Dr. József Sütő
Department of IT Systems and Networks

Embedded Systems Laboratory

Lab 5: Sensors on the Sense HAT

Project overview

The Sense HAT has a set of environmental sensors for detecting the surrounding conditions. It can measure orientation, pressure, temperature, and humidity. The main topic of this lab is the data acquisition from Sense HAT's sensors.

The following topics will be covered in this lab:

- Sense HAT - sensors

Technical requirements

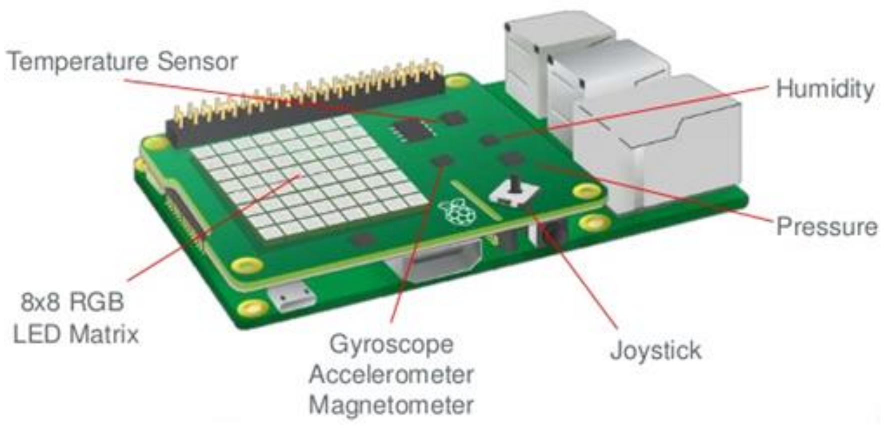
The following components are required to complete this lab:

- Raspberry Pi 3 Model B+ (MicroSD card, power supply, keyboard, mouse)
- Sense HAT
- Or you can use the emulator!

1. Sense HAT

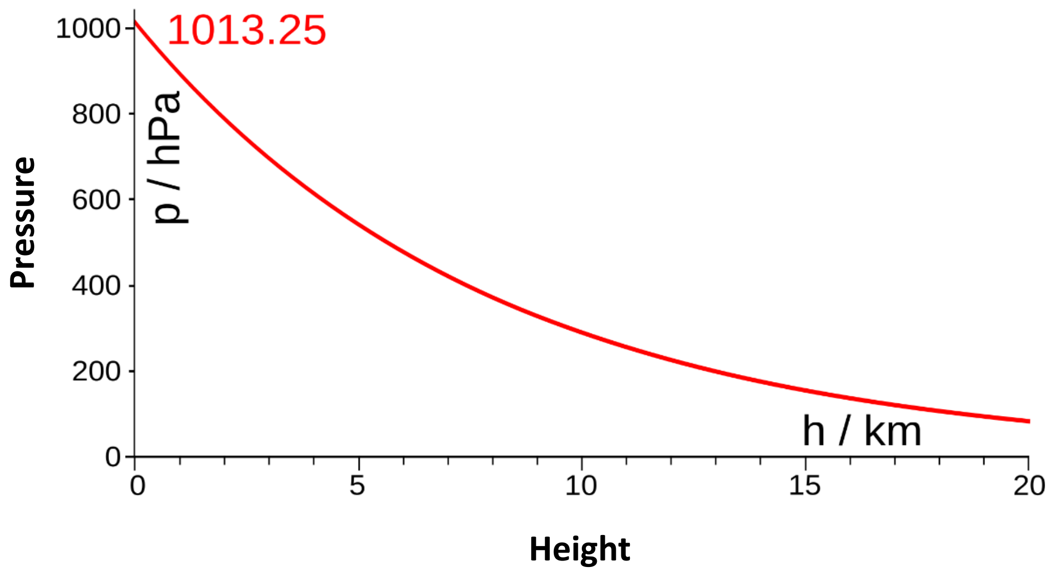
The HAT includes the following components:

- 8x8 LED matrix display
- Accelerator
- Gyroscope
- Magnetometer
- Air pressure sensor
- Temperature sensor
- Humidity sensor
- Small joystick



