**Eye tracker**

Design of glasses:

1. Use of normal glasses and attaching an arm for the sensor support.
2. Designing or choosing a 3d model and printing the whole glasses with an arm support.

Methods for eye tracking:

1. Using 4 QTR-1A Reflectance Sensor placed at both sides, top and bottom of eye, we can determine the position of eyeball. But this would result in lack of visibility for the user.
   * <https://create.arduino.cc/projecthub/H0meMadeGarbage/eye-motion-tracking-using-infrared-sensor-227467>
2. Use of IR filter removed camera module to record eye movement. A pre trained model can be used to identify the position of the eye. But to do so we need to find a way to remove the pc requirement for running the model.

Finalization:

Method 2 is selected since it allows the user to see his surroundings much clearly.

Updates:

Base of eye tracking software is developed and looking forward for improvements. Refer the provided video for understanding the code:

<https://www.youtube.com/watch?v=-jFobb6ARc4&list=PLJ958Ls6nowUwRXHUcFwZy2CT0naMULR3>

(7/12/2021): Raspberry pi 4 is selected as microcontroller to remove the pc requirement for running the python model.

Improvements needed:

Tracker need to detect up and down movement of iris. Currently it detects only left, right and blink.

Update (7/12/2021): tracker is capable of detecting up and down movement of iris along with previous positions.

**Home Automation**

**Micro Controller:**

1. Arduino UNO
2. Raspberry Pi 4

Finalization: Raspberry Pi 4 have been selected as the microcontroller since it have better computational power and support complex models.

**Wheelchair**

**Node MCU**

**Simulations:**

Simulation can be done using protheus in raspberry pi 4.  
Refer the below video for more understanding on how to simulate camera detection using raspberry pi 4 in protheus:

<https://youtu.be/5uptP78_LAw>

**Eye Tracker Program Working:**

We use googles python library *mediapipe* for eye pupil detection. Different landmarks are tracked using this library. We select the landmarks around the eyes from *facemesh* (landmarks function for face). A horizontal and vertical line is drawn between the extreme sides and up and down of both eyes. The distance of these 2 lines are calculated using Euclidian Formulae.

*d = √ [(x22 – x11)2 + (y22 – y11)2]*

The position of the distance of the horizontal and vertical lines are used to estimate whether the eye blinked or not. When the ratio between horizontal and vertical line is greater than 4 the program detects the eye as blinked.

For position of pupils, the image of eye is gone through different filtration process and the binary image of the eye is obtained. From that image, the black pixels position indicate the position of pupil. In order to identify which zone the black pixels are, the horizontal line of eye is divided into 3 zones.

*hz = hd/3*

And vertical line of eye is divided into 2 zones.

*vz = vd/2*

Now the different zones are

Centre Zone = *[hz : hz + hz]*

Left Zone = *[0 : hz]*

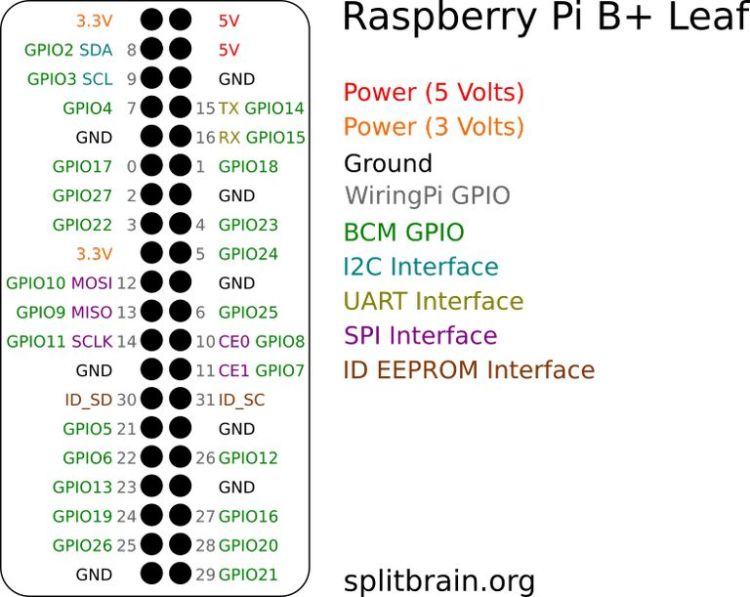
Right Zone = *[hz + hz : hd]*

Upper Zone = *[vz : vd]*

Lower Zone = *[0 : vz]*

When the black pixels enters these zones we get the positons of the pupil.

