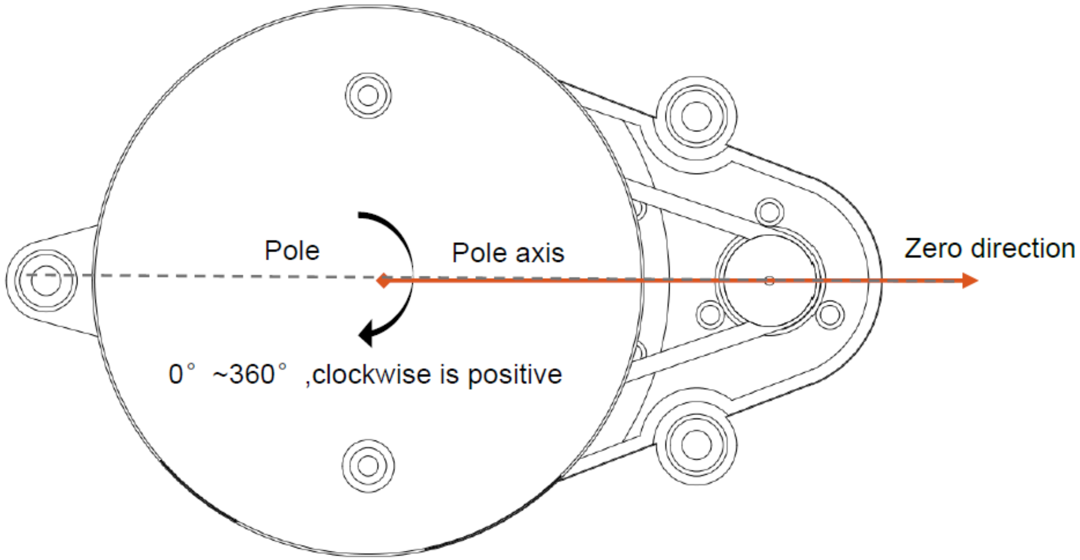
An interferometer: the Lidar

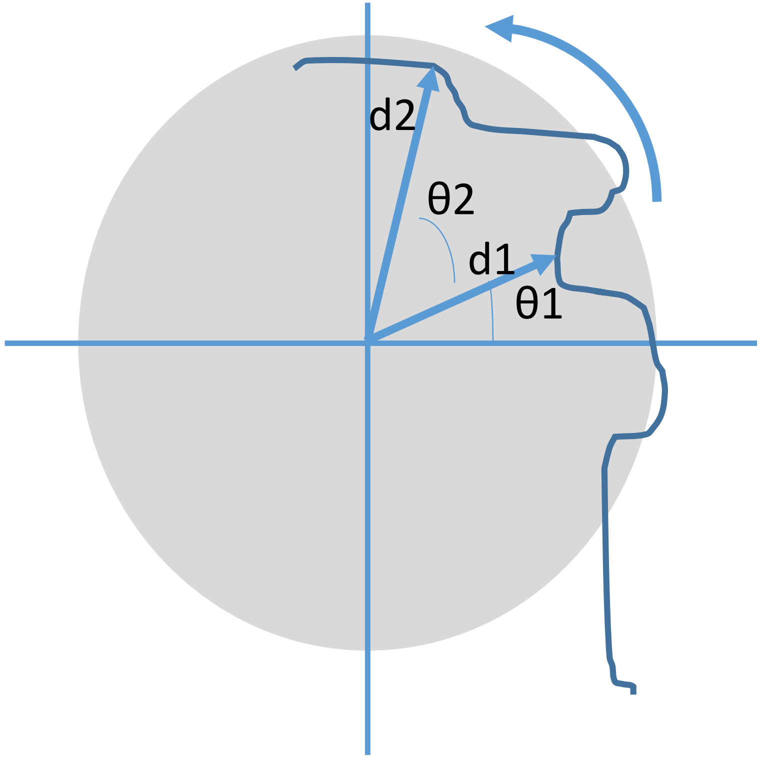
Lidar



Lidar is a method for determining ranges (variable distance) by targeting an object with a laser and measuring the time for the reflected light to return to the receiver (Wikipedia).
So the Lidar can map his surrounding.

Some features

Item	Min	Typical	Max	Unit	Remarks
Ranging frequency	/	5000	/	Hz	Ranging 5000 times per second
Motor frequency	6	/	12	Hz	PWM or voltage speed regulation
Ranging distance	0.12	/	>10	m	Indoor environment, 80% Reflectivity
Fileld of view	/	0-360	/	°	/
Systematic Error	/	2	/	cm	Range≤1m
Statistical Error	/	3.5%	/	/	1m<Range ≤6m
Angle resolution	0.43 (frequency @6Hz)	0.50 (frequency@ 7Hz)	0.86 (frequency@ 12Hz)	°	Different motor frequency



The YDLidar3 Library

Installing the library: “py -m pip install PyLidar3” or “python3 -m pip install PyLidar3” it depends on the symbolic link

- **Connect:** initialises serial connection with Lidar by opening serial port. Result “success status” =True/False.
- **StartScanning:** begins the lidar and returns a generator which returns a dictionary consisting **angle(degrees) and distance(millimeters)**. Result Format : {angle(0): distance, angle(2): distance, ...,angle(359):distance}
- **StopScanning:** stops scanning but keeps serial connection alive.
- **GetHealthStatus:** result = True if Health of lidar is good else returns False
- **GetDeviceInfo**: returns information of Lidar version, serial number…
- **Reset:** restarts the lidar.
- **Disconnect:** stops scanning and close serial communication.

