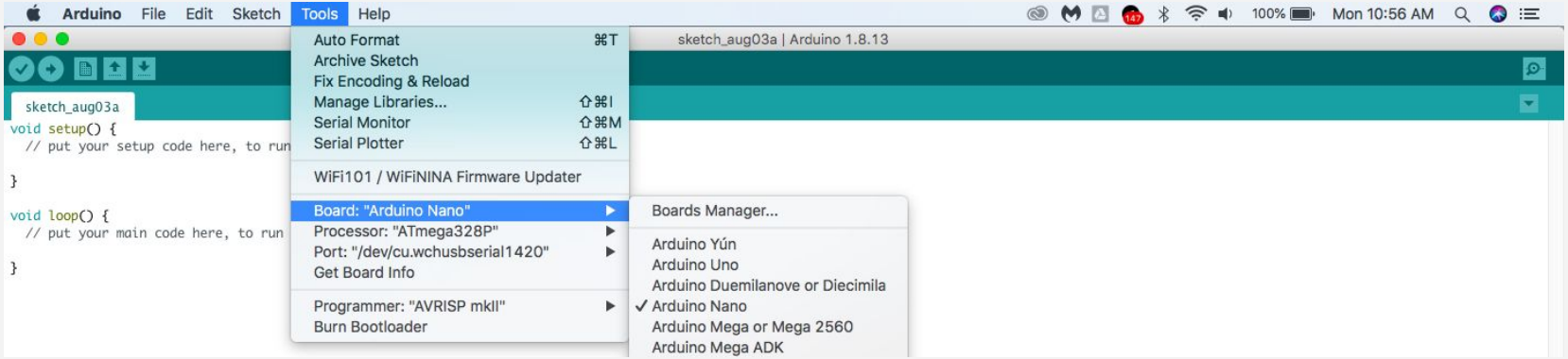
Navigate under downloads until you see the OttoDIYLib.zip then select **that file**.

To check if it was uploaded, select **Sketch** again > Include Library > & scrolling down to the bottom you should see **OttoDIY_Lib**

