**Flight plan**

**Step #1: Create a polygona on a map and get the coordinate.**

Done:

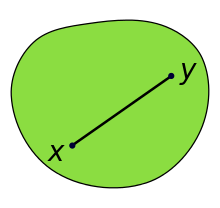
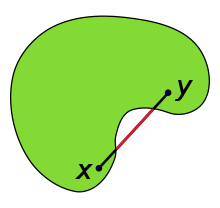
* Create a new page on Vue to do flight plans.
* Could add point.

#TODO: Prevent points from being sent if less than 3

**Step #2: Send the GPS coordinates to the server with the python code.**

#TODO: No idea yet

**Step #3: On python, with the coordinates and the distance between to line create a path.**

**Divide the problem in two:**

* Convex figures (convex in the mathematical sense, [Convex set - Wikipedia](https://en.wikipedia.org/wiki/Convex_set))
* Non-convex figures

**Convex forms:**

Hypothesis: The 2 minimums distance between 3 points are the edges of our surface. (#TODO: Verify in theory if it’s correct)

So now, we will take the ***longest side*** and its ***perpendicular***. And create a grid pattern of the aera perpendicular to our grid pattern.

#QUESTION: longest side or longest distance between 2 points? Which one will make the flight plan the smaller possible?

We’ll choose the **distance** between 2 lines in the program in meters.

To calculate the distance between two gps points we’ll use the client geopi ([geopy · PyPI](https://pypi.org/project/geopy/)) with the function distance.distance(pt1, pt2).

The points of our flight plan will be the crossing of each line or the centre of each cell.

**How to find the point on the edge of the polygon?**

On a orthogonal marker, we can use the equations of lines (y = mx + b) and we can find it easily.

In our case, it’s not possible. Because the “longest distance” won’t necessary be on the x axe.

**Pt1**

**Pt2**

**Longest distance**

**Perpendicular to the longest distance**

**(x, y)**

**Idea:** Try with the scalar product.

* Make 2 vectors with the norm of the distance between photos (could be different between length and width)
* Or with the part of vectors for example (OPt1 + OPt2) / 2, if x =

**Whished method:**

**Pt1**

**Pt2**

**Longest distance**

**Pt**

**X\_pt R1**

**X\_pt R0**

**R1 (u, v)**

**R0 (x, y)**

From a point on the longest distance find the abscissa corresponding and with the function already created find the point on the segment Pt1, Pt2

**How to?**

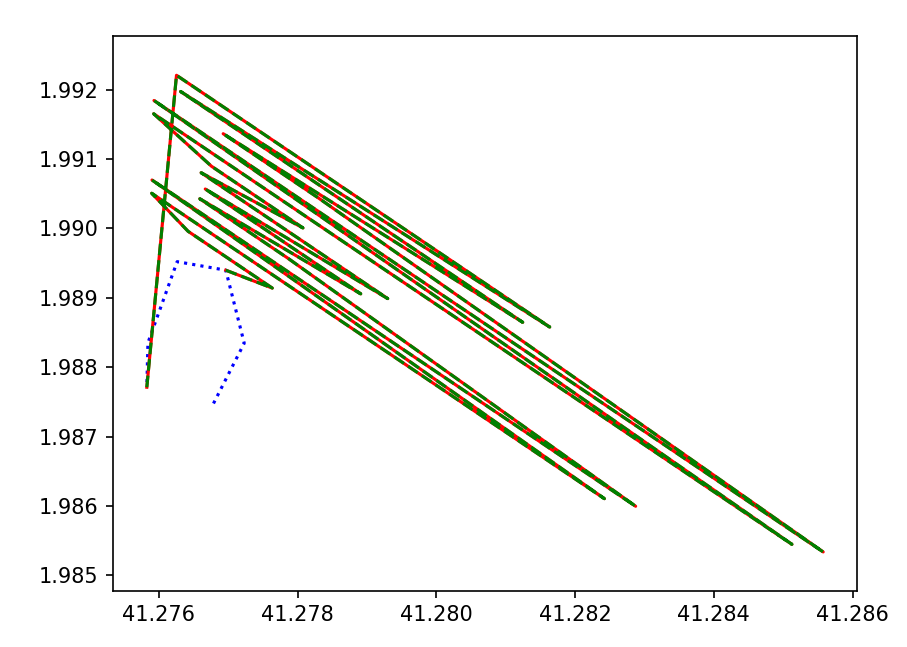
Find the relation between the x on R1 and x on R0. We search the function giving f(x\_R1) = x\_R0.

For x\_R1 => scalar product

For x\_R0 => already have it by coordinate

And with this, we can access to the point researched and create our flight plan.

Bug to resolved:



[(41.275827, 1.987712), (41.275843, 1.988352), (41.276264, 1.989522), (41.276965, 1.989399), (41.277231, 1.988347), (41.276788, 1.987478)]

[(41.276788, 1.987478), (41.277231, 1.988347), (41.276965, 1.989399), (41.276264, 1.989522), (41.275843, 1.988352), (41.275827, 1.987712)]

I found 2 big problems in my application:

***Minimum distance***

***2 Neighbors***

* I had 3 times the same neighbours for different points. It’s impossible.
* I can’t find the 2 neighbours with the minimum distances.