The Lidar scanning is working

```
import PyLidar3
import matplotlib.pyplot as plt
import math
import time
```

Global variables and initialisation

```
#port =  "Enter port name
# which lidar is connected:"
# for instance /dev/ttyUSB0
Obj = PyLidar3.YdLidarX4(port)
```

```
x=[0]*360
y=[0]*360
```

The scanning lasts for 30 secs

```
if(Obj.Connect()):
    print(Obj.GetDeviceInfo())
    gen = Obj.StartScanning()
    plt.figure(1)

    t = time.time() # start time
    data = next(gen)
    delta = time.time() - t
    while (delta) < 30: #scan for 30 seconds
        delta = time.time() - t
        print(delta)
        data = next(gen)
        for angle in range(0,360):
            if(data[angle]>300):
                x[angle] = data[angle] * math.cos(math.radians(angle))
                y[angle] = data[angle] * math.sin(math.radians(angle))
        plt.cla()
        plt.ylim(-2000,2000)
        plt.xlim(-2000,2000)
        plt.scatter(x,y,c='r',s=8)

        plt.pause(0.05)

    plt.close("all")
    Obj.StopScanning()
    Obj.Disconnect()
else:
    print("Error connecting to device")
```

Creating a panel to plot the points

The lidar is working and returns a Dictionary format data

Create an iterator to get the items one by one

Fixing the range

Displaying the points

Waiting for 0.05 s

The results organised as a dictionary

Polar coordinates: The key of the dictionary standing for the angle, precision =1°

The values of the dictionary standing for the distance in cm

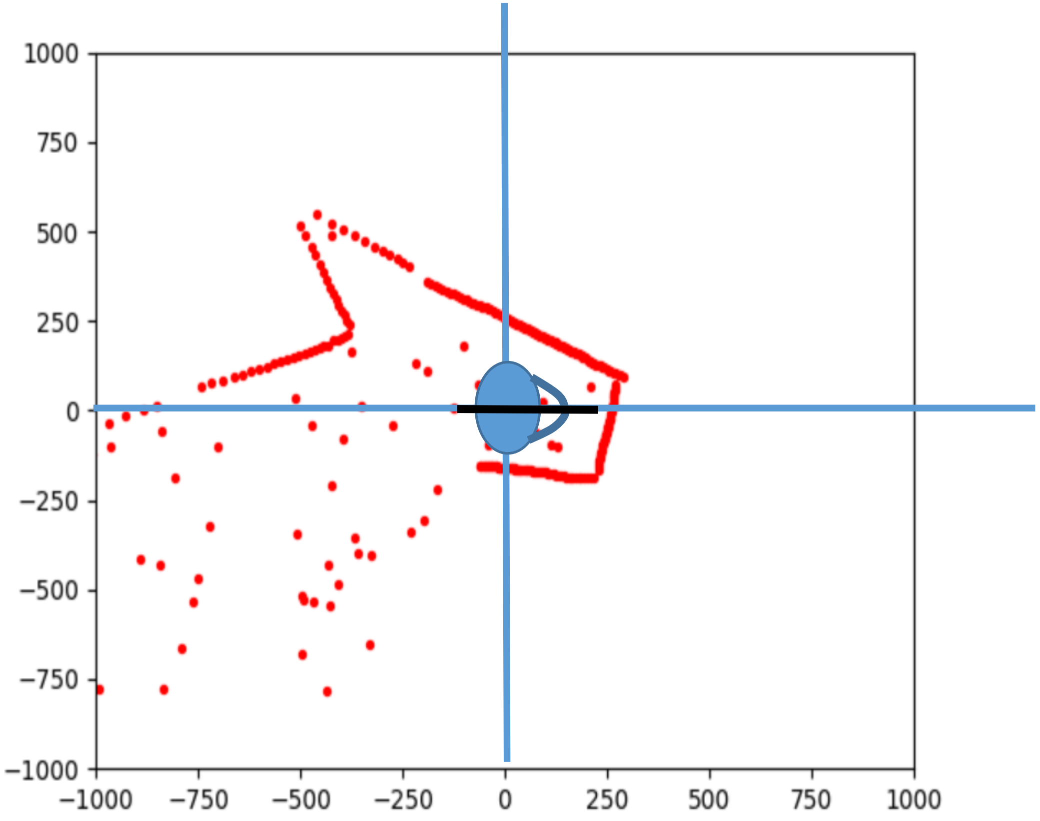
```
{0: 312, 1: 312, 2: 313,
334, 17: 336, 18: 339, 19
32: 392, 33: 398, 34: 403
342, 48: 336, 49: 330, 50
278, 64: 276, 65: 273, 66
 79: 249, 80: 248, 81: 24
```