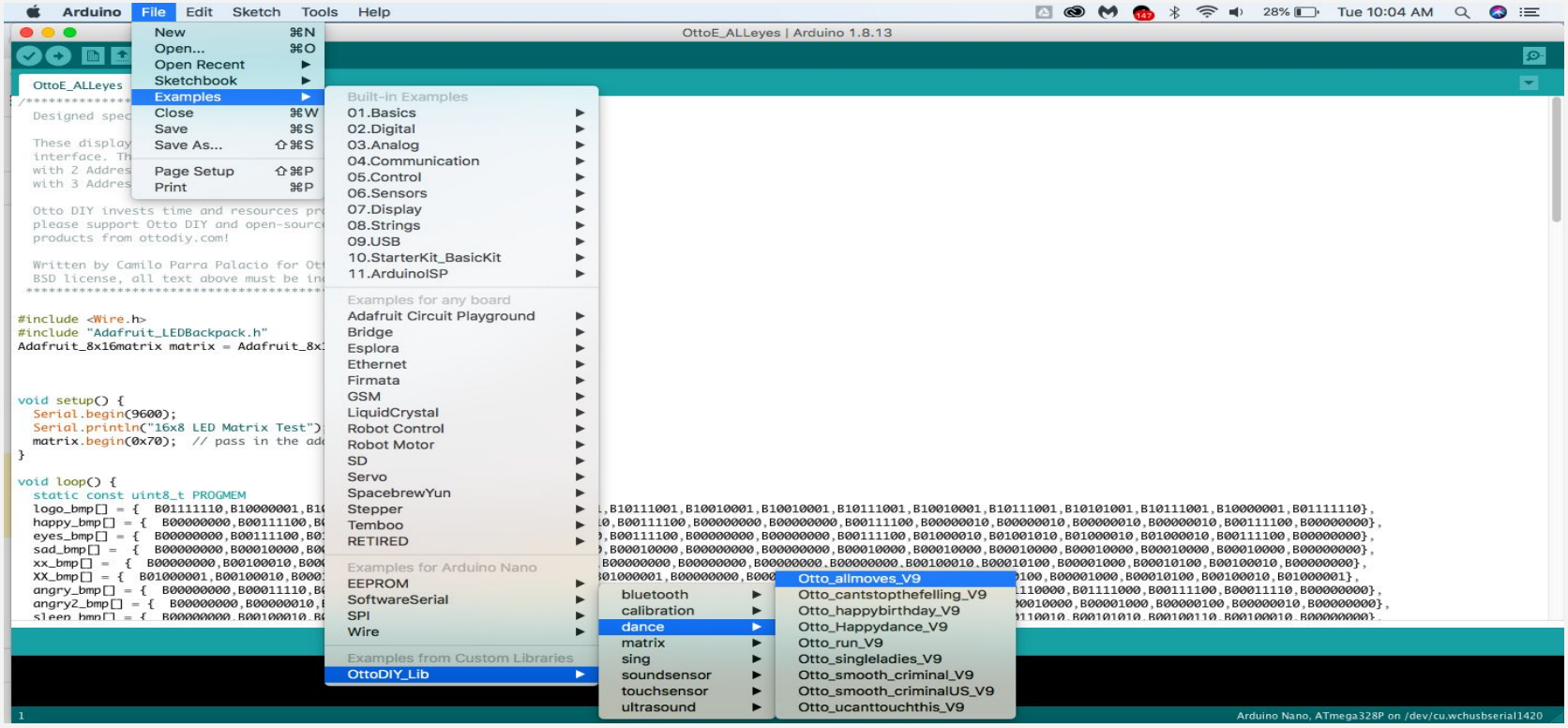


Select Board, Processor & Port Settings

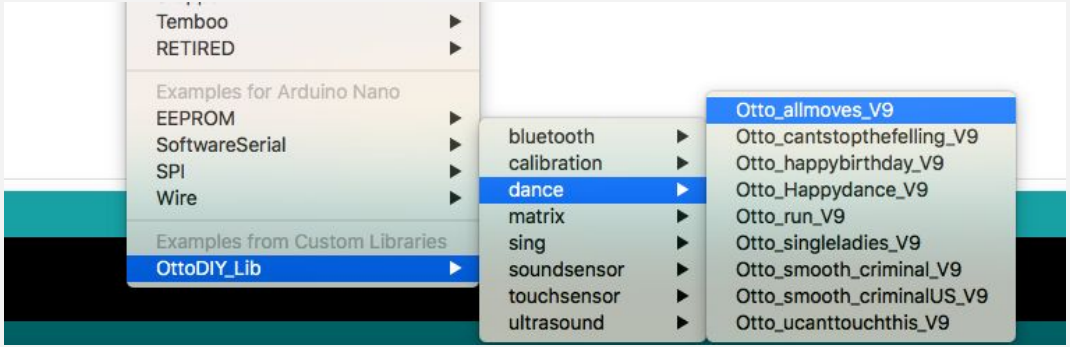
1) Select tools from menu bar
2) Board select “Arduino Nano”
3) Processor select “Atmega328P” (if error select **Old bootloader**)
4) Port select “COM #” (where your Otto is connected, this can be different in your computer) if not visible, driver was not installed

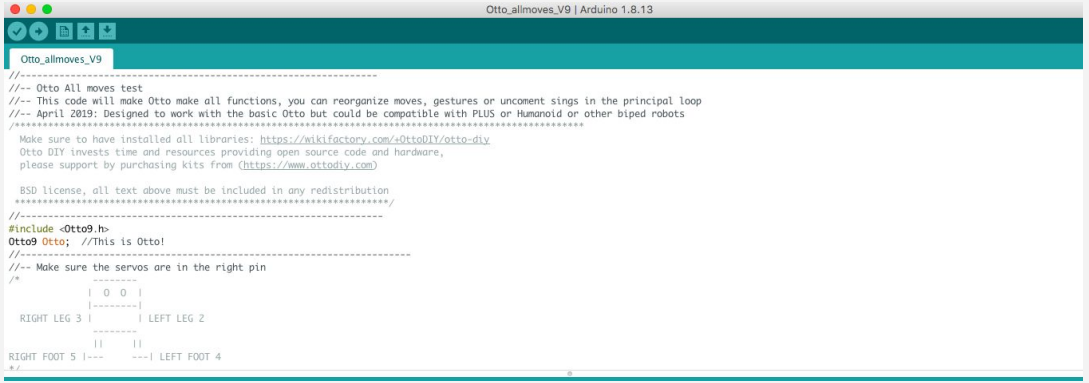


File/Examples/OttoDIYLib/Otto_allmoves



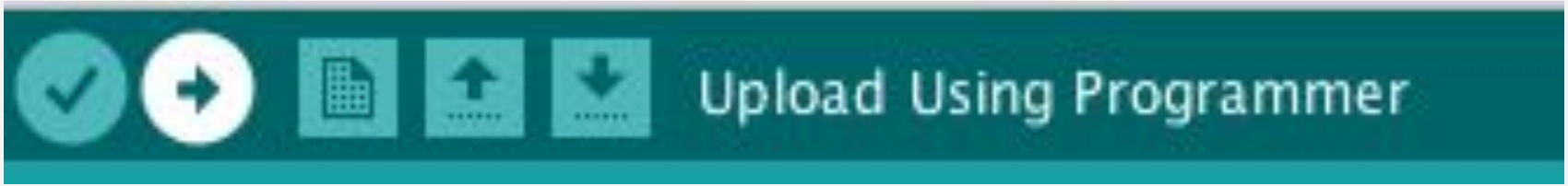
Open Codes from examples menu





Upload code to Otto

Lastly select the arrow pointing to the right to upload.