2. Atmospheric pressure

- Atmospheric or air pressure is the force per unit of area exerted on the Earth's surface by the weight of the air above the surface
- Atmospheric pressure is expressed in several different systems of units. With **SI** (International System of Units) units, pressure measures force per unit area - **Pascals (1 pascal = 1 newton per square meter, 1 N/m^2)**
- Standard sea-level pressure, by definition is equal to **101.325 kilopascal (kPa) = 1013.25 hPa**
- The pressure (and temperature) sensor on the HAT is **LPS25H**:
  - 260 to 1260 hPa pressure range (accuracy depends on the temperature and pressure, ~0.1 hPa under normal conditions)
  - SPI and I2C interfaces
- Pressure increases as temperatures increase and decreases as temperature cool down
- Pressure increases at lower altitudes and decreases at higher altitudes. Therefore, air pressure is highest at sea level, where air density is greatest
- Read pressure value from the HAT's sensor: **get_pressure()**



3. Humidity

- Humidity is the amount of water vapor present in air
- Humidity indicates the likelihood for precipitation, dew, or fog to be present
- The amount of water vapor needed to achieve saturation increases as the temperature increases
- Two primary measurements of humidity are widely employed: absolute and relative:
    - **Absolute humidity** describes the water content of air and is typically expressed in grams per cubic meter g/m^3
    - **Relative humidity (RH)**, expressed as a percentage. It is a measure of the amount of water vapor that air is holding compared the amount it can hold at a specific temperature. At 100% relative humidity, the air is saturated (its dewpoint)
- The humidity (and temperature) sensor on the HAT is **HTS225**:
    - 0 to 100% relative humidity range
    - Temperature accuracy: ± 0.5 °C, 15 to +40 °C
    - SPI, I2C interfaces
- Read humidity value from the HAT’s sensor: **get_humidity()**

4. Temperature

- The Sense HAT has two sensors capable of reading the ambient temperature:
    - Humidity sensor
    - Pressure sensor
- Most commonly used scales of temperature:
    - Celsius (C)
    - Fahrenheit (F)
    - Kelvin (K - the unit of temperature in SI)
- Many physical processes are affected by temperature. For example, the physical properties of materials
- Read temperature from the HAT's humidity sensor: **get_temperature()**
- Read temperature from the pressure sensor: **get_temperature_from_pressure()**
- Unfortunately, the temperature measure on the HAT is not accurate due to the heat generated by the Raspberry Pi. It can be used as a reference value only!

5. Acquire data from sensors

Read temperature, humidity and pressure values from sensors, one times in a second in an infinite loop. Print the three values in a single line into the console.

5.1. Import necessary modules:

```
from sense_hat import SenseHat

from time import sleep
```

5.2. Create an object from the SenseHat class:

```
sense = SenseHat()
```

5.3. Acquire temperature, humidity, and pressure values in an infinite loop:

```
while True:

    t = sense.get_temperature()

    p = sense.get_pressure()

    h = sense.get_humidity()
```

```
# round() - round the values to some decimal places
t = round(t, 1)
p = round(p, 1)
h = round(h, 1)

# str() – convert a value to a string
message = "Temperature: " + str(t) + "C" + " Pressure: " +
str(p) + "hPa" + " Humidity: " + str(h)   + "%"
print(message)
sleep(1)
```

6. Weather forecast (task to be solved alone)

Use simplified Zambretti's algorithm to predict weather. The input parameters of the algorithm are season, sea-level pressure, barometric tendency, hemisphere and wind direction. The algorithm is empirical and created for the northern hemisphere (in Great Britain). Analysis made on the algorithm conclude, that wind direction has little effect to the final result, so it is negligible. We can read the temperature, humidity and station pressure from the Sense HAT, but the **relative pressure** should be calculated. Using the following algorithm, we will calculate a tabulated forecast number **Z**:

1. From your measured pressure P (in hPa), temperature T (in Celsius) and the altitude h (in meter) compute the atmospheric pressure reduced to sea level P_0. For simplicity, the h value is a constant 125m.

$$P_0 = P\left(1 - \frac{0.0065 * h}{T + 0.0065 * h + 273.15}\right)^{-5.257}$$

2. The most significant factor in any weather forecast is whether the atmospheric pressure is falling, rising, or steady. A falling barometer is indicated by a drop of **1.6 millibars** within a three-hour period. If the pressure rises 1.6 millibars within three

hours, it is a rising barometer. If the pressure neither rises nor falls it is steady. For simplicity, you should compute the pressure trend for 10 seconds:

a. If the pressure is **falling** compute the forecast number as $Z = 130 - \frac{P_0}{81}$
b. If the pressure is **steady** compute the forecast number as $Z = 147 - \frac{5P_0}{376}$
c. If the pressure is **rising** compute the forecast number as $Z = 179 - \frac{2P_0}{129}$