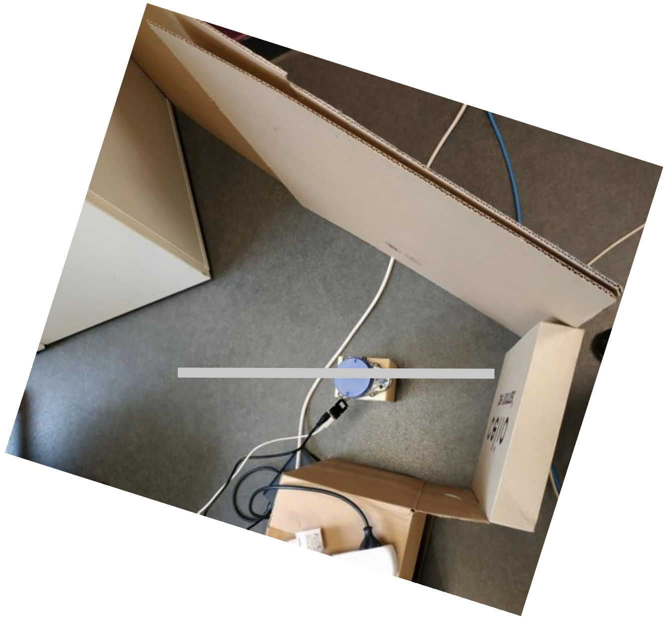
```
x[angle] = data[angle] * math.cos(math.radians(angle))
y[angle] = data[angle] * math.sin(math.radians(angle))
```

```
62, 151: 358, 152: 355, 1
325, 165: 324, 166: 322,
 312, 179: 312, 180: 312,
: 318, 193: 319, 194: 320
6: 346, 207: 349, 208: 35
```

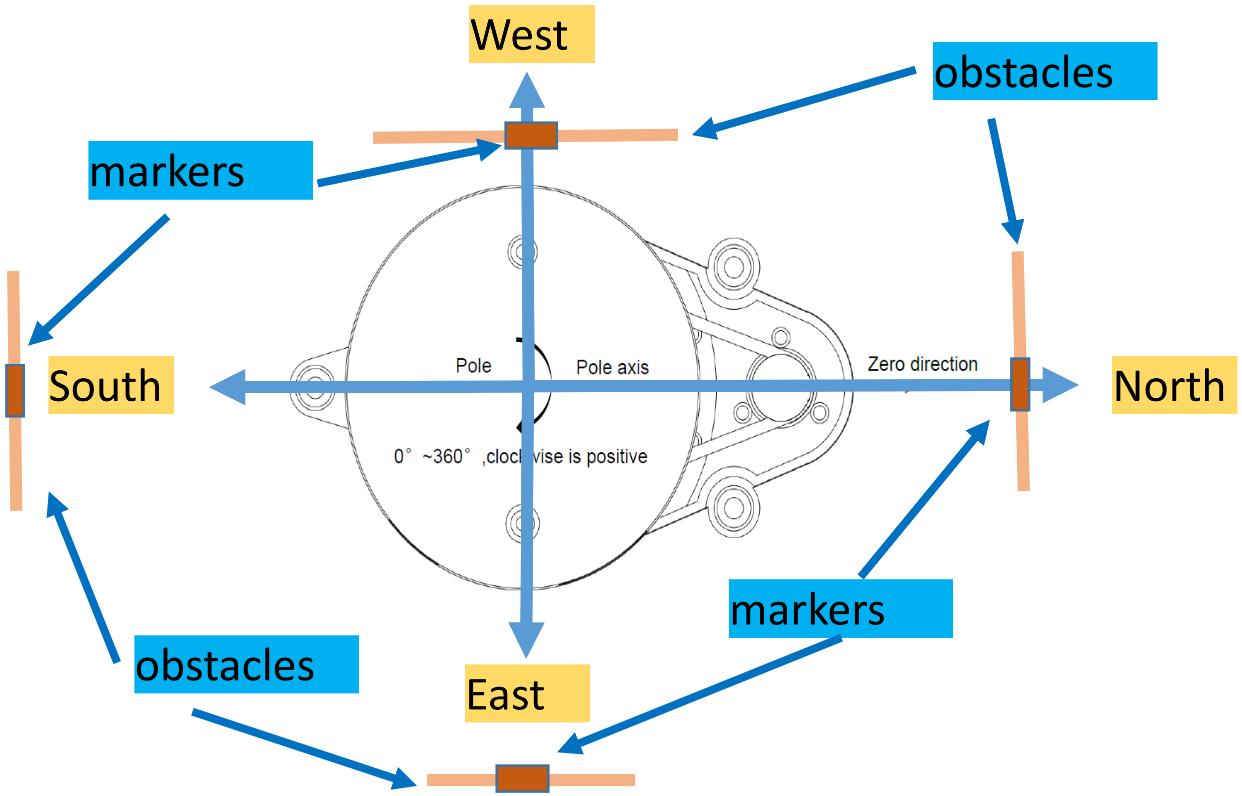
The Cartesian coordinates x,y

the conditions of the experiment and the result





Detecting obstacles according to the cardinal points



Programme for each revolution

```
north = []
…
lineOne = [1]*20
lineMinus10Plus10 = list(range(-10,10))
yNorth = lineMinus10Plus10
ySouth = lineMinus10Plus10
….
data = next(gen) # a dictionary
data = list(data.values()) #now a list with the distance for each angle from 0°
for angle in range(0,360):
        x[angle] = data[angle] * math.cos(math.radians(angle))
        y[angle] = data[angle] * math.sin(math.radians(angle))

north = data[350:359]+ data[0:10] # collecting distance between -10° et 10°
east = data[80:100]) # collecting distance between -80° et 100°
south = = data[170:190]) # collecting distance between -170° et 190°
West = data[260:280]) # collecting distance between -260° et 280°
```