Possible solution : Try with angles

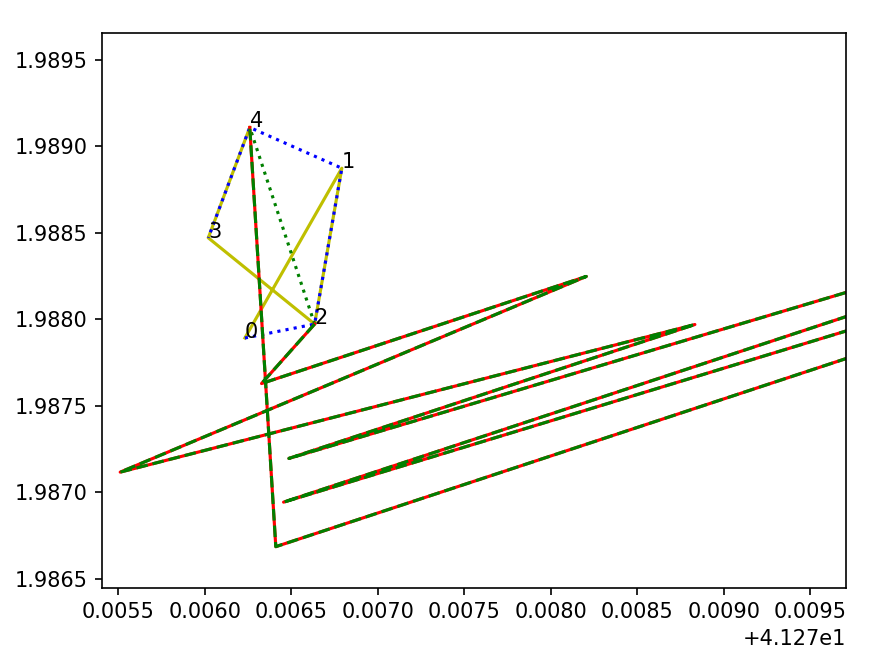
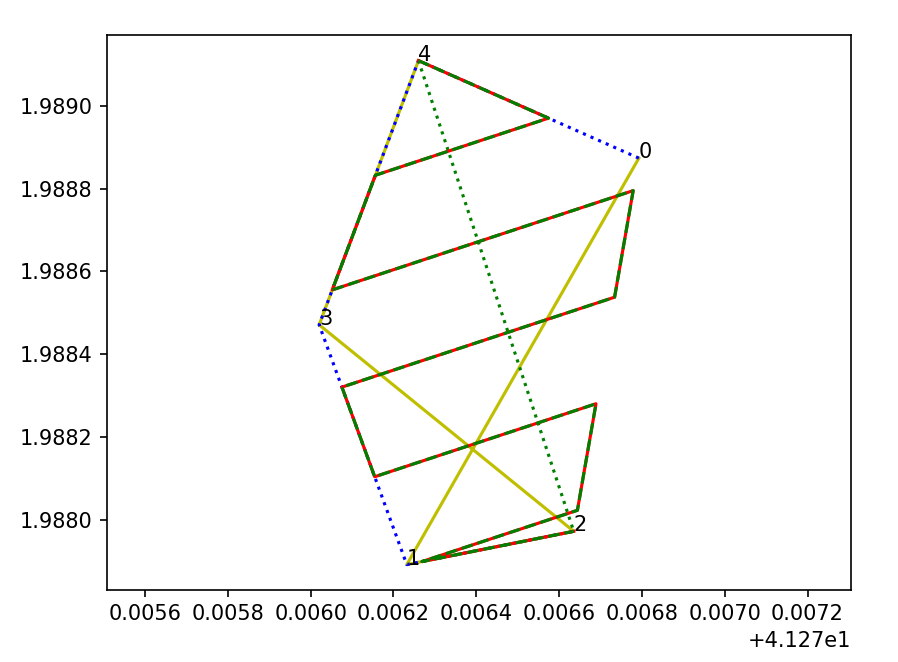
The sorting:

list\_pts = [0 | 1 | 2 | … | … ]

sorted = [0 | pts(neighbours(0)) | pts(neighbours(neighbours(0)) | … | … ]

New bug:

With the 2 same list just by inversing the points 0 and 1.



Problem of vector, the second point always went down even if the area was up.

Some sources:

* Study a bit of topology (mathematic) [Convex set - Wikipedia](https://en.wikipedia.org/wiki/Convex_set) // [Connexité (mathématiques) — Wikipédia (wikipedia.org)](https://fr.wikipedia.org/wiki/Connexit%C3%A9_(math%C3%A9matiques))
* Scalar product and orthogonal projection
* Python documentation

function onMapClick(e){

            count\_points = count\_points + 1;

            console.log(e.latlng)

            if (count\_points > 1) {

                let last = points.value[points.value.length-1];

            }

            // points.value.push(e.latlng);

            // console.log('points', points);

            // console.log('points val', points.value);

            // console.log('points lght', points.value.length);

            // if (points.value.length > 1){

            //     leaflet.polyline(points.value, {color: 'red'}).addTo(map);

            // }

            let wp = leaflet.marker(e.latlng, {draggable:'true'}).addTo(map).bindTooltip(count\_points.toString(),  {

                            permanent: true,

                            direction: 'center',

                            className: "my-labels"});

            // wp.on('dragend', onMarkerDragEnd);

            // wp.on('dragend', function(event) {

            //     var marker = event.target;

            //     var position = marker.getLatLng();

            //     console.log('moving to ', position);

            //     let index = points.value.findIndex(point => point.equals(marker.getLatLng()));

            //     if (index > -1) {

            //         points.value[index] = position;

            //     }

            //     redrawPolyline();

            //     marker.setLatLng(position, {draggable: true})

            //     // marker.setLatLng(new L.LatLng(position.lat, position.lng), {draggable: true})

            //     // map.panTo(new L.LatLng(position.lat, position.lng))

            // });

            points.value.push(e.latlng);

            if (points.value.length > 1) {

                redrawPolyline();

            }

            // let wp = leaflet.marker(e.latlng, {draggable:'true'}).addTo(map).bindTooltip(count.toString(),  {

            //                 permanent: true,

            //                 direction: 'center',

            //                 className: "my-labels"

            //              });

        }