This will automatically **check the code** and if good it will immediately upload the code to Otto.

If there were no problems, your Otto is now a walking dancing machine! Well only the legs but that is ok because now we tested that everything is working ;) If not you need to check the previous steps again.

You are all setup!

Now you can code Otto using Arduino IDE **play with the other examples** depending on your robot kit.

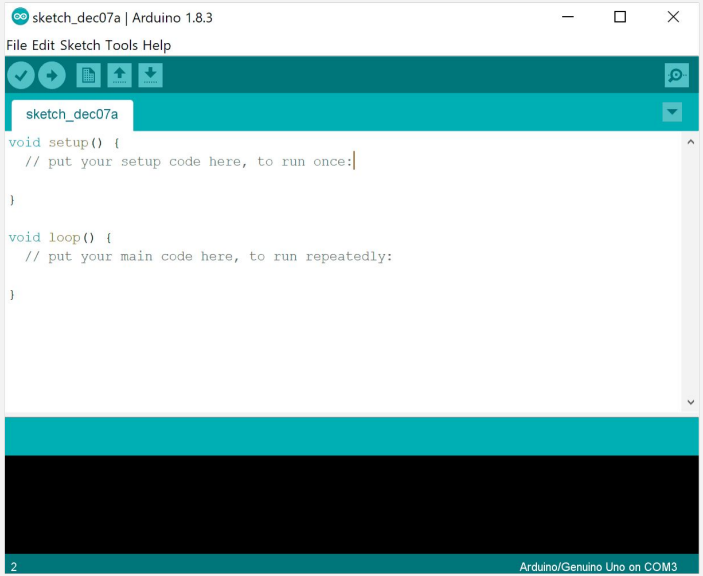
Text based programming

Text based

Languages that are typed using a keyboard and stored as a text files.

Typically when people talk about **text-based languages they are referring to programming languages such as Python**, Lua and JavaScript which are 'real' programming languages that are used by professional software developers.

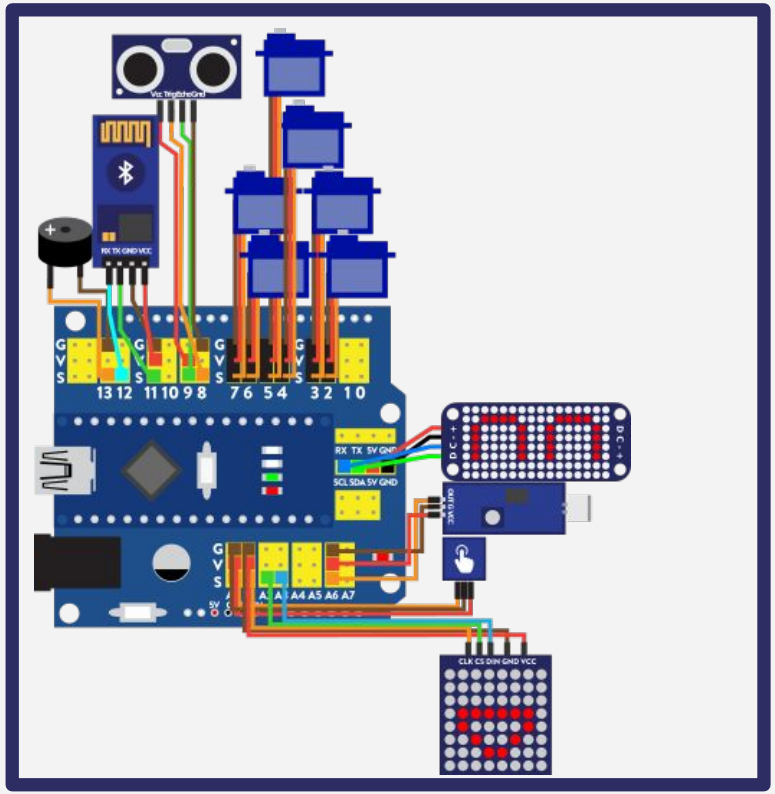
Arduino IDE is based on C++ language



Otto Libraries

Otto.h and Otto.cpp // main functions
Otto_sounds // sound functions
Otto_gestures // gestures functions
Otto_mouths // mouth functions
Otto_matrix // matrix functions

Oscillator is the main algorithm for the servos ¨smooth¨ movement



Wiring shield + nano

Electronic diagram of all components you can expand your robot with our kits, **Otto is modular,** you can plug or unplug them to have more or components as you decide and just change the Pins in Otto Blockly.