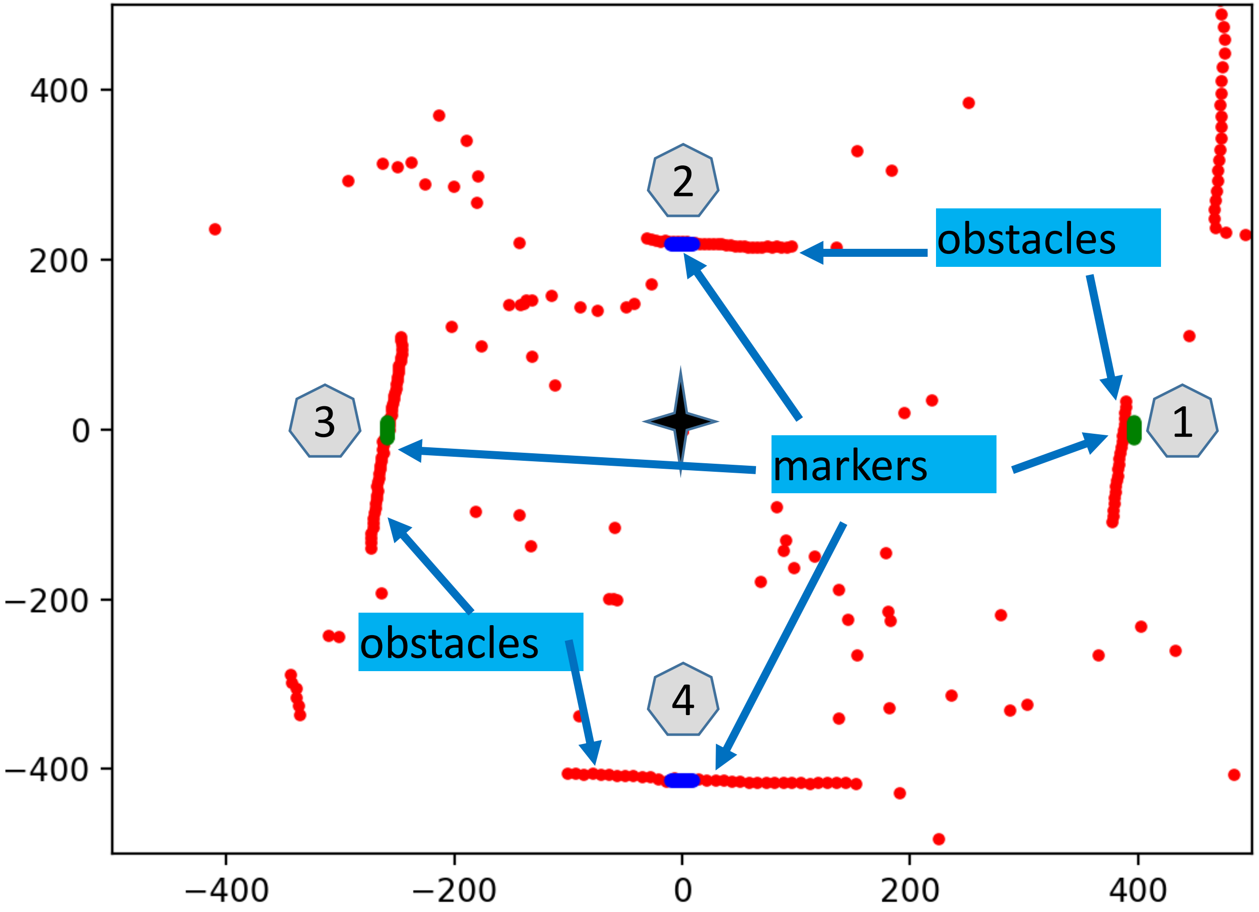
Generation of coordinates for displaying **markers**

“north” is an list, likewise with east, south and west; it’s intended for implementing the **markers**

