Ground Control Station for UAV Quadcopter

Project By:

Ran Glait
Kfir Nir-Zvi

Supervised By:

Majd Srour

Introduction

Autonomous Quadcopters are used to different purposes where they are required to navigate from a starting point to a destination point in its area. Thus, a GCS (Ground Control Station) should have information about local and target positions of the UAV.
The GCS must perform the following tasks:

1. Defining the mission for the UAV – i.e. sending the flying track to the UAV (waypoints).
2. Monitoring the UAV's status – getting the present position, data of heading, speed etc.
3. Communicating with the UAV – Using ZigBee communication protocol, in order to enable bidirectional communication between the GCS and the UAV.
4. Provide a GUI – for a comfortable, simple and efficient use of the GCS.

Project's Goal

Build a GCS in order to control and track an UAV, specifically, a home-built quadcopter.

The GCS should send the mission (a set of waypoints) to the quadcopter, and a start mission command. From that point, the GCS should keep a connection with the quadcopter in order to track and monitor it's data and movement. In case of connection lost – the GCS must show an ERROR message, while the quadcopter lands by itself immediately.

While the UAV is in the air, the GCS should present its position on a map. On the same map, it should also present the waypoints (as bald points), and the complete track (as a line).

The current GCS is a basic platform that can be extended, depending on the requirements. This is done by simple operations of adding the needed functions to the platform.

Writing an autonomous algorithm to navigate the UAV, with obstacles avoiding ability.

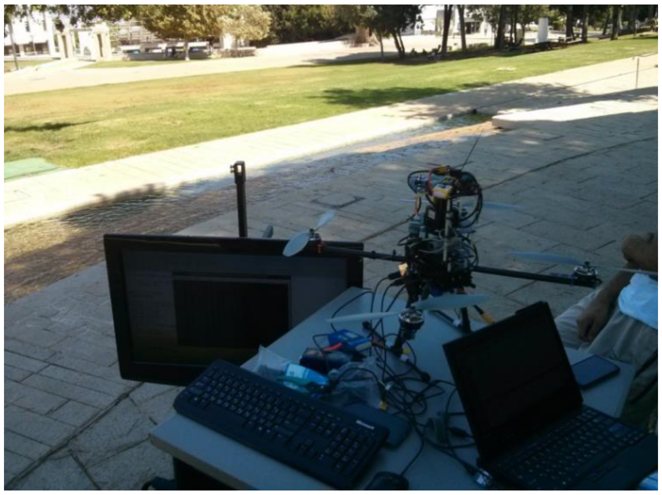


Figure 1 - Working Process

GUI

The GUI was developed in Qt Creator version 3.0.1, on C++.

The main screen of the GUI is based on a map that shows the current point of the UAV, its heading, its path, a list of waypoints and several available commands:

1. START MISSION – start the navigation according to the waypoints.
2. Clear All Waypoints – empty the waypoints list, delete the mission.
3. Get Waypoint List – receive the waypoint list that is held in the UAV itself. This is done in order to compare to the list we sent to it.
4. EMERGENCY – command UAV to halt the mission and land immediately.
5. Remove Marker – delete a specific waypoint from the list and send a signal to the UAV so that it is removed from its mission.
6. CheckBox – disables or enables (depending on the status) a laser sensor in order to avoid obstacles.

