|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| This worksheet will teach you how to use three different types of sensors with the GPIO pins of the raspberry pi. You will learn how to wire each one up, and read it’s value(s) in python code.   |  |  | | --- | --- | | Sensors worksheet | | | Project | Line detector | | Description | In this project you will learn how to read a digital line detector/reflectance sensor |  |  |  |  |  | | --- | --- | --- | --- | | Tools required | | | | | □ Raspberry Pi + SD card | | □ 4tronix digital line sensor |  | | □ Keyboard |  | □ 3 x f/f jumper wires |  | | □ Monitor + HDMI cable | |  |  | | □ Power supply |  |  |  |   Connecting line detector:  Firstly, we will connect the line sensor to the Raspberry Pi like so:   |  | | --- | |  | | Why connect it like this? | | Use female to female (hole/slot both ends) for all of these connections. Make sure pin is fully slid into wire, firmly.  We connect the G (meaning ground) pin of the line sensor to the GND (meaning ground) pin of the Raspberry Pi. On the sensor, that is the first pin along, and on the pi, the GND pin I used was Pin #20. We also connect V+ (meaning voltage+) pin on the line sensor to 3V3 (3.3 volts) on the Pi. These pins are to power it.  We connect S (sense/signal) of the line sensor to any GPIO pin. I chose the last pin on the Pi (Pin #26). This will get one reading (either high (1 or true) or low (0 or false), meaning 3.3 volts or 0 volts) when the sensor is pointing towards a dark, non reflective (e.g. black) surface, and the opposite reading when pointing towards a light, reflective (e.g. white) surface. | | Adjusting sensitivity of line sensor: | | You should all have a white piece of paper with a black line down the middle. We will use the white as our reflective and black as non-reflective surface to detect. We need to adjust the line detectors sensitivity so that the white and black get different readings.  Adjust the screw (potentiometer) in the middle of the line detector so that the red light is lit when the black line is above the sensor, but isn't lit when the white is above the sensor.  The sensor part is the end of the line sensor opposite the pins with the two bulbs. You should hold the paper between 1 and 3 centimeters away from the sensor. | |  |  |  | | --- | | Code | | We know that the line sensor’s sense pin will be HIGH (meaning at 3.3v) when the line sensor sees a dark, non-reflective surface (like the black line). The Pi should read the sense pin (Pin 26) as high during this time, so we will make our script print ‘The sensors is seeing a black/dark/non-reflective surface’, when the pin reads HIGH/1/TRUE. We will print the opposite when the reading isn’t that (else)  #Import the time module so we can make out program wait for #a length of time  import time  #Import this module to control the GPIO pins and call it #GPIO for the rest of our code  import RPi.GPIO as GPIO  #We are using this way to number our pins which matches the #physical pin numbers  GPIO.setmode(GPIO.BOARD)  #Set pin 26 as an input so we can read it’s value  GPIO.setup(26,GPIO.IN)  try:  #Repeat the next indented block forever  while True:  #If the sensor is HIGH, print the following  if GPIO.input(26)==1:  print(‘The sensors is seeing a black/dark/non-reflective surface’)  #If not (else), print the following  else:  print(‘The sensors is seeing a white/light/reflective surface’)  #Wait a second then do the same again  time.sleep(1)  #If you press CTRL+C, cleanup and stop  except KeyboardInterrupt:  GPIO.cleanup()  sys.exit() | |  | | 1. Create file line.py by running the following command in the command line (either open LXTerminal in the graphical desktop, or log in to the kernel): nano line.py  3. Enter the code above code (there’s no need to copy the lines that start with a ‘#’, they’re just there to explain the code to you!  