Workshop 8: Introduction to Sense HAT

Jack Kelly

Learn the features of the Sense HAT add-on board and how to program them using the Python API.

Difficulty: Introductory workshop

Contents

0 Introduction

The *Sense HAT* is an add-on board for the Raspberry Pi designed especially for the Astro Pi mission, which now sent several Raspberry Pi's into orbit onboard the International Space Station.

The Sense HAT is a single board which adds a variety of input sensors, as well as a programmable 8x8 LED matrix to the Raspberry Pi.

This workshop will teach you how to write Python programs to access the features of the Sense HAT, using the Python API that is installed in Raspbian.

This is an introductory workshop, all of the Python concepts will be taught from scratch.

How to use these booklets

The aim of these booklets is to help you attempt these workshops at home, and to explain concepts in more detail than at the workshop. You don’t need to refer to use these booklets during the workshop, but you can if you’d like to.

When you need to make changes to your code, they’ll be presented in *listings* like the example below. Some lines may be repeated across multiple listings, so check the line numbers to make sure you’re not copying something twice.

helloworld.py

```python
#This is a comment
while True:
        print("Hello, World!")
```

Occasionally, something will be explained in greater detail in asides, like the one below. You can read these as you wish.

Comments

Red lines beginning with number signs (#) are called comments. They are ignored by the program, so we can use them to label pieces of code.

Programming with Python

We’ll be coding in Python for this workshop, and we’ll be using a program called IDLE to write our programs. It is an ‘IDE’, containing a text editor to write our programs, as well as a Python ‘Shell’ where we will run them.

To start IDLE, go to the main menu from the Raspberry Pi’s desktop, then click ‘Programming → Python 3’ IDLE will start after a couple of seconds.

Everything else

These booklets were created using LaTeX, an advanced typesetting system used for several sorts of books, academic reports and letters.

If you’d like to have a look at using LaTeX, We recommend looking at TeXstudio, which is available on most platforms, and also in the Raspbian repository.

To allow modification and redistribution of these booklets, they are distributed under the CC BY-SA 4.0 License. Latex source documents are available at http://github.com/McrRaspJam/booklet-workshops

If you get stuck, find errors or have feedback about these booklets, please email me at: jam@jackjkelly.com

1 Weather station

The Sense HAT has a pair of environmental sensors, for temperature, atmospheric pressure, and relative humidity. We'll start with a simple script, which probes on of these sensors, and prints out the data as text on screen.

weather.py

After opening IDLE, click 'File → Open...'. From the Home folder 'pi', where you currently are, navigate to 'workshops → 008_senseHAT → morning' and open 'weather.py'.

weather.py

```
#setup the sense HAT
from sense_hat import SenseHat
hat = SenseHat()

#Write your code below
```

The first two lines of the script tell Python that this program will use the Sense HAT. The first line loads the 'API', which allows us to program the Sense HAT in Python, and the second line runs some set-up procedures for our program.

First, we'll add a basic print function, as a welcome message when running the program.

print()

A **print function** will 'print' text on-screen when the program is run.

We put what we want to be printed inside the parentheses, text must be surrounded by quote marks.

```

#Say hello
print("Manchester Raspberry Jam weather station")
```

You can try running your program now, by first saving, then going to 'Run → Run Module'. The window should switch back to the one that first opened when we ran IDLE, and our text should turn up after a couple of seconds.

The print statement is a simple start to our program, and we'll come back to them shortly, but now, let's try pulling data from the Sense HAT sensors:

```

#get the temperature
temperature = hat.get_temperature()
```

We have used the Sense HAT function 'hat.get_temperature()' to get the temperature, in Celsius, from the sensor. It returns it as a number which we have stored in the variable 'temperature'.

Function

A function calls a piece of code that somebody else has written for us.

A function can be identified by the pair of parentheses. print(), str() and round() are examples of functions.

Some functions output, or 'return' a value, they can also take inputs, called 'parameters' which are placed within the parentheses.

Variable

When we store a number or other value in our program for later use, it is called a variable.

We can give a variable any name we want, and we set a variable using: 'variablename = value'

We need to print out the number so we can see it when we run the program. We'll use another print function, but instead of text, surrounded by quotes, we'll use our variable 'temperature' instead:

```python
temperature = str(temperature)
print(temperature)
```

str() is used to turn a number into a 'string', the text representation of our number, as print() cannot print numbers without conversion.

When you run this program, you'll see that the temperature is printed out with a high number of decimal points. (e.g. 29.852941513061523) This is a lot more than we need, so we can round the number down to less decimal places using another function round().

round has two parameters, which we separate within the parentheses using commas, round(number, decimal places). In our code that looks like this:

```python
temperature = hat.get_temperature()
temperature = round(temperature, 1)
temperature = str(temperature)
print(temperature)
```

The program now outputs a temperature which is easy to read. One last thing we can do is add a label and unit when we print out the temperature.