As a good practice, check and test your electronics and software. Avoid having to disassemble the whole robot later to fix something.

Components like the 16x8 LED matrix for the eyes can be interchanged for and OLED display or any other **I2C device**.

Touch you can swap to any other **digital sensor.**

Sound you can swap to any other **analog sensor.**

Otto Functions

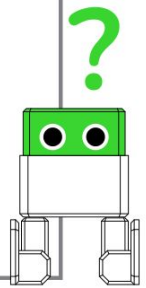
This are simple lines of text that send commands, they are very easy to understand to anyone unfamiliar with the programming of Otto or even coding itself.

With a simple analysis of the parameters we can get to understand the **commands that we can send to the robot.**

Let's practice with an example, the **Otto avoiding obstacle code** on the right.

Otto_avoid.ino

```
void loop() {
if(obstacleDetected){
Otto.sing(S_surprise);
Otto.playGesture(OttoFretful);
Otto.sing(S_fart3);
Otto.walk(2,1300,-1);
Otto.turn(2,1000,-1);
delay(50);
obstacleDetector();
}
else{
Otto.walk(1,1000,1);
obstacleDetector();
}
}
```



Otto Functions

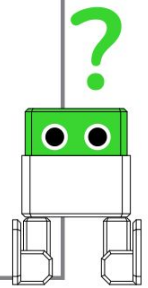
From this example, Otto will walk endlessly until detecting an obstacle.

When this occurs, Otto will **sing surprise**, then **fretful**, followed by a **fart** sound. Next, it **walks back** 2 times, **turns right** 2 times and if no obstacle, Otto will go back to **walking** endlessly.

All those Otto actions are functions! Thanks to the libraries, this makes our life easier by compressing lots of complicated code into simple text commands.

Otto_avoid.ino

```
void loop() {
if(obstacleDetected){
Otto.sing(S_surprise);
Otto.playGesture(OttoFretful);
Otto.sing(S_fart3);
Otto.walk(2,1300,-1);
Otto.turn(2,1000,-1);
delay(50);
obstacleDetector();
}
else{
Otto.walk(1,1000,1);
obstacleDetector();
}
}
```



Otto Sing

Next if we use the Otto.sing function we can change for the emitting sounds:

See how simple it is now?

By just changing what is **inside the ()** we can swap the sounds easily to other 19 different ones.

To make multiple is as simple as copying and pasting in a new row to make the sounds as many times as you like.

Otto Walk

Let's take a look to a more complex function for Otto to **walk** and dance!

These functions have a different structure inside the () as you can see it is more about **parameters and numbers** that we change to alter the speed, direction, and size of the movements.

#steps are just how many times you want to repeat that movement without the need of further coding or adding additional rows. For example, Otto.walk (**5**,1000,1) is **5 steps**.

Otto Walk

Speed it is translated in milliseconds with “T” meaning **“period of time”** in the libraries. The higher the VALUE, the faster the movement.

For example: Slow=2000 **Normal=1000** Fast= 500. Therefore for the function Otto.walk(5,**2000**,1), **2 seconds** for a step of the movement.

The last number is for **Direction** with

1 for forward/left

-1 for backward/right.

Otto Dances

For **moveSize** function, this is for additional Otto moves. It can be Small=5 Medium=15 and Big=30 but feel free to play with other values and see what happens!

For example, **moonwalker** you can try between 15 and 40, crusaito 20 to 50, flapping 10 to 30, swing 0 to 50, up down 0 to 90, tiptoe and swing 0 to 50, up down 0 to 90, jitter 5 to 25 or ascending turn 5 to 15. Thus, you can have Otto do the moonwalk using this function: Otto.moonwalk(5,1000.**28**,25) with Otto doing a medium swing from the servo for the moonwalk.

Otto Gestures

Finally, our favorite, which is gestures.

This is a combination of the 2 previous functions we learnt **sing + walk**

Their goal is to express emotions by combining sounds with movements at the same time and if you have the Otto Humanoid expansion using the LED matrix you can show them in the robot mouth!

As you see it's very simple, but what it does is quite advanced.

Learn more

If you want to learn more about how the functions work behind these simple calls, you can open the source files of the libraries and you will see all the "secret" code behind them.

It is recommended for you to understand how the code works but please note you will need some advanced knowledge of C coding.