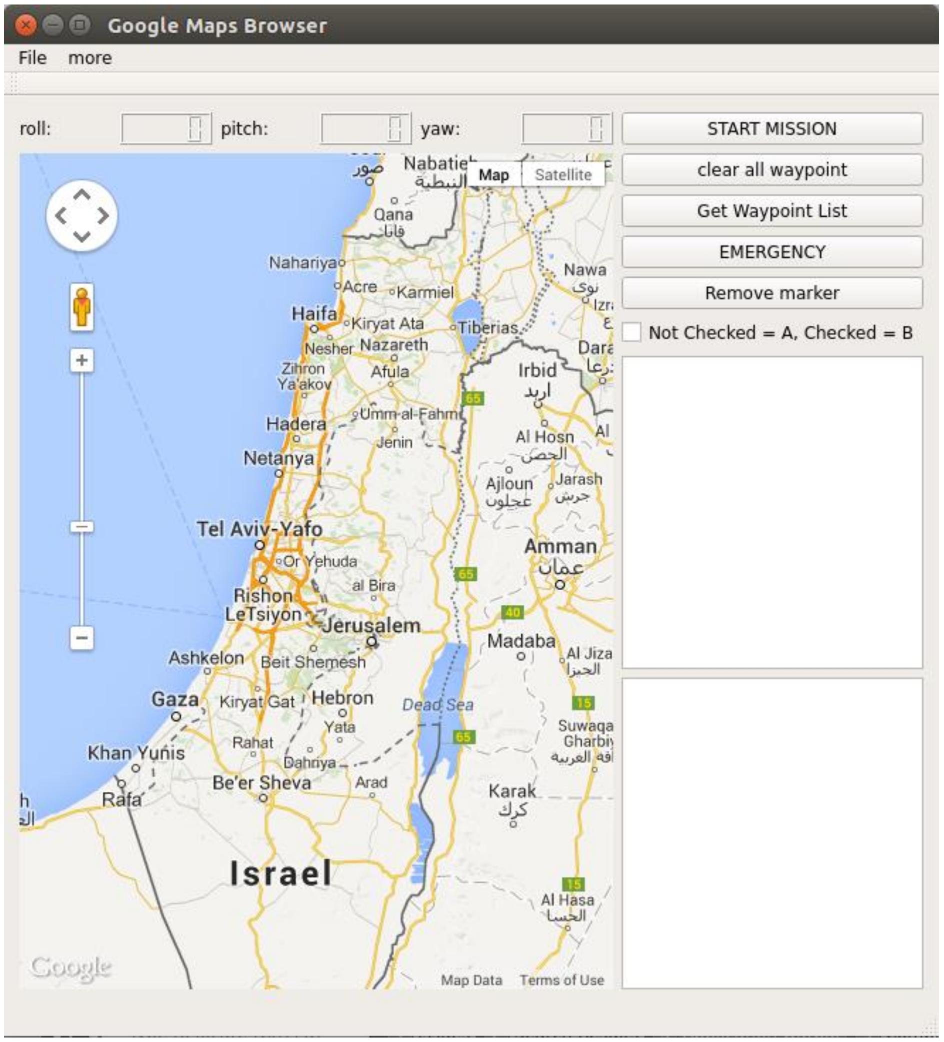
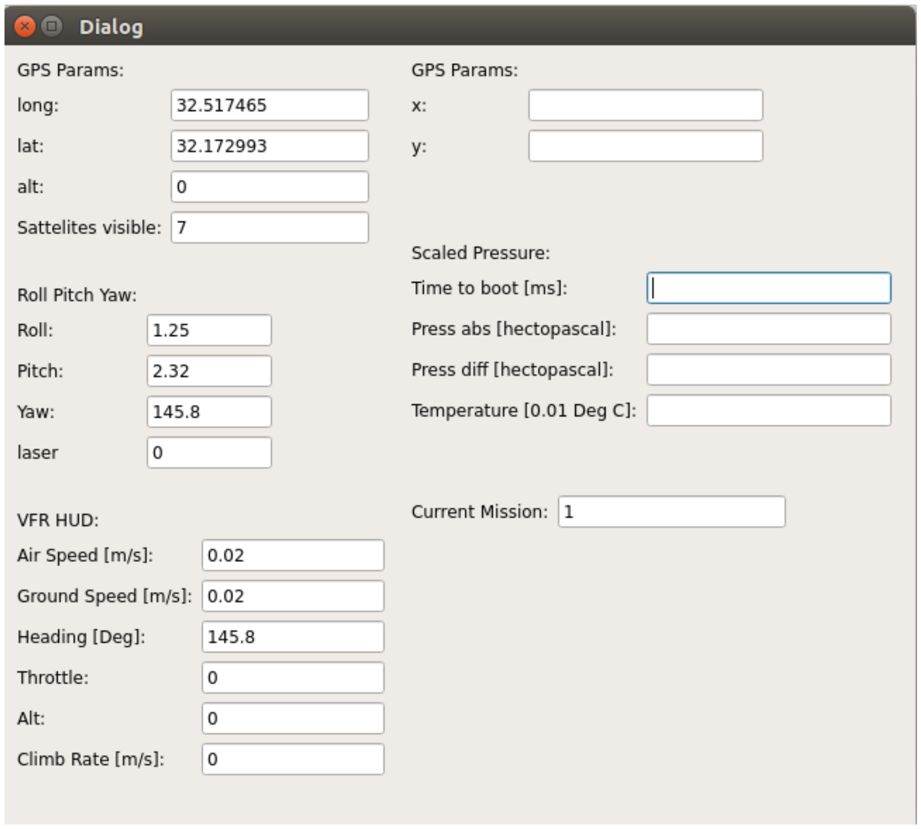


Figure 2 - GUI

Two additional buttons at the top-left part of the screen:

1. File – has the Quit command.
2. More – includes 3 options:
   i. Parameters Display – an auxiliary screen, which includes the data needed for quality conditions of flight.
   For example: number of visible satellites, current GPS data as longitude and latitude, etc.