We can combine strings in the print function by adding them together. Instead of mathematical addition like numbers, Python will perform a 'string concatenation'; basically, putting one string after another to make one big string.

```python
print("Temperature: "+temperature+" Celsius")
```

Make sure each 'string' of text is encapsulated in its own set of quotes, like above.

Adding Humidity

Once we've gotten our heads around the temperature, we can use exactly the same steps for the humidity and pressure.

Humidity

Humidity is a measure of the water vapour in the air, and is measured as a percentage %.

Water holds a lot of heat energy, which can increase temperatures. Humid air also slows down the rate at which we can sweat, which can make hot temperatures even more unbearable![?]

Instead of using the get_temperature() function, we just need to use the 'get_humidity()' function instead. Everything else is the same, so we can copy and paste our code, and change the variable names and text labels.

```

#get the humidity
humidity = hat.get_humidity()
humidity = round(humidity, 1)
humidity = str(humidity)
print("Humidity: "+humidity+"% RH")
```

Adding Atmospheric Pressure

Finally, we can do atmospheric pressure. This time the function is called get_pressure(). Have a go at this implementing this last measurement yourself.

Atmospheric pressure

Atmospheric pressure is caused by the weight of air from the atmosphere, and is measured in millibars.

As altitude increases, the amount of air in the atmosphere starts to decrease (referred to as thin air), resulting in a lower pressure. Pressure is also affected by temperature, with local differences affecting winds. Low pressure is associated with high cloud cover and precipitation. [?]

2 G-Force

As well as the environmental sensors, the Sense HAT has a set of inertial momentum unit (motion and orientation) sensors. These are used primarily for determining the orientation of your Pi, which we'll use this afternoon.

For now, we'll use the raw accelerometer data to measure the 'G-force' under various scenarios.

G-forces

G-forces are a measure of acceleration acting on an object, relative to 1G, the acceleration due to the force of gravity. A still object like our Pi is experiencing 1G downwards.

Here are a few interesting examples of different situational G-forces:[?]

Example
Gravity on the Moon
Gravity on the Earth
Space Shuttle launch
Top Fuel dragster
High-G roller coaster
Jet fighter manoeuvre
Human on Rocket sled test
Bullet in gun barrel
Proton in LHC

gforce.py

Click 'File → Open...'. From the home folder, go to 'workshops → 008_senseHAT → morning' and open 'gforce.py'.

Like before, we'll be calling an API function to fetch the sensor data. get_accelerometer_raw() returns a vector, with x, y and z axis 'components'.

```
#setup the sense HAT
from sense_hat import SenseHat
hat = SenseHat()

#get acceleration data in G's
acceleration = hat.get_accelerometer_raw()
```

We've seen vectors before, used in Minecraft: Pi Edition's APIs, but this time the function has returned the three components within a 'dictionary' data structure, rather than a list, as used previously. We'll need to know how to access the individual components to round them.

Aside

A dictionary is quite similar to a list in Python. A list is a group of variables, each referenced by a number. A vector (13, 24, 17) in a list would look like:

Index	Value
0	13
1	24
2	17

The y component would be accessed with: Vector[1]

A Dictionary uses text indices instead of numerical ones. This may be more intuitive to you, but it also makes dictionaries great for data with no inherent ordering. If you want to use a dictionary in your own programs, you can define one like so: dictionary = {’x’: 13, ’y’: 24, ’z’: 17}

Which means a dictionary for that vector would look like:

Index	Value
’x’	13
’y’	24
’z’	17

The y component would be accessed with: Vector[’y’]

We can now access each component and round it, returning it back to the dictionary

```
#round values to 1.d.p.
acceleration[’x’] = round(acceleration[’x’], 1)
acceleration[’y’] = round(acceleration[’y’], 1)
acceleration[’z’] = round(acceleration[’z’], 1)
```

We now have rounded all of the 3 G-Force Values, and we can print them. Luckily Python will neatly print out all of the components of a data structure like a dictionary, we can just to pass it to the print function:

```
print(acceleration)
```

You can now run your program. Make sure your Raspberry Pi is flat on the desk, can you figure out which axis is the up/down axis, based on the numbers?

One final modification to our program. It’d be nice to measure G’s changing as we move the Raspberry Pi about, so we’ll encapsulate our program in an infinite loop, like so:

```
while True:
        #get acceleration data in G’s
        acceleration = hat.get_accelerometer_raw()
        #round values to 1.d.p.
        acceleration[’x’] = round(acceleration[’x’], 1)
        acceleration[’y’] = round(acceleration[’y’], 1)
        acceleration[’z’] = round(acceleration[’z’], 1)
        print(acceleration)
```

Rerun the program, and experiment with the different G-forces you can create.