Due to Otto being open-sourced we can see all the backend of the programming, but is up to you to learn it!

```
class Otto
{
  public:

    //-- Otto initialization
    void init(int YL, int YR, int RL, int RR, bool load_calibration, int Buzzer);
    //-- Attach & detach functions
    void attachServos();
    void detachServos();

    //-- Oscillator Trims
    void setTrims(int YL, int YR, int RL, int RR);
    void saveTrimsOnEEPROM();

    //-- Predetermined Motion Functions
    void _moveServos(int time, int  servo_target[]);
    void _moveSingle(int position,int  servo_number);
    void oscillateServos(int A[4], int O[4], int T, double phase_diff[4], float cycle);

    //-- HOME = Otto at rest position
    void home();
    bool getRestState();
    void setRestState(bool state);

    //-- Predetermined Motion Functions
    void jump(float steps=1, int T = 2000);

    void walk(float steps=4, int T=1000, int dir = FORWARD);
    void turn(float steps=4, int T=2000, int dir = LEFT);
    void bend (int steps=1, int T=1400, int dir=LEFT);
    void shakeLeg (int steps=1, int T = 2000, int dir=RIGHT);

    void updown(float steps=1, int T=1000, int h = 20);
    void swing(float steps=1, int T=1000, int h=20);
    void tiptoeSwing(float steps=1, int T=900, int h=20);
    void jitter(float steps=1, int T=500, int h=20);
    void ascendingTurn(float steps=1, int T=900, int h=20);
```

Become a programmer in our community

join now!

Install Driver on Mac

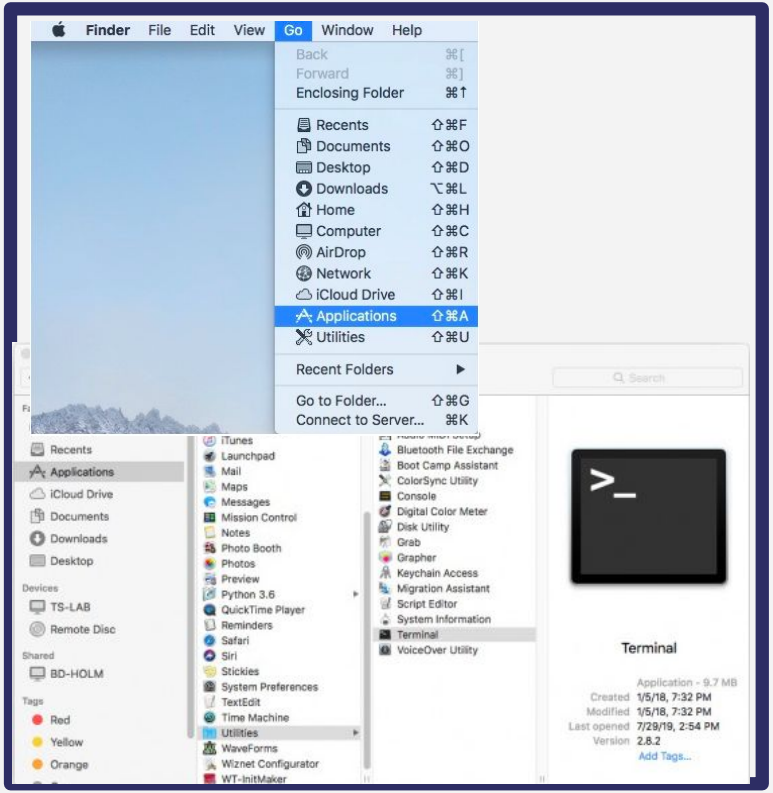
This slides are if you had problem seeing Otto when connected to USB

From your desktop click **go**, and then **applications**

Click **utilities**, then **terminal**

For Mac OSX v10.9+ type in the terminal and hit enter: **cd /Library/Extensions**

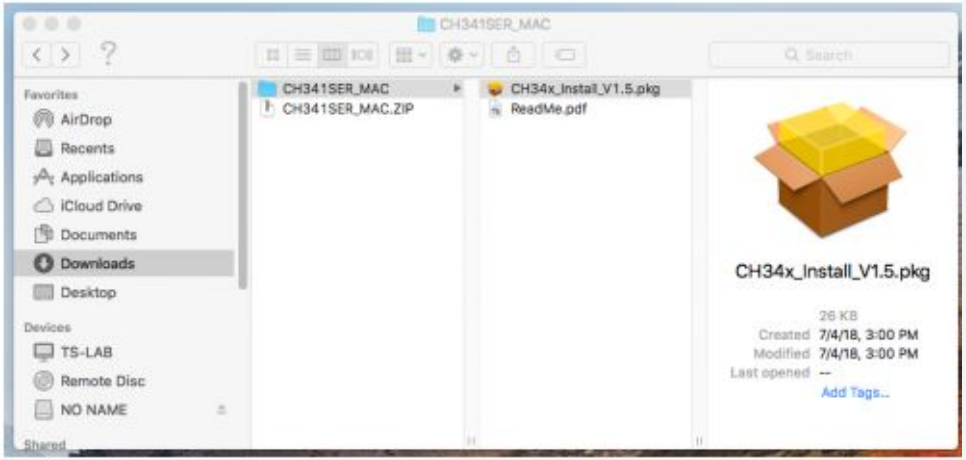
For Mac OSX v10.8 and below type and hit enter:
Cd /System/Library/Extension