GPS Params:

long:	32.517465
lat:	32.172993
alt:	0
Sattelites visible:	7

Roll Pitch Yaw:

Roll:	1.25
Pitch:	2.32
Yaw:	145.8
laser	0

VFR HUD:

Air Speed [m/s]:	0.02
Ground Speed [m/s]:	0.02
Heading [Deg]:	145.8
Throttle:	0
Alt:	0
Climb Rate [m/s]:	0

GPS Params:

x:	
y:	

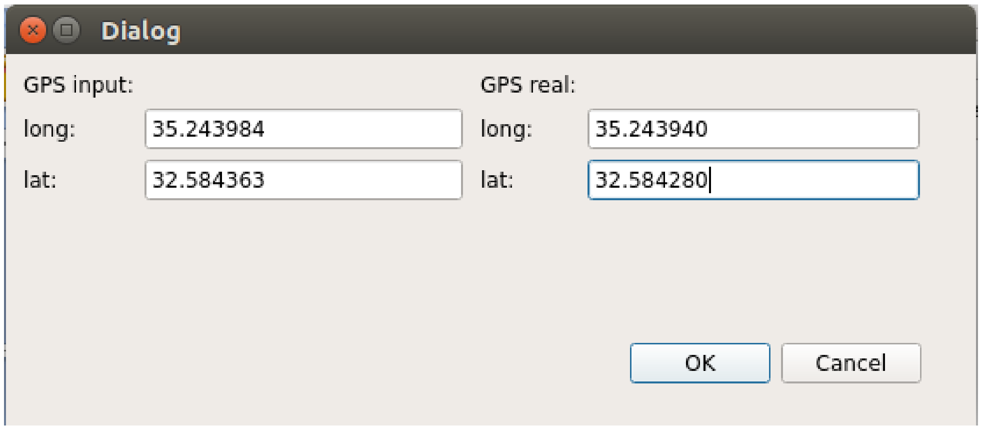
Scaled Pressure:

Time to boot [ms]:	
Press abs [hectopascal]:	
Press diff [hectopascal]:	
Temperature [0.01 Deg C]:	

Current Mission:	1

Figure 3 - Parameters Display

   ii. GPS Calibration – A dialog box that calibrates the GPS by feeding a known local accurate point.

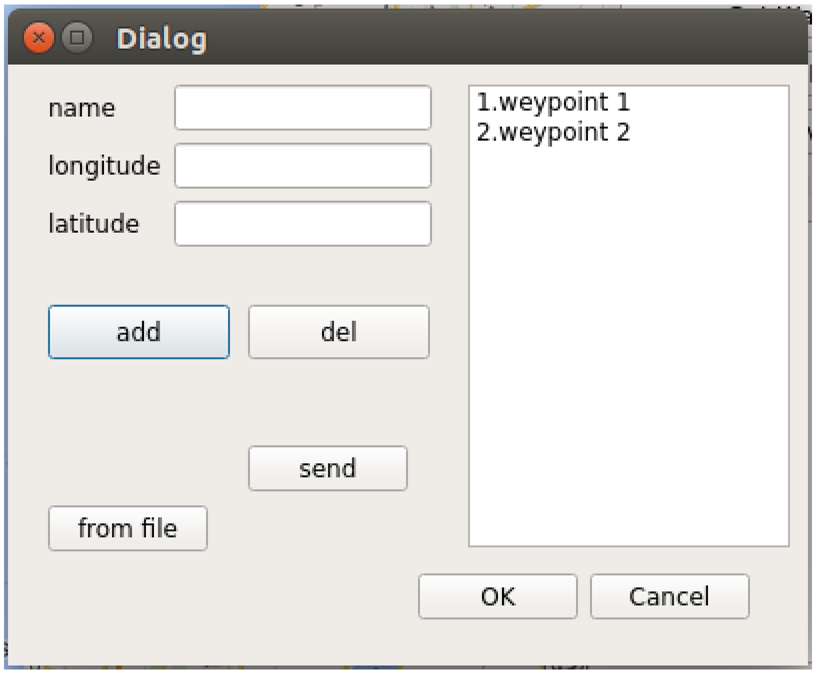


GPS input:
long: 35.243984
lat: 32.584363

GPS real:
long: 35.243940
lat: 32.584280

Figure 4 - GPS Calibration

iii. Add Markers – a dialog box, in which we feed our mission waypoints. It can be done manually for each point, or from a file that contains a list in a certain format.