By rotating your pi to stand on different sizes, you can change the axis and polarity of gravity. By shaking your pi you can cause brief forces in directions other than that of gravity. If you were to drop your Pi on a hard surface, you might see G spikes in the 10’s or 100’s, consistent with crash forces.

3 G-Force Demonstration

We'll take a look at a few applications for the G-force sensor now. There's no coding for this section, we'll just be looking at some interesting examples.

With a small adjustment to our code, we can output to a text file, instead of to the Python shell. I chose to output the captured telemetry in CSV format.

Aside

CSV formatted data is a simple text format, where data is separated into rows, separated by columns. For example, the following CSV data:

```
time, x, y, z
0.0, 0.97, 0.03, 0.01
0.1, 0.98, 0.02, 0.01
```

is equivalent to a spreadsheet table:

time	x	y	z
0.0	0.97	0.03	0.01
0.1	0.98	0.02	0.01

Many pieces of software will accept CSV as data input, we'll look at two pieces of software, Wolfram Mathematica and Libreoffice Calc. (a spreadsheet software)

Dropping the Pi

Elevator forces are bit pedestrian, far more exciting is the prospect of Zero-G. Astronauts onboard the ISS experience 'microgravity' in orbit around the earth, but we can also experience brief moments of zero G-force on earth.

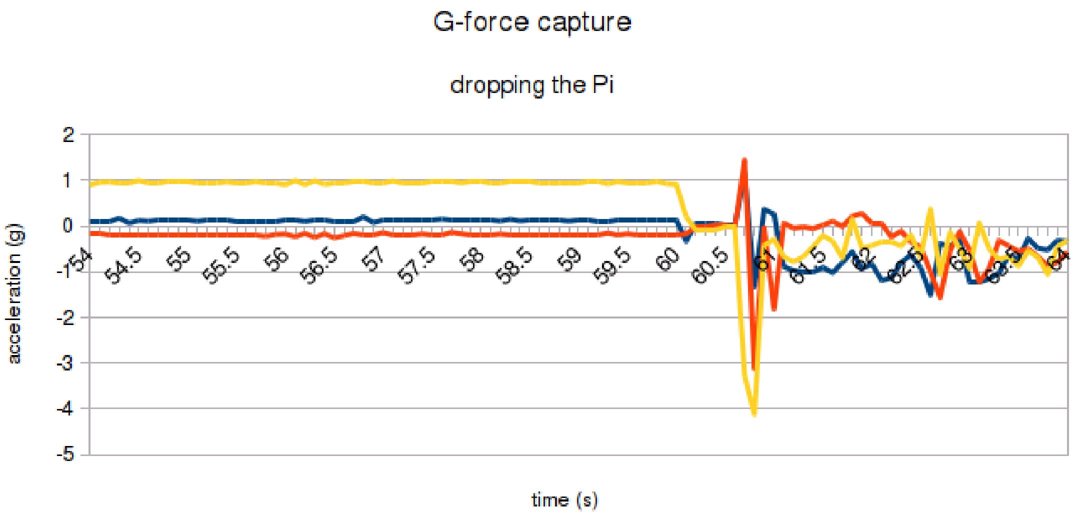
Aside

Objects in **free-fall** are accelerating with gravity, and feel no resisting forces from the ground; They measure 0G.

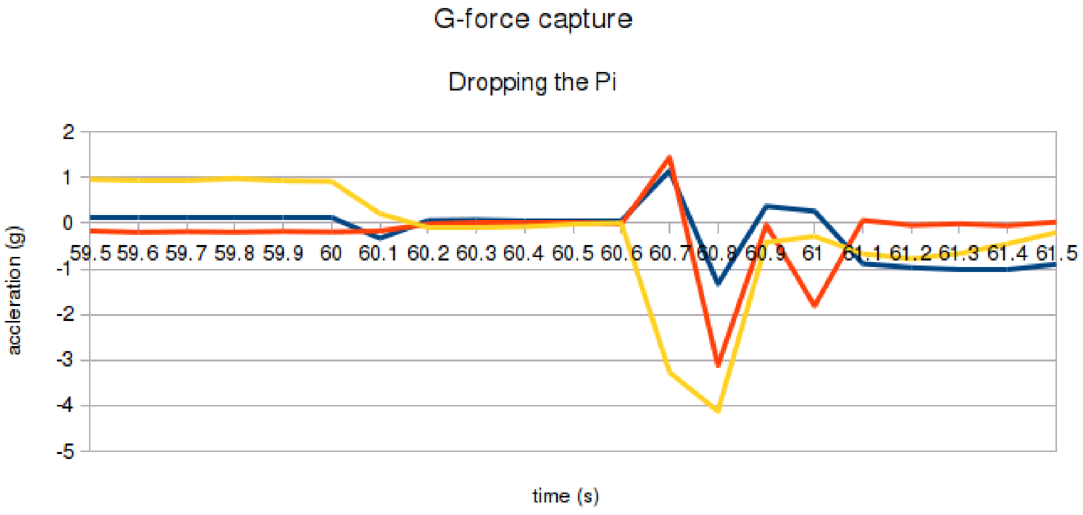
We can experience zero-G in earth, essentially by strapping ourselves to "falling" objects, such as the 'Vomit comet', a plane manoeuvre made infamous by NASA, in their astronaut training.