Install CH340 Mac

Next, download & install the CH340 Driver for Mac. You can get it from here.

Once installed you will need to restart the computer and done!



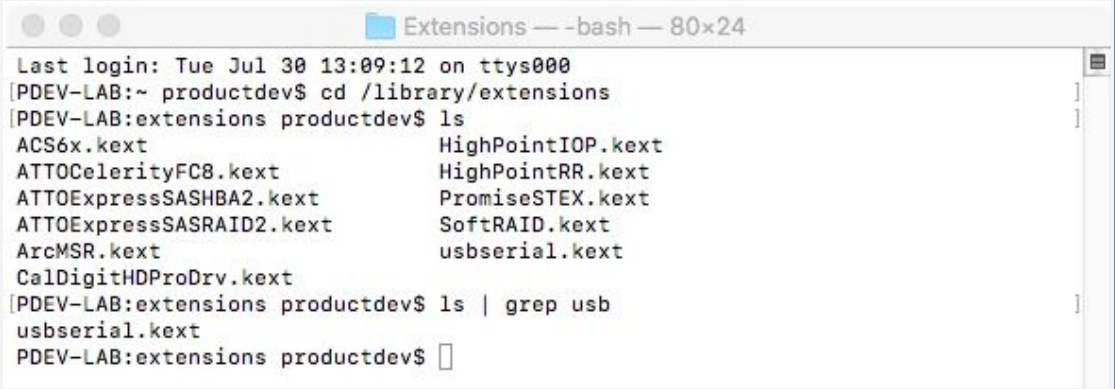
Lets double check

To check to see if the CH340 driver is in the correct path, use the following command to list the contents of the folder: ls

To look for CH340 driver files (i.e. **usb.kext** or **usbserial.kext**) in the path, you could use the following command:

```
ls | grep usb
```

You should see something similar here



Troubleshooting driver continued

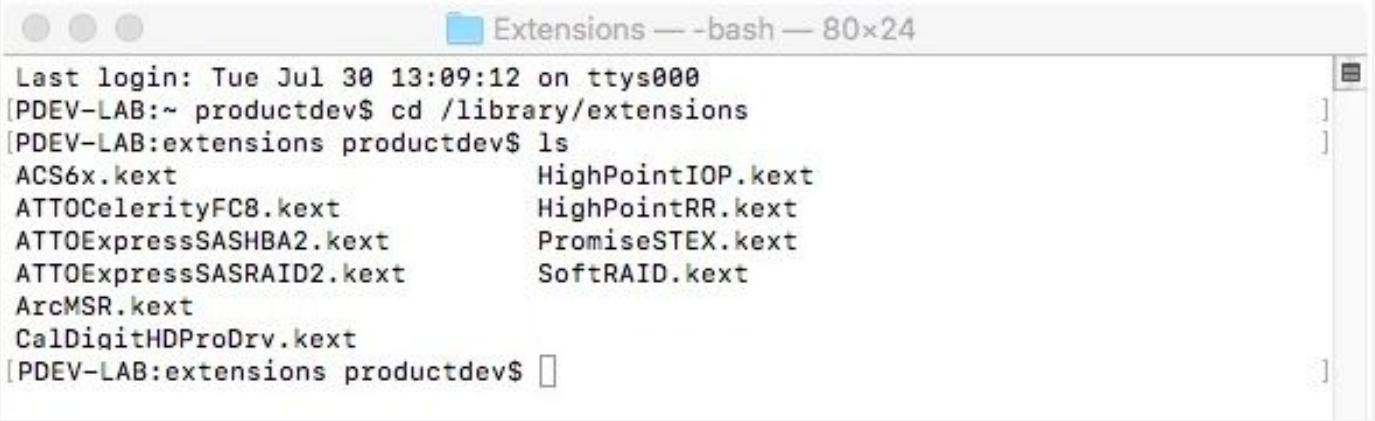
If you have found the file in the path, you will need to run each of the following commands below in the CLI/Terminal to remove old CH340 drivers. In this case, there was only the **usbserial.kext** file but it does not hurt to run both commands. Make sure to have administrative privileges to ensure that the drivers are removed.

```
sudo rm -rf /Library/Extensions/usb.kext
sudo rm -rf /Library/Extensions/usbserial.kext
```

Troubleshooting driver continued

Check if the old drivers were removed in the paths by using the ls command with your respective OS version. You will notice that the *.kext file is removed from the respective paths. In this case, the usbserial.kext was removed from Mac OSX High Sierra.