3. Use the following look-up table to forecast weather:

Z	Forecast
1	Settled Fine
2	Fine Weather
3	Fine Becoming Less Settled
4	Fairly Fine Showery Later
5	Showery Becoming More Unsettled
6	Unsettled, Rain Later
7	Rain at Times, Worse Later
8	Rain at Times, Becoming Very Unsettled
9	Very Unsettled, Rain
10	Settled Fine
11	Fine Weather
12	Fine, Possibly Showers
13	Fairly Fine, Showers Likely
14	Showery Bright Intervals
15	Changeable Some Rain
16	Unsettled, Rain at Times
17	Rain at Frequent Intervals
18	Very Unsettled, Rain
19	Stormy, Much Rain
20	Settled Fine
21	Fine Weather
22	Becoming Fine
23	Fairly Fine, Improving
24	Fairly Fine, Possibly Showers, Early
25	Showery Early, Improving
26	Changeable Mending
27	Rather Unsettled Clearing Later
28	Unsettled, Probably Improving
29	Unsettled, Short Fine Intervals

30	Very Unsettled, Finer at Times
31	Stormy, Possibly Improving
32	Stormy, Much Rain

7. Weather logger

Collect temperature and humidity data from the Sense HAT's sensors in every 0.5 seconds until 5 seconds and log it to a *.txt* file. Thereafter, use the **matplotlib** module to display that data as a line graph.

7.1. Install the matplotlib. In the terminal issue the following command:

```
$pip3 install matplotlib
```

7.2. Install an additional module:

```
$sudo apt-get install libatlas-base-dev
```

7.3. Import necessary libraries:

```
from sense_hat import SenseHat

import matplotlib.pyplot as plt

import time
```

7.4. Define the data visualisation function which reads data from file:

```
def plot(filename):

    #define two lists

    temp_list = []

    humi_list = []


    try:

        #open the file for read

        file = open(filename, ‘r’)
```

```
                    #break file content into lines
                    lines = file.readlines()

                    #go through all lines and split lines by ‘,’
                    for line in lines:
                              values = line.split(‘,’)
                              temp_list.append(float(values[0]))
                              humi_list.append(float(values[1]))
          finally:
                    file.close()

          #create graph using the plot function from matplotlib
          # arguments of the plot function: x values, y values, line_type
          plt.plot(range(1, len(temp_list) + 1), temp_list, ‘r-’)
          plt.plot(range(1, len(humi_list) + 1), humi_list, ‘b--’)
          plt.title(‘Weather’)
          plt.xlabel(‘Measurements’)
          plt.ylabel(‘Value’)
          plt.show()
```

7.5. Data acquisition is in the main() function:

```
def main():
          sense = SenseHat()
          #time() – return the time in seconds
          start_time = time.time()
          stop_time = time.time()
```

```
        filename = ‘weather.txt’
        #open file for append
        file = open(filename, ‘a’)
        print(‘Data acquisition is starting…’)

        while stop_time – start_time < 5:
            file.write(str(sense.get_temperature()) + ‘,’ +
            str(sense.get_humidity()))
            file.write(‘\n’)
            stop_time = time.time()
        print(‘Stop data acquisition!’)
        file.close()
        plot(filename)

    if __name__ == ‘__main__’:
        main()
```

8. Using pressure value to predict altitude (task to be solved alone)

Write a Python program which tries to predict your altitude. Print the predicted value into the console window!

- Near Earth’s surface the pressure decreases by about 1.2 kPa for every 100 meters but it also depends on the temperature
- A change of 1 hPa of atmospheric pressure corresponds to a change in altitude of about 8.3 meters.