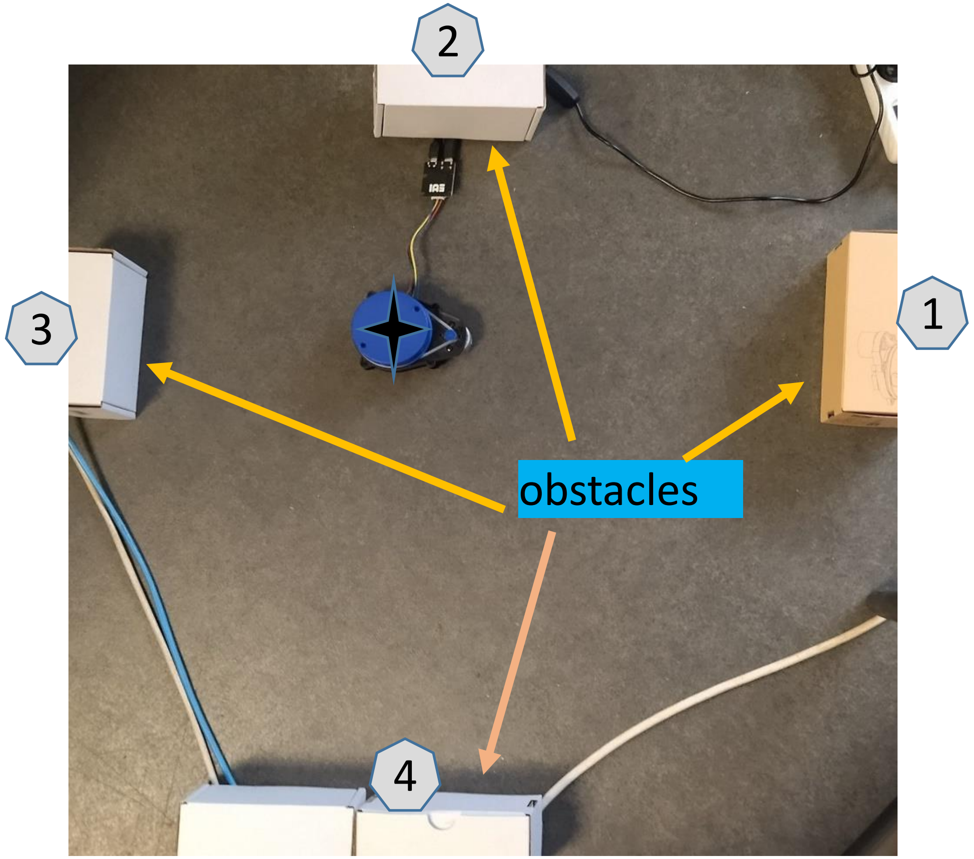
The next part of the programme

```
plt.cla()
plt.ylim(-500,500)
plt.xlim(-500,500)
plt.scatter(x,y,c='r',s=8)
if len(north)!= 0:
    distanceNorth = statistics.mean(north)
    print("north = ", distanceNorth, " ",len(north))
    xNorth= list(map(lambda item: item * distanceNorth, lineOne))
    plt.scatter(xNorth,yNorth,c='g',s=12)
if len(east)!= 0:
    distanceEast = statistics.mean(east)
    print("east = ",distanceEast ," ",len(east))
    yEast = list(map(lambda item: item * distanceEast, lineOne))
    plt.scatter(xEast,yEast,c='b',s=12)
         …
 #end of the loop “while”
```

Results



400
200
0
−200
−400
−400
−200
0
200
400
2
3
1
4
obstacles
markers
obstacles

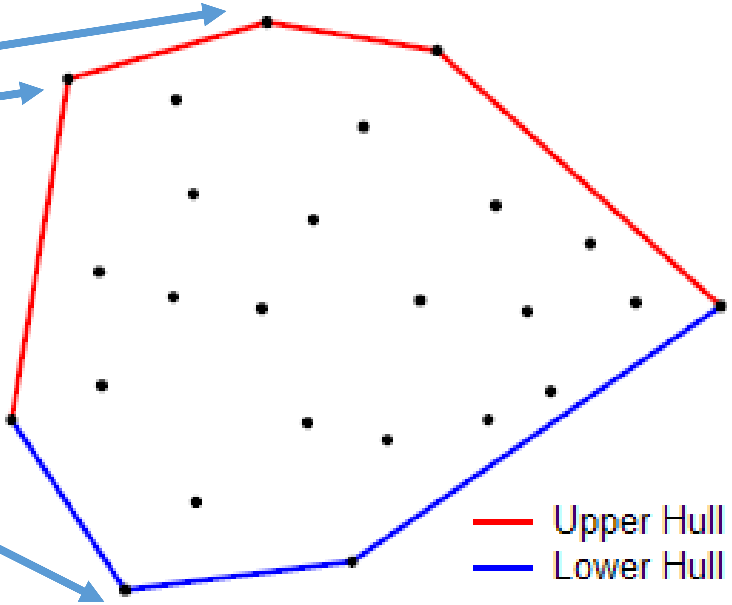


2
3
1
4
obstacles

Determining the polygon surrounding a cloud of points: Hull Algorithm

- Given a cloud of points in the plane. The convex hull of the set is the smallest convex polygon that contains all the points of it.
- https://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain#Python