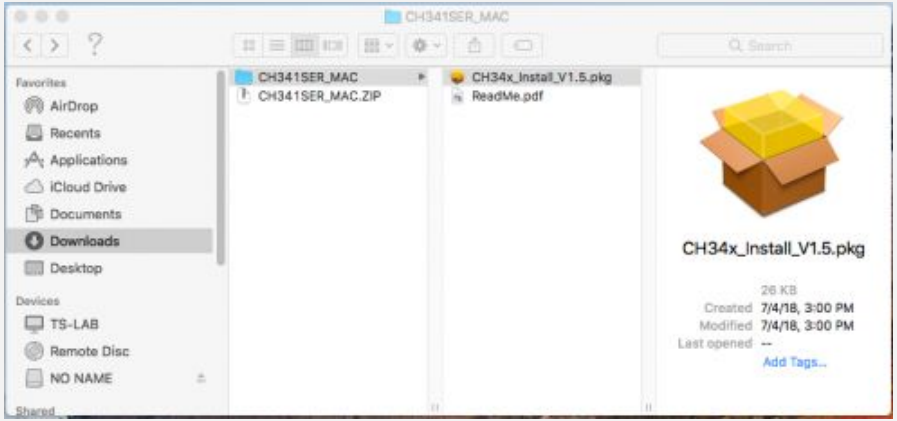
```
ls
```



Troubleshooting driver continued

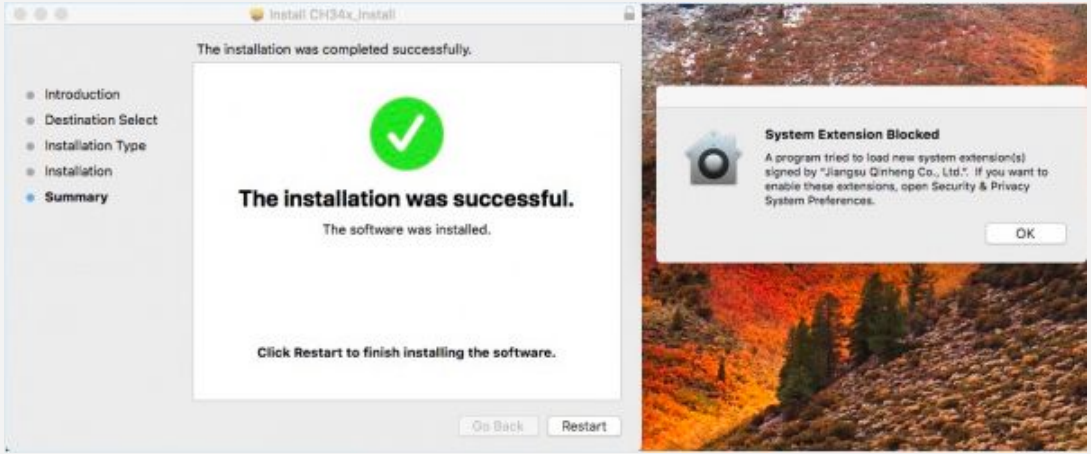
Next, download and extract the folder here.

Then, open the "***.pkg**" file from the unzipped folder and follow the instructions. You'll need to restart your computer for the changes to take effect.



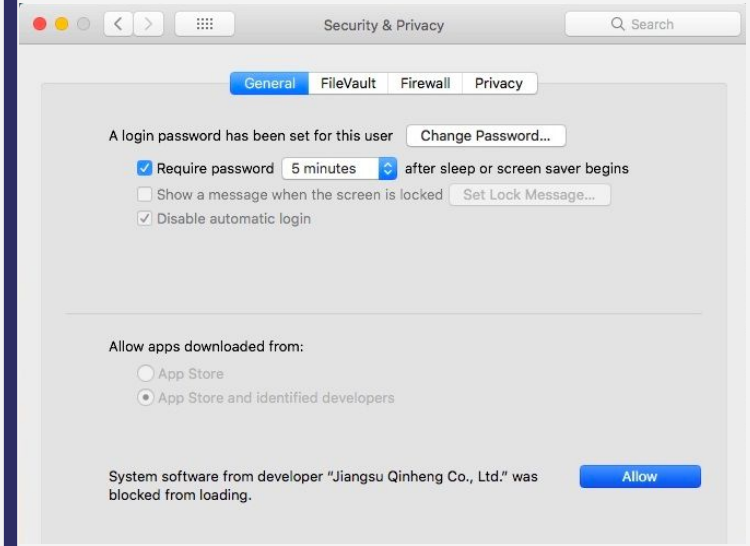
Troubleshooting driver continued

Heads up! Depending on your settings, you may need to adjust your **Security & Privacy** settings to allow the CH340 drivers to function. A window may pop up indicating that the drivers have been block as shown in the image below