- Altitude can be calculated using the international barometric formula (in meter):

$$h = 44331 * \left(1 - \left(\frac{P}{P_0}\right)^{\frac{1}{5.2558}}\right)$$

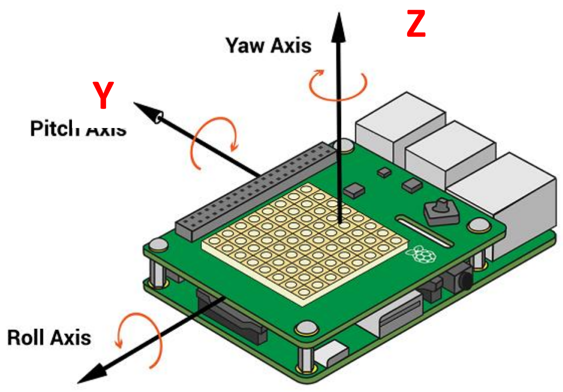
P – atmospheric pressure at current location in hPa

P_0 – atmospheric pressure at sea level in hPa (**1013.25 hPa**)

- Find altitude on map (online): https://www.daftlogic.com/sandbox-google-maps-find-altitude.htm

9. Detecting movement

- The Sense HAT has an **9 DOF** (degrees of freedom) **IMU** (inertial measurement unit) chip which includes a set of sensors that detect movement:
  - **Gyroscope**: measures rotation
  - **Accelerometer**: measures acceleration forces, can be used to find the direction of gravity
  - **Magnetometer**: measures the Earth's own magnetic field
- All objects have three axes around which they can rotate:
  - **Pitch** — rotation around y axis
  - **Roll** — rotation around x axis
  - **Yaw** — rotation around z axis
- If you know how much rotation has happened on each axis of an object, then you know which way the object is pointing.
- Access orientation data: **get_orientation()**



10. Detecting acceleration and rotation

Write a program to detect acceleration on each axis and the current pitch, roll, and yaw. Run the program and move the Sense HAT. Watch how the values change as the Sense HAT moves.

10.1. Create a SenseHat object.

10.2. Get orientation:

```
o = sense.get_orientation()
```

10.3. Get acceleration:

```
acceleration = sense.get_accelerometer_raw()
```

10.4. Extract components from orientation:

```
pitch = o["pitch"]
roll = o["roll"]
yaw = o["yaw"]
```

10.5. Extract components from acceleration and round their values:

```
x = acceleration['x']
y = acceleration['y']
z = acceleration['z']

x=round(x, 0)
y=round(y, 0)
z=round(z, 0)
```

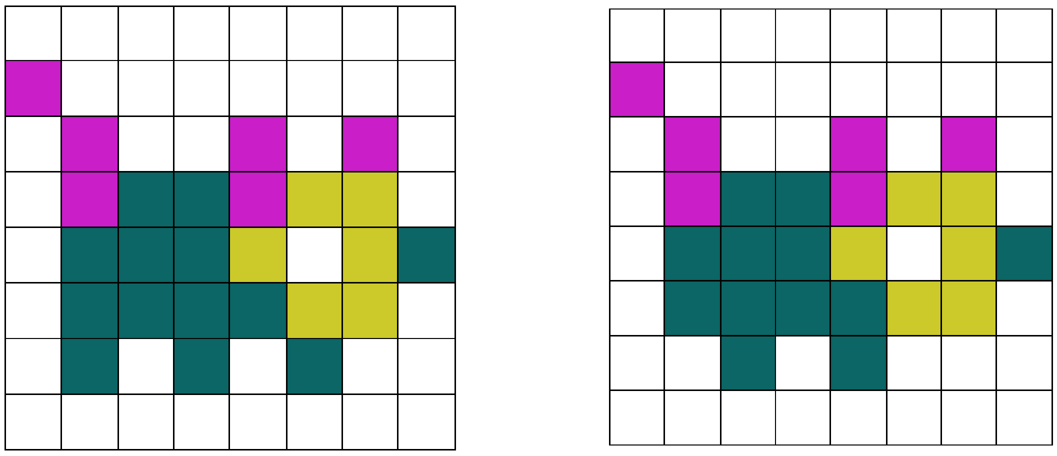
10.6. Print out all values:

```
print("pitch: ”, pitch, “ roll: ”, roll, “ yaw: ”, yaw)
print("x: ”, x, “ y: ”, y, “ z: ”, z)
```

11. Walking pet (task to be solved alone)

Write a Python program which animates a walking pet (cat)!

11.1. Design two similar pet patterns where the only difference is the leg position. The difference between leg positions helps as animate movement.



11.2. To animate the movement, you need to create a loop where the two pet patterns will be repeatedly switched every half-second. This code needs to be placed in a function which will be called if a trigger event occurs.