The most outer points

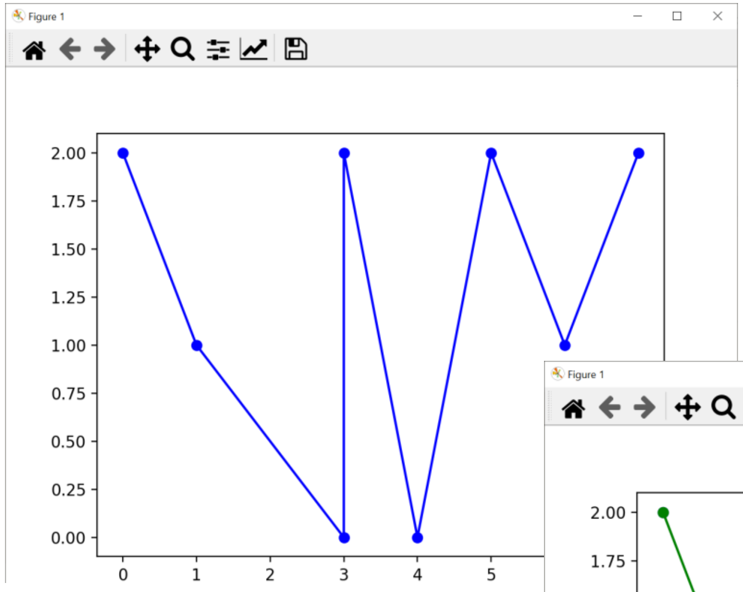


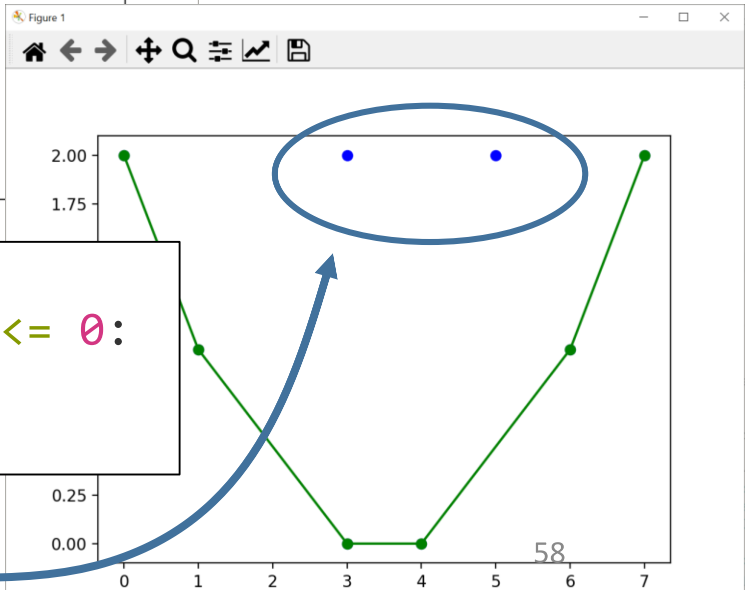
Application of Hull Algorithm

[[0, 2], [1, 1], [3, 2], [3, 0], [4, 0], [5, 2], [6, 1], [7, 2]]

sorted(...)

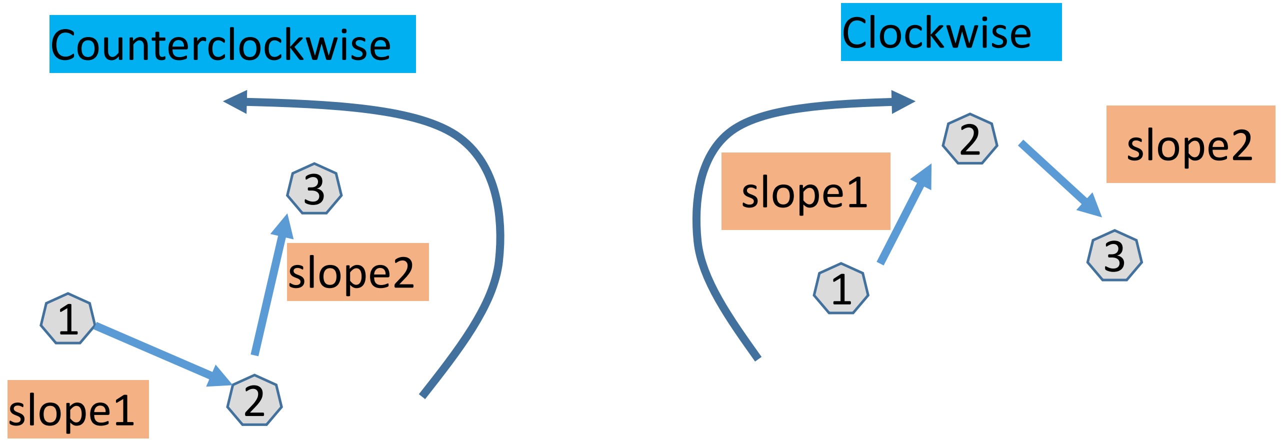
[[0, 2], [1, 1], [3, 0], [3, 2], [4, 0], [5, 2], [6, 1], [7, 2]]