Driver

If you receive a window that indicates that the system extension is blocked, you'll need to open a search with Spotlight by pressing ⌘ (**Command**) + space bar (**Space Bar**). Type **Security & Privacy** and click on the "**Allow**" button to enable the CH340 drivers.

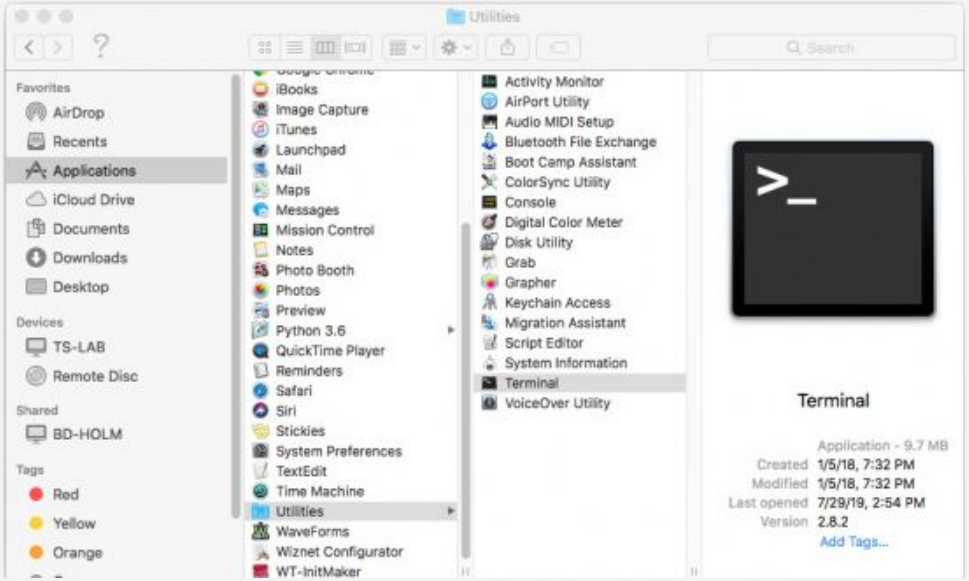


Driver Verification

To verify that your driver is working, you should see a difference in the following pictures after plugging the CH340 to a USB port.

Command Line

Open the Terminal by heading to **Applications** > **Utilities** > **Terminal** if the program is not open yet.

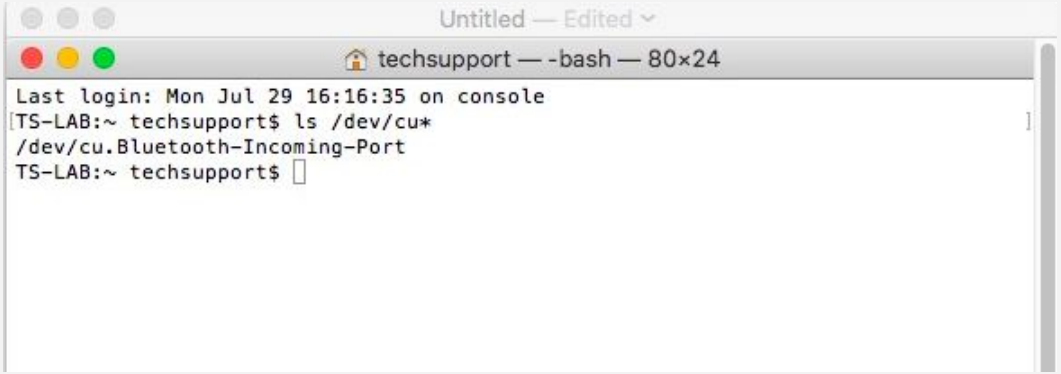


Driver Verification Continued

Next, run the following command:

```
ls /dev/cu*
```

A list of devices connected to your Mac's COM ports will be displayed as a result. Assuming that the CH340 is not connected to your computer, you should see something similar to the image.



Driver Verification Continued

Connect the CH340 to one of your Mac's COM ports. Check for the following changes (your board may show up under a different device name). The CH340 should show up as **/dev/cu.wchusbserial*****. Depending on your computer, the COM port may show up as a different number.