Once complete “Ctrl + x” then “y” then “enter”  4. To run the python code “sudo python line.py” << watch the output, to close, press Ctrl+C | | Sensors worksheet | | |  |  |  | | --- | --- | | Sensors worksheet | | | Project | Ultrasonic distance measurement | | Description | In this lesson you will use an HR-SC04 to measure real world distances |  |  |  |  |  | | --- | --- | --- | --- | | Tools required | | | | | □ Raspberry Pi + SD card | | □ HR-SC04 Ultrasonic module |  | | □ Keyboard |  | □ Mini 170-pt breadboard |  | | □ Monitor + HDMI cable | | □ 8x Male to female jumper wires |  | | □ Power supply |  | □ 330ohm resistor |  | |  |  | □ 470ohm resistor |  |   Connecting HR-SC04   |  | | --- | | Firstly, connect the module like so using the wires, resistors and breadboard: | | Why connect it like this? | | Use male to female jumper wires to connect each pin of the sonar to a separate row of the breadboard. Connect the row with the VCC connection to the Pi’s 5v pin (Pin #2). We need to connect it to 5v rather than 3v3 because the sonar module needs more than 3.3 volts. Connect the row with the connection to the sonar’s ground to the Pi’s ground (Pin #14). Connect the row with trigger to a GPIO (Pin #11). Connect the row with the connection to echo to another row with a 330-ohm resistor to another row. Connect a GPIO (Pin #12) to that row, and also connect that row to the ground connection row with the 470-ohm resistor. We connect the echo pin to a GPIO with resistors and ground because the module uses a +5V level for a “high” but this is too high for the inputs on the GPIO header, which only like 3.3V. In order to ensure the Pi only gets hit with 3.3V we can use a basic voltage divider. This is formed with two resistors. |  |  | | --- | | Code | | We will now code it. The trigger pin is what we set to high to send a pulse of sound out of the module. We will make it high for 1us for a 1us pulse of ultrasonic sound. We will then wait until the pulse returns, and time how long the echo pin is high after the pulse has returned. The echo pin should stay high however long it took the pulse to return. We work out from the pulse length, the distance with the following formulae:  Distance (a variable in our code that stored the time of the pulse in seconds) = elapsed (the variable we store the length of the pulse in seconds in) \* 34300 (speed of sound cm/s)  Half the distance as the pulse had to travel to the object and back and we only want one way  Distance=distance/2  print "Distance : %.1f" % distance #print the distance to 1 decimal place | | #!/usr/bin/python  # sonar.py  # Measure distance using the ultrasonic module  #  # Author : Zachary Igielman  # Import required Python libraries  import time  import RPi.GPIO as GPIO  # Use physical pin numbers  GPIO.setmode(GPIO.BOARD)  # Define GPIO to use on Pi  GPIO\_TRIGGER = 11  GPIO\_ECHO = 12  print "Ultrasonic Measurement"  # Set pins as output and input  GPIO.setup(GPIO\_TRIGGER,GPIO.OUT) # Trigger  # Set trigger to False (Low)  GPIO.output(GPIO\_TRIGGER, False)  # Allow module to settle  time.sleep(0.5)  # Send 10us pulse to trigger  GPIO.output(GPIO\_TRIGGER, True)  time.sleep(0.00001)  GPIO.output(GPIO\_TRIGGER, False)  start = time.time()  count=time.time()  GPIO.setup(GPIO\_ECHO,GPIO.IN) # Echo  while GPIO.input(GPIO\_ECHO)==0 and time.time()-count<1:  start = time.time()  while GPIO.input(GPIO\_ECHO)==1:  stop = time.time()  # Calculate pulse length  elapsed = stop-start  # Distance pulse travelled in that time is time  # multiplied by the speed of sound (cm/s)  distance = elapsed \* 34300  # That was the distance there and back so halve the value  distance = distance / 2  print "Distance : %.1f" % distance  # Reset GPIO settings  GPIO.cleanup() | | 1. Create file sonar.py by running the following command in the command line (either open LXTerminal in the graphical desktop, or log in to the kernel): nano sonar.py  3. Enter the code above code (there’s no need to copy the lines that start with a ‘#’, they’re just there to explain the code to you!  Once complete “Ctrl + x” then “y” then “enter”  4. To run the python code “sudo python sonar.py” << watch the output, to close, press Ctrl+C | |  |   Sensors worksheet | |
| Project | MMA7455 i2c accelerometer |
| Description | In this project you will learn how to read an MMA7455 accelerometer using i2c to communicate with the Pi |