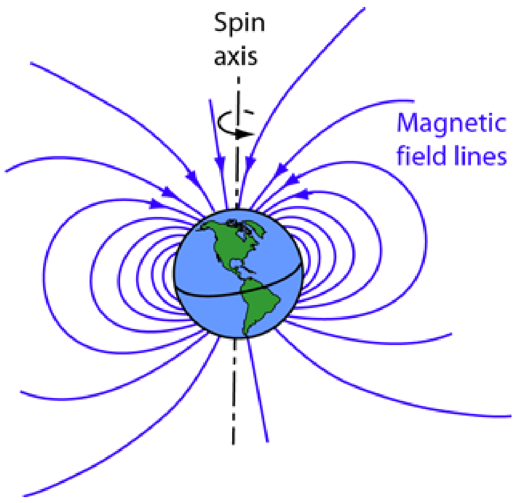
11.3. Use the Sense HAT's movement sensor to generate a trigger event. You need to read accelerometer values and calculate the length of the force with the following formula:

$$F = \sqrt{x^2 + y^2 + z^2}$$

If F is higher than a predefined threshold (trigger event), the above-described walk function will be called.

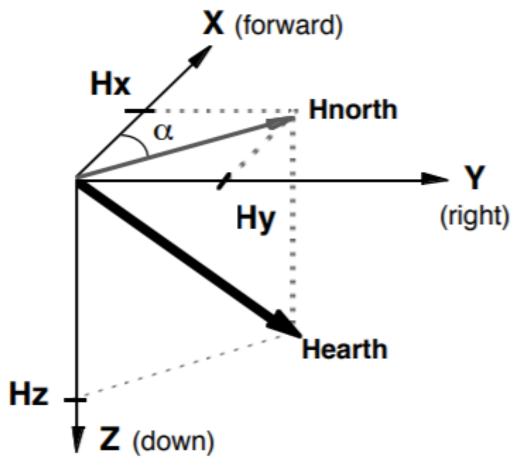
12. Electronic compass

The earth's magnetic field can be approximated with the dipole model shown below. This figure illustrates that the earth's field points down toward north in the northern hemisphere, is horizontal and pointing north at the equator, and point up toward north in the southern hemisphere. In all cases, the direction of the earth's field is always pointing to magnetic north. It is the components of this field that are parallel to the earth's surface that are used to determine compass direction.



The term magnetic north refers to the earth's magnetic pole position and differs from true, or geographic, north. True north is at the earth's rotational axis and is referenced by the meridian lines found on maps. At different locations around the globe magnetic north and true north can differ by **±25 degrees**. The difference between them at your location can be found on: *http://magnetic-declination.com/*. Since the magnetometer is fixed on the Printed Circuit Board (**PCB**), their readings change according to the orientation of the PCB. If the PCB remains flat, then the compass **heading (or azimuth)** could be computed with a two-step process:

1. Determine the Hx and Hy horizontal components of the earth's magnetic field
2. Add or subtract the proper **declination angle** to correct for true north.



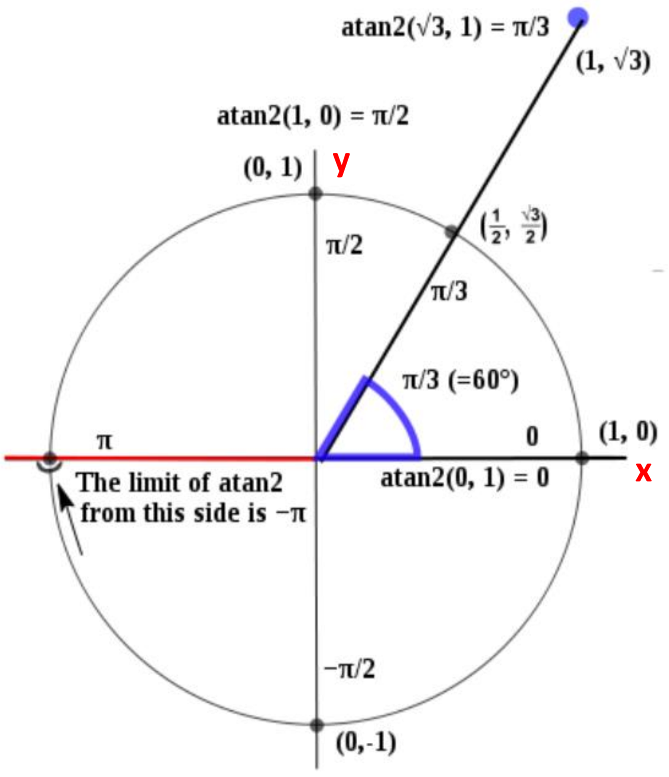
α = Azimuth or Heading

To account for the tangent function being valid over 180° and not allowing the zero-division calculation, the following formula can be used:

$$\alpha = \begin{cases} 90\ if\ (x = 0, y < 0) \\ 270\ if\ (x = 0, y > 0) \\ 360 + \arctan\left(\frac{y}{x}\right) * \frac{180}{\pi}\ if\ (y < 0) \\ \arctan\left(\frac{y}{x}\right) * \frac{180}{\pi}\ if\ (y > 0) \end{cases}$$

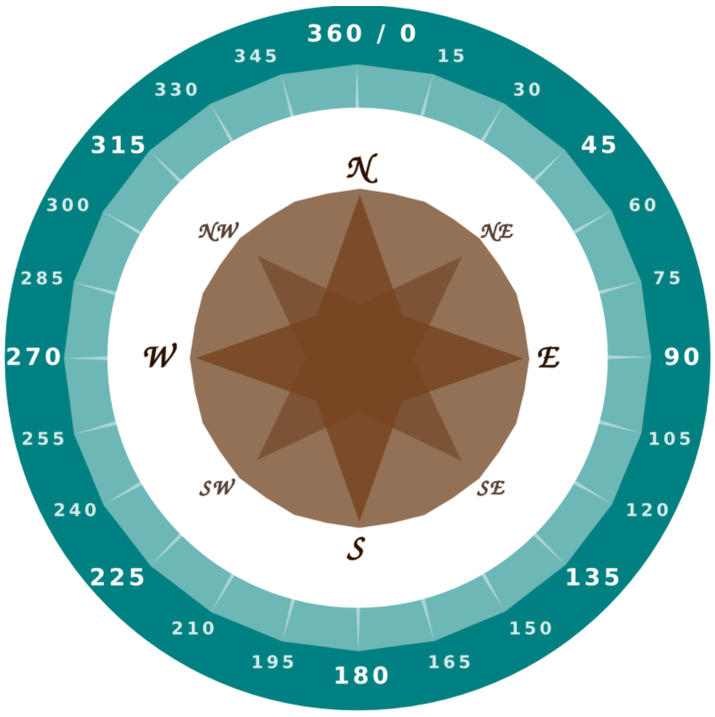
The arcus tangent function can be calculated by the **atan2(y,x)** function. It gives back the result in radian between **-π and π**. To convert radian into degree the following simple equation has been used:

$$\alpha_{degree} = \alpha_{radian} * 180/\pi$$



13. 4-piont compass

A simple four-point compass depicts the cardinal points (**N, S, E, W**). This type of compass may be used for basic automotive use where the driver needs to know the general direction of travel. For this application, the magnetic sensor can be reduced to a two-axis sensor using only the X and Y axis.



13.1. Import necessary libraries.

13.2. Create a global calibration variable:

```
calibration = True
```

13.3. Define stop function which will terminate the calibration process.

```
def stop():
    global calibration
    calibration = False
```

13.4. Use a modified version of the *plot()* function to visualize the acquired x and y values of the magnetometer. The function gives back the max and min values of x and y.

```
def plot(filename):
    #define two lists
    x_list = []
    y_list = []
```

```
try:
    #open the file for read
    file = open(filename, ‘r’)
    #break file content into lines
    lines = file.readlines()

    #go through all lines
    for line in lines:
        values = line.split(‘,’)
        x_list.append(float(values[0]))
        y_list.append(float(values[1]))
finally:
    file.close()

#max-min values
xmax = max(x_list)
xmin = min(x_list)
ymax = max(y_list)
ymin = min(y_list)
print(‘Max x: ’, xmax, ‘Min x: ’, xmin)
print(‘Max y: ’, ymax, ‘Min y: ’, ymin)

#create graph
```

```
        line1, = plt.plot(range(1, len(x_list) + 1), x_list, ‘r-’, label=’x’)
        line2, = plt.plot(range(1, len(y_list) + 1), y_list, ‘b--’, label=’y’)
        plt.xlabel(‘Measurements’)
        plt.ylabel(‘Value’)
        plt.legend(handles=[line1, line2])
        plt.show()
        return xmax, xmin, ymax, ymin
```