It stands to reason that by dropping our Pi, whilst recording G-force data, we could measure a moment of zero-G. In the hope of preventing you from destroying your Pi, we decided to test it ourselves.

We dropped the Pi from a balcony, about a storey high, and caught it from below. We imported the resulting CSV file into Libreoffice Calc, and created a graph.



You can probably see the drop point on the graph. When held on the balcony, the Z-axis shows a steady 1G under gravity, until it is dropped, where it drops to zero. The large spike is the force exerted as I catch the Pi. Let's look closer at that section:



Our recorded data looks perfect, we can see consistent 0G in free-fall, just as we predicted. What's more, by getting the duration of free-fall (which I make to be 0.5s), we can work out some more details of our drop.

There are a set of mathematical formulas for Constant acceleration, such as under gravity, which you may use in A-level Physics. Don't worry though, they're really simple when calculating gravity! One of those is as follows:

$v = u + at$

We want to find 'v', the speed at the end of the drop. 'u' is the speed at the start of the drop, which is 0. ‘a’ is acceleration due to gravity, which is always $9.81m/s^2$. ‘t’ is the time in free-fall, which we measured as $0.5s$.

$v = 0 + 9.81 \times 0.5$

$= 9.81 \times 0.5$

$v = 4.9m/s$

Which means that the Raspberry Pi was caught at over 10 miles per hour!

4 Motion controlled Snake

In workshop 7, we created a simple version of the video game Snake, on the unicorn HAT board.

In this workshop, we'll be modifying the game to make the snake move based on how we tilt the Raspberry Pi, rather than using keyboard controls.

snake.py

Click 'File → Open...'. Go to 'workshops → 008_senseHAT → afternoon' and open 'snake.py'.

Note: because the snake game was originally written for the Unicorn HAT, all of the hat functions are accessed by ***UH.<command>****, not hat.<command> as in the earlier programs.*

If you attended our previous workshop, this is the same code you completed previously.

As we are working with an already complete and playable version of the game, I have separated the control inputs into a separate function, called 'get_input'.

2

```
def get_input(direction):
        for event in pygame.event.get():
                if (event.type == KEYUP) or (event.type == KEYDOWN):
                        if (event.key == K_ESCAPE):
                                quit_game = True
                        if (event.key == K_w):
                                direction = 0
                        elif (event.key == K_d):
                                direction = 1
                        elif (event.key == K_s):
                                direction = 2
                        elif (event.key == K_a):
                                direction = 3

        return direction
```

This bit of code checks each keyboard key, and if pressed, will change the direction the snake (player) is travelling. Direction is a number, where 0 signifies up, 1 is right, 2 is down, and 3 is left.

Our version of the code will follow the same steps, but rather than using keyboard input, we will use

Our function will follow the same steps, except rather than

Rewriting get_input

First we need to determine to get the G-forces from the Sense HAT. We will store it as a variable, as we did earlier

```
def get_input(direction):
        #get accelerometer data
        force = hat.get_accelerometer_raw()
        del force[’z’]
```

Before we can return the correct direction, we need to know which direction we need to move in based on the direction of tilt. Rerun your looping G-force program, and record the G measurements for each of the 4 directions.

Remember that the HDMI port is at the "bottom" of game board, and the snake should move “downhill”, so to move up, you will tilt the top of the Pi downwards.

The directions I determined are below:

Axis	Direction
X+	1
X-	3
Y+	2
Y-	0

Then, it is simply a matter of rewriting the old function, but with appropriate x/y values for each direction:

```
        if force[’x’] < -0.3:
                direction = 3
        elif force[’x’] > 0.3:
                direction = 1
        elif force[’y’] < -0.3:
                direction = 0
        elif force[’y’] > 0.3:
                direction = 2
```

I have chosen values greater than 0.3, as we want a control ‘deadzone’, so the direction does not change erratically when we try to hold our Raspberry Pi flat.

We are more or less finished, and your game should now be playable.

A final modification I would consider making is to disallow movement changes when holding the Pi diagonally, as currently certain directions are chosen before others which may also be satisfied. I’ll leave for you, with a hint below.

Hint: change the if statement as above to an elif, and add a first if before. Use the boolean **and** and **or** operators to combine comparisons for your condition. Solution at github.com/McrRaspJam