1.weypoint 1
2.weypoint 2

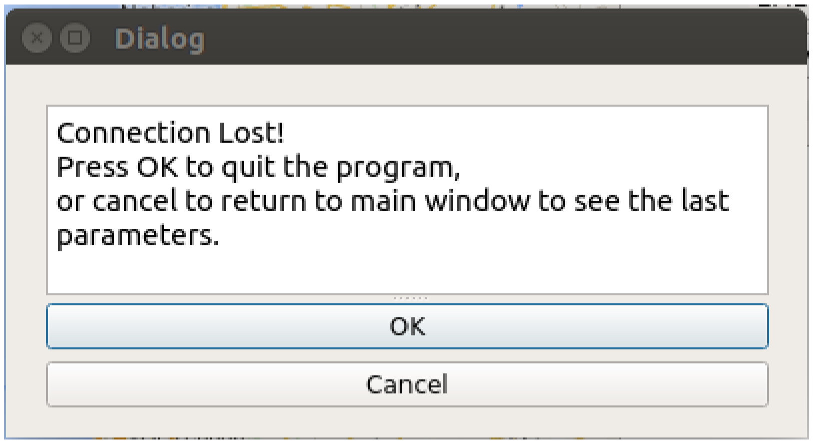
Figure 5 - Add Markers Dialog Box

Implementation

The software is divided into several parts which communicate via signals. Those parts run on different threads paralleled.

The different parts of the software:

1. GUI thread – the main thread of the program. Loads the GUI, creates the communication thread, and keeps the GUI updated according to the data that is received from the communication thread (from the UAV). When data is received from the user, the GUI thread sends a signal to the communication thread with the relevant data, and when a signal is received from the communication thread (new or updated data from UAV), it handles the signal and updates the relevant fields.
   If for a predefined period of time the GCS does not get any message from the communication thread, it is assumed that the connection has been lost. Given such a case, an error message is presented in the GUI.



Connection Lost!
Press OK to quit the program,
or cancel to return to main window to see the last parameters.

Figure 6 - Connection Lost

2. Communication thread – The main goal of this thread is to take care of the communication between the UAV and the GCS. It uses the ZigBee hardware, with the MavLink communication protocol, which is a unique protocol for UAV's.

On-Board

The quadcopter is composed of several layers, as is described below:

On-Board Algorithm

The Odroid computer is installed on the quadcopter onboard, its goal is to perform the following algorithmic tasks:

1. Navigation between two points: once a navigation track is accepted by the computer, the Odroid computer calculates the route of flight between each two points in the route. The calculations are done separately as follows:

    a. Dividing the route into short sub-routes, each 1 meter long, the calculation for each of them is executed separately.

    b. Every sub-route's coordinates are calculated by inserting the current waypoint and target direction into the known formulas below. Note that the target direction is calculated only once from the original target and destination points, whereas each new sub-route is calculated after the execution of the previous sub-route.

    $$Latitude = \frac{x\cos\alpha}{2\pi R}\cdot 360^{\circ} + Latitude_{old}$$

    $$Longitude = \frac{x\sin\alpha}{2\pi R\cos(Latitude)}\cdot 360^{\circ} + longitude_{old}$$

    $$Altitude = Const\ \ predefined.$$

    Where x is the distance to travel, α is angel from the north and R is radius of the earth

2. In case an obstacle is detected in front of the quadcopter by the laser sensor at range of 3 meters at most. The Odroid recalculates recursively the route in the following way:
    a. Deciding from which direction to bypass the obstacle, according to boundaries that are predefined: bypass from the direction of the farthest edge of the boundary.
    b. Moving 1 meter right or left according to the previous decision.
    c. Repeating sections (a) and (b) until the obstacle is no longer detected.
    d. Returning to section (1) in order to calculate a new route, the current point is the new start point, and the target point remains the original one.