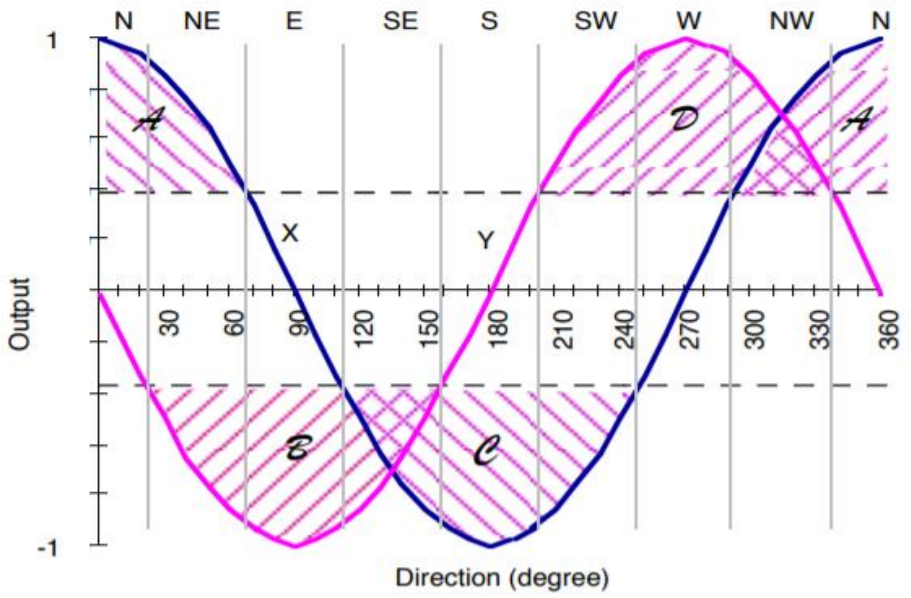
13.5. In the main there are two loops. The first is a the calibration loop. It is terminated by the joystick push. **During calibration, you have to turn around your Pi (in place on a flat surface) and than push the joystick!** Acquired data will be drawn by the plot() fucntion.

```
    def main():
        sense = SenseHat()
        filename = ‘compass.txt’
        #open file for write (rewrite its content)
        file = open(filename, ‘w’)

        sense.stick.direction_middle = stop
        print(‘Start data acquisition…’)

        #calibration process
        while calibration:
            magnet = sense.get_compass_raw()
            x = magnet[‘x’]
            y = magnet[‘y’]
            file.wtite(str(x) + ‘,’ + str(y) + ‘\n’)
        file.close()

        xmax, xmin, ymax, ymin = plot(filename)
```

13.6. Unfortunately, the surrounding environment can significantly influence magnetometer's operation. To deal with this problem, we will use range transformation:

$$f(t) = c + \frac{d-c}{b-a}(t-a)$$

It map the interval [a,b] onto the interval [c,d] ([-1,1] in our case). In ideal case, the relationship between transformed x, y values can be seen on the next figure (your plot is not so regular and may be phase shifted).



```
while True:

        magnet = sense.get_compass_raw()

        x = magnet['x']

        y = magnet['y']
```

```
#range transform
xz = -1 + ((1-(-1)) / (xmax - xmin)) * (x - xmin)
yz = -1 + ((1-(-1)) / (ymax - ymin)) * (y - ymin)

#degree (a) calculation
if xz == 0 and yz < 0:
        deg = 90
elif xz == 0 and yz > 0:
        deg = 270
elif yz < 0:
        deg = 360 + math.atan2(yz, xz) * (180/3.14159)
else:
        deg = math.atan2(yz, xz) * (180/3.14159)

#cardinal points
if deg < 45 or deg > 315:
        sense.show_letter('N')
elif deg < 135:
        sense.show_letter('E')
elif deg < 225:
        sense.show_letter('S')
else:
        sense.show_letter('W')
time.sleep(0.2)
```

```
sense.clear()
```

13.7. Try out the application. Do not forget the calibration process!

14. 8-piont compass (task to be solved alone)

Extend the above 4-point compass to an 8-point compass. It depicts both cardinal (**N, S, E, W**) and intermediate (**NE, SE, SW, NW**) points (use the earlier figure as reference).

References

[1] Raspberry Pi projects, https://projects.raspberrypi.org/en. Accessed on 10/01/2020.

This work was supported by the construction EFOP-3.4.3-16-2016-00021. The project was supported by the European Union, co-financed by the European Social Fund.