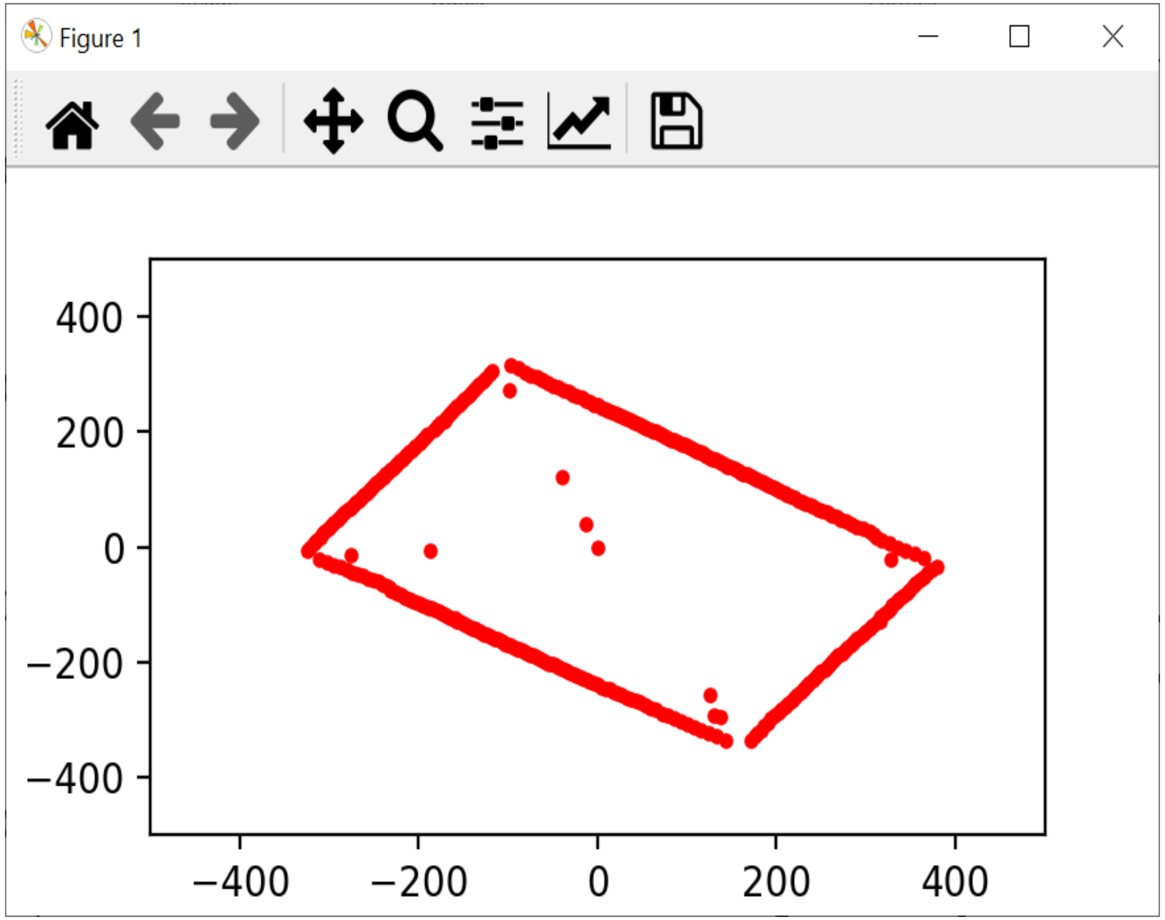

```
for p in points:
    while len(lower) >= 2 and orientation(lower[-2], lower[-1], p) <= 0:
        lower.pop()
    lower.append(p)
```

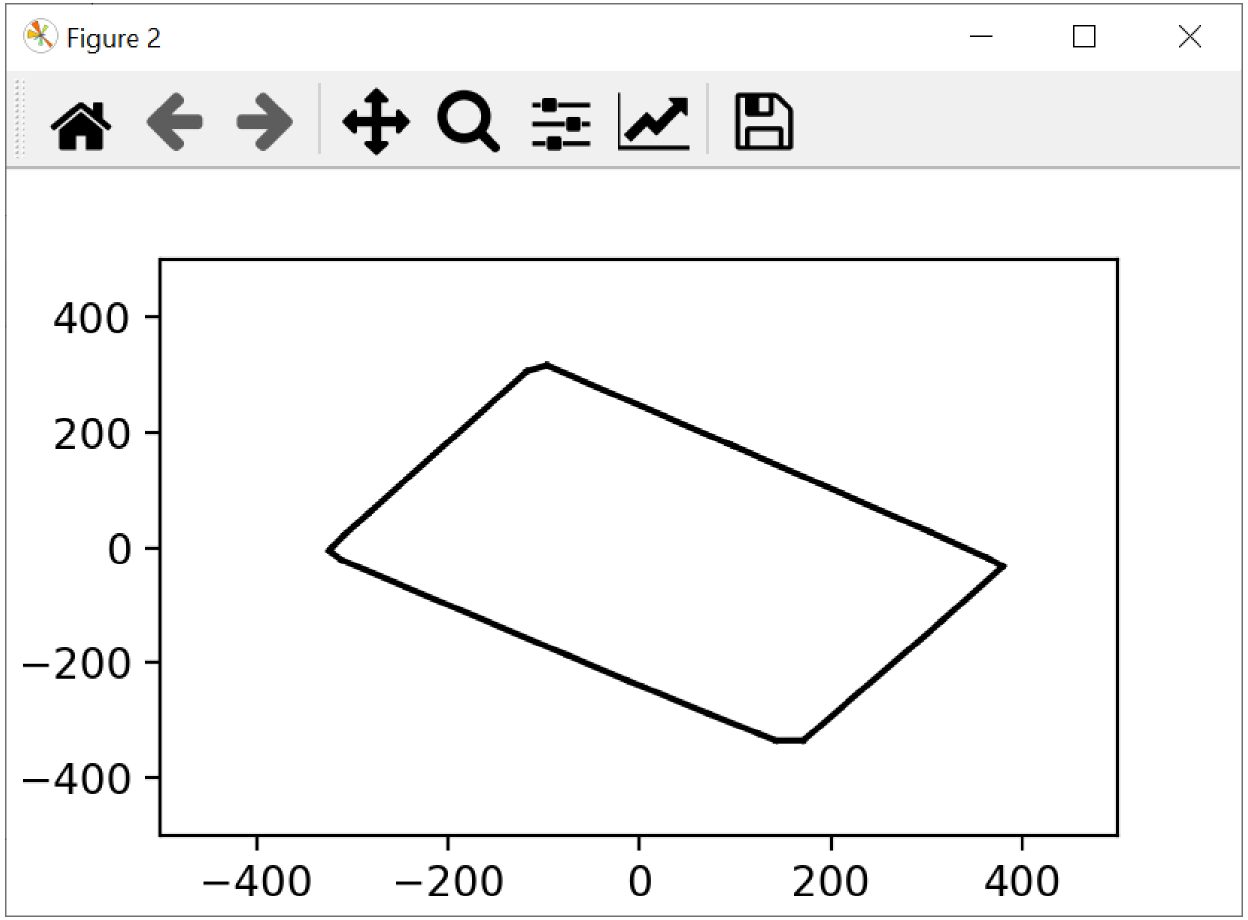
Determining the orientation of 3 ordered points