|  |  |  |  |
| --- | --- | --- | --- |
| Tools required | | | |
| □ Raspberry Pi + SD card | | □ MMA7455 accelerometer module |  |
| □ Keyboard |  | □ 5 x f/f jumper wires |  |
| □ Monitor + HDMI cable | | **□** Ethernet cable/USB wireless dongle already set up for the internet |  |
| □ Power supply |  | **□** Internet router |  |

**Connecting accelerometer:**

Firstly, we will connect the accelerometer to the Raspberry Pi like so:

|  |
| --- |
|  |
| Why connect it like this? |
| Use female to female (hole/slot both ends) for all of these connections. Make sure pin is fully slid into wire, firmly.  We connect the VCC pin of the accelerometer to 3v3 on the Pi (pin #1) and GND of the accelerometer to ground of the Pi (pin #6) to power the accelerometer with 3v3. The accelerometer can work with 3v3, and by using this voltage; we do not need a level converter as the Pi’s GPIOs work with 3.3 volts.  We connect CS of the accelerometer to another 3v3 of the Pi (Pin #17) to set it logically high. By making CS logically high, we tell the accelerometer that we would like to use i2c to communicate with it. If we set it to low and connected it to ground, it would assume we want to use SPI, an alternative way of communicating to i2c.  We connect SDA of the module to SDA of the pi (pin #3) and SCL of the module to SCL of the pi (pin #5) |
| Adjusting raspbian to work with the accelerometer: |
| On the Pi, I2C is disabled by default. To enable it, we need to change a file. To open this file, run the commands:  sudo nano /etc/modprobe.d/raspi-blacklist.conf  In this file, there is a comment, and two lines. Add a hash before the I2C line, to comment it out.  It should look like this afterwards:  # blacklist spi and i2c by default (many users don't need them)  blacklist spi-bcm2708  #blacklist i2c-bcm2708  Once complete “Ctrl + x” then “y” then “enter”  The next thing to do is add the I2C module to the kernel. Run the command:  sudo nano /etc/modules  Add the line i2c-ddev to them bottom of that file so that it looks like this:  # /etc/modules: kernel modules to load at boot time.  #  # This file contains the names of kernel modules that should be loaded  # at boot time, one per line. Lines beginning with "#" are ignored.  # Parameters can be specified after the module name.  snd-bcm2835  i2c-dev  Once complete “Ctrl + x” then “y” then “enter”  We need to install two packages, install them with this command:  sudo apt-get install i2c-tools python-smbus  If that fails, try running:  sudo apt-get update  and try again  To configure the software, we will add the Pi user to the I2C access group, by running the command  sudo adduser pi i2c.  Now run  sudo reboot  to reboot, and test the new software.  To test the software, run the command  i2cdetect -y 0  to see if there is anything connected.  If the MMA7455 accelerometer module is connected correctly and working, it should respond with this:       0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  00:          -- -- -- -- -- -- -- -- -- -- -- -- --  10: -- -- -- -- -- -- -- -- -- -- -- -- -- 1d -- --  20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  70: -- -- -- -- -- -- -- --  If nothing is connected or it is incorrectly connected, it will respond with this:  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  00:          -- -- -- -- -- -- -- -- -- -- -- -- --  10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  70: -- -- -- -- -- -- -- --  If this happens, don’t panic. Try this command:  i2cdetect -y 1  If that responds as if the MMA7455 accelerometer module is connected correctly, then it just means you have a new raspberry pi where the i2c bus is 1 rather than 0 (my code works with either as it detects which is correct before connecting).  If both i2cdetect -y 1 and i2cdetect -y 0, respond with this:  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  00:          -- -- -- -- -- -- -- -- -- -- -- -- --  10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  70: -- -- -- -- -- -- -- --  Then there’s something wrong with your hardware. Try wiring up the module again. |

|  |
| --- |
| Code |
| #!/usr/bin/python  #import necessary modules  import smbus  import time  import os  import RPi.GPIO as GPIO  # Define a class called Accel  class Accel():  myBus=""  #work out which version raspberry pi it is and setup the i2c bus #apropriately  if GPIO.RPI\_REVISION == 1:  myBus=0  elif GPIO.RPI\_REVISION == 2:  myBus=1  b = smbus.SMBus(myBus)  def setUp(self):  self.b.write\_byte\_data(0x1D,0x16,0x55) # Setup the Mode  self.b.write\_byte\_data(0x1D,0x10,0) # Calibrate  self.b.write\_byte\_data(0x1D,0x11,0) # Calibrate  self.b.write\_byte\_data(0x1D,0x12,0) # Calibrate  self.b.write\_byte\_data(0x1D,0x13,0) # Calibrate  self.b.write\_byte\_data(0x1D,0x14,0) # Calibrate  self.b.write\_byte\_data(0x1D,0x15,0) # Calibrate  def getValueX(self):  return self.b.read\_byte\_data(0x1D,0x06)  def getValueY(self):  return self.b.read\_byte\_data(0x1D,0x07)  def getValueZ(self):  return self.b.read\_byte\_data(0x1D,0x08)  MMA7455 = Accel()  MMA7455.setUp()  for a in range(10000):  x = MMA7455.getValueX()  y = MMA7455.getValueY()  z = MMA7455.getValueZ()  print("X=", x)  print("Y=", y)  print("Z=", z)  time.sleep(0.5)  os.system("clear") |
|  |
| 1. Create file MMA7455.py by running the following command in the command line (either open LXTerminal in the graphical desktop, or log in to the kernel): nano MMA7455.py  3. Enter the code above code (there’s no need to copy the lines that start with a ‘#’, they’re just there to explain the code to you!  Once complete “Ctrl + x” then “y” then “enter”  4. To run the python code “python MMA7455.py” << watch the output, to close, press Ctrl+C |

Now you know how to connect some different types of sensors to the Pi, and read them with the Pi, you can integrate them into projects or transfer what yoh learnt here for another sensor.

Here are some ideas of what you can do with each sensor.

Line detector:

CREATE A LINE FOLLOWER – use two line followers next to each other to follow a line, controlling motors to go staright when the lines in the middle and turn appropriately when it’s not

Ultrasonic diatnce sensor:

CREATE A DIGITAL RULER – make a program that keeps reading and updating the distance, you could make it portable with batteries and use an LCD for a screen

CREATE AN OBJECT DETECTING ROBOT – this robot culd drive about autonomously, and could stop or drive round objects when it gets too close

Accelerometer:

CREATE A DIGITAL SPIRIT LEVEL – keep measuring an outputting the readings from each axis, you could make it portable with batteries and use an LCD for a screen

CREATE A SELF BALANCING ROBOT – make a two wheeled robot that stands upright because when it starts leaning, it detects it with the accelerometer and drives in that direction to stop it falling and correct it, you could then pull it around on a lead and it’ll follow you

**Contact me if you have any trouble following this tutorial, want help with a project or want to tell me about your project.**

**Twitter: @ZacharyIgielman**

**Email: ZacharyI123@gmail.com**