Raspberry Pi 4 Pins



**Using motors**

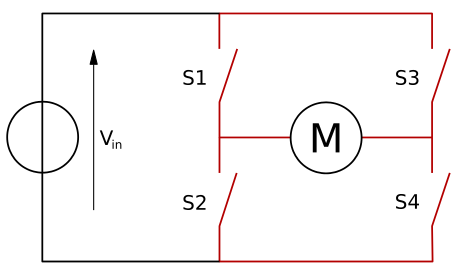
Motors are great for physical computing: they allow you to turn a wheel forwards and backwards, or make something spin around.

A motor can’t be controlled directly from the Raspberry Pi’s GPIO pins, because it needs a variable supply of 5 volts. This means you need to power it separately. However, motor controller add-on boards can be used to provide this functionality.

In this guide, you’ll be controlling two motors from your Raspberry Pi using Python on the desktop. First, it’s best just to learn how to control the motor. Then, once you have it working, you could easily use your code to drive a Raspberry Pi-powered robot by detaching the monitor, mouse, and keyboard and building a robot around a chassis.

**H Bridge**

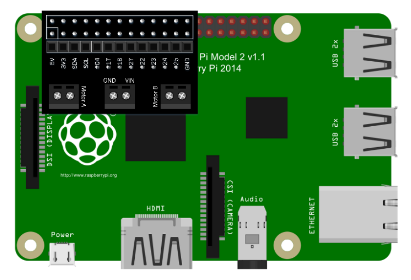
A motor can be driven forwards or backwards depending on which way around current flows through it. However, it would be awkward to have to rewire a motor, every time you want to change the direction it spins. To overcome this issue, motor controller boards include an H bridge. An H bridge uses 4 transistors to allow digital control of which way current flows through the motor. Most H bridges also contain *flyback diodes*. A flyback diode prevents the voltage spike that is generated by the motor when it is no longer powered (but still spinning) from damaging delicate electronics.



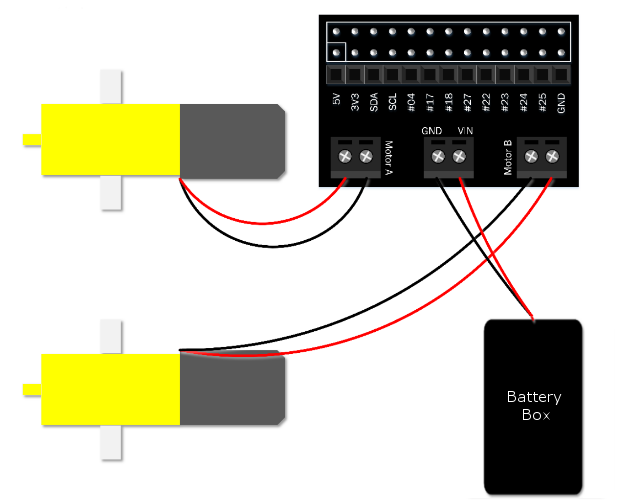
**Wiring**

You’ll need to wire up two motors and your battery pack using the motor controller.

With your Pi switched off, mount your motor controller board on the GPIO pins:



Connect a battery pack to the power ports of the motor controller, connecting the positive (red) battery wire to the positive (+) power terminal on the motor controller, and the negative (black) battery wire to the negative (-) power terminal on the motor controller, and connect two motors:



**Robot class**

If you had a robot with two wheels you would want to control the two motors together, rather than separately, just like you did for the two pins of each motor. Luckily, there’s also a Robot class in GPIO Zero.

* Import the Robot class:

from gpiozero import Robot

* Now create a Robot instance using the pin numbers for each motor:

robot = Robot((4, 14), (17, 27))

Note: to make it easier to see which pin is which, you can use Robot(left=(4, 14), right=(17, 27)) for future reference.

* Now drive one of the motors forward using the following code:

robot.forward()

Both motors should now be driving forwards.

* And backwards:

robot.backward()

Both motors should now be driving backwards.

* Try reverse a few times:

robot.reverse()

robot.reverse()

robot.reverse()

* Or try half speed:

robot.forward(0.5)

* That’s not all! What would happen if the left wheel went forwards and the right wheel went backwards? The robot would turn right. Try it:

robot.right()

* Then try this:

robot.left()

* Now stop the robot:

robot.stop()