- Slope of line segment (p1, p2): slope1 = (y2 - y1)/(x2 - x1)
- Slope of line segment (p2, p3): slope2 = (y3 - y2)/(x3 - x2)
- If slope1 < slope2, the orientation is counterclockwise (left turn)

Example: suppressing the inner points





Using Hull function included in Scipy Library

[[335.0, 0.0], [326.95019631613997, 5.706936904991708], [317.80628299207245, 11.098039951395307], [311.57241484342705, 16.328818347798478], [308.2472915302857, 21.554750386934717], [303.8393829179824, 26.58250153803574], [296.3675248197454, 31.149482053760735], [289.823476279266, 35.58584827430306]…]

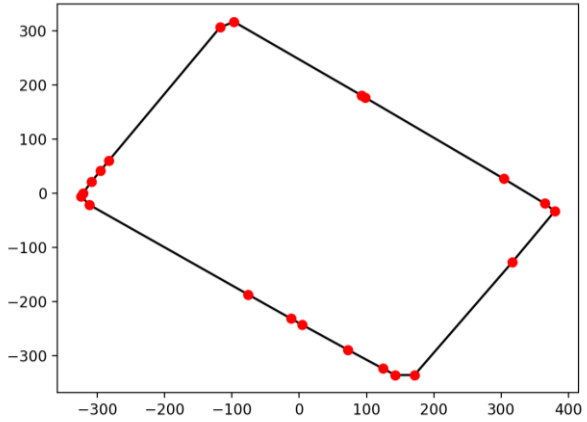
```
from scipy.spatial import ConvexHull, convex_hull_plot_2d
# to fill
hull = ConvexHull(pointSet)
for simplex in hull.simplices:
    plt.plot(pointSet[simplex, 0], pointSet[simplex, 1], 'k-')
plt.plot(pointSet[hull.vertices,0], pointSet[hull.vertices,1], 'ro', lw=-1)
plt.show()
```

Importing the library scipy

Calculation of the most outer points

Black lines

Red Dots



Using threads: how to display the objects detected by the Lidar

Global variable

```
#port =  "Enter port name
# which Lidar is connected:"
# for instance /dev/ttyUSB0
Obj = PyLidar3.YdLidarX4(port)
```

```
port  = "        "
Obj = PyLidar3.YdLidarX4(port)
threading.Thread(target=ScanLidar).start()
plt.figure(1)
while is_plotting:
    plt.cla()
    plt.ylim(-1000,1000)
    plt.xlim(-1000,1000)
    plt.scatter(x,y,c='r',s=8)
    plt.pause(0.001)
plt.close("all")
```

The programme is organised as follows:

- a thread intended to deal with the Lidar, namely scanning the surrounding.
- the main part intended to display points machting the obstacles.

Launching the thread intended to deal with the Lidar

Using threads: the thread “ScanLidar”

```python
import threading
import PyLidar3
import matplotlib.pyplot as plt
import math
import time
```

Global variables and initialisation

```python
is_plotting = True
x=[]
y=[]
for _ in range(360):
    x.append(0)
    y.append(0)
```

```python
def ScanLidar():
    global Obj
    global is_plotting
    if(Obj.Connect()):
        print(Obj.GetDeviceInfo())
        gen = Obj.StartScanning()
        t = time.time() # start time
        while (time.time() - t) < 60: #scan for 1 mns
            data = next(gen)
            for angle in range(0,360):
                if(data[angle] < 1400 and data[angle] > 50):
                    x[angle] = data[angle] * math.cos(math.radians(angle))
                    y[angle] = data[angle] * math.sin(math.radians(angle))
                else:
                    x[angle] = 0
                    y[angle] = 0
        is_plotting = False
        Obj.StopScanning()
        Obj.Disconnect()
    else:
        print("Error connecting to device")
        is_plotting = False
```

The lidar is working and returning a Dictionary format data

The scanning lasts for 60 secs

Create an iterator to get the items one by one

Measuring the distance at each degree (the resolution)

Stops the scanning and closes the connection