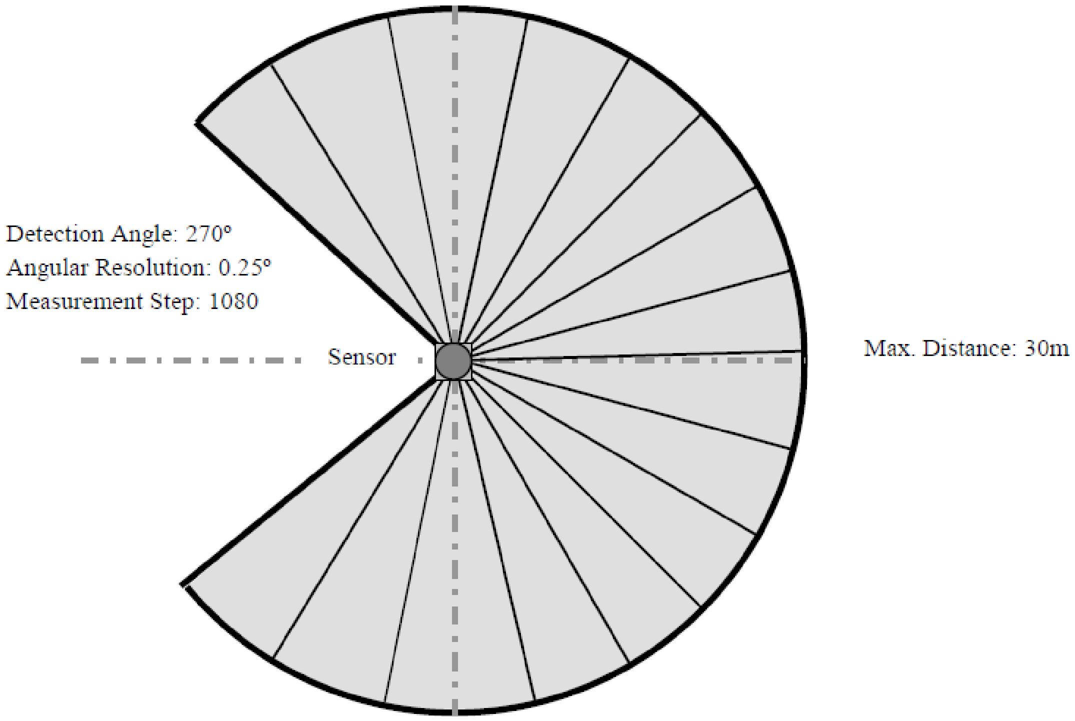
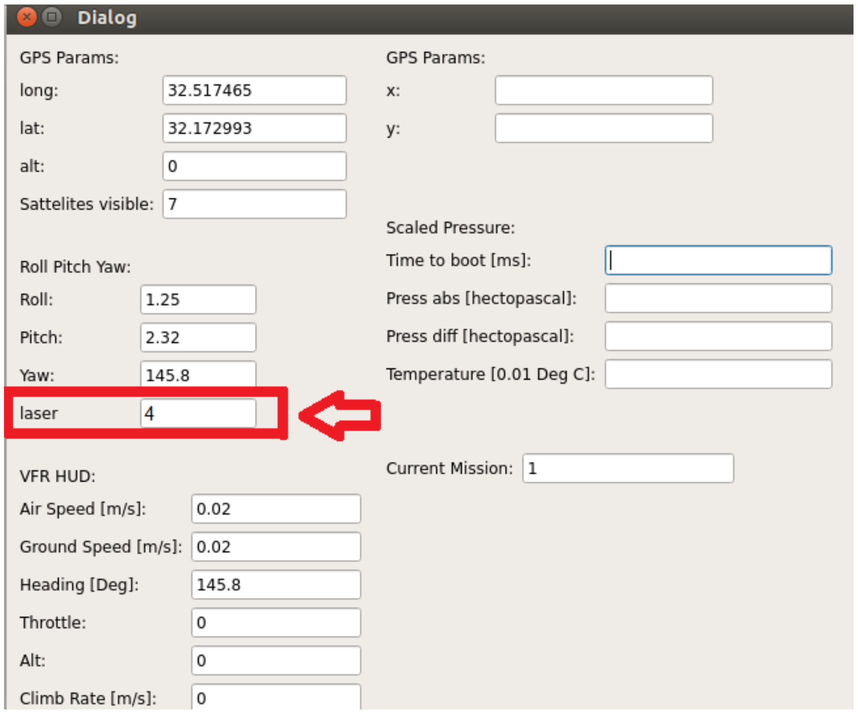


Figure 7 - Laser Measure Diagram


Dialog

GPS Params:

long:	32.517465
lat:	32.172993
alt:	0
Sattelites visible:	7

Roll Pitch Yaw:

Roll:	1.25
Pitch:	2.32
Yaw:	145.8
laser	4

VFR HUD:

Air Speed [m/s]:	0.02
Ground Speed [m/s]:	0.02
Heading [Deg]:	145.8
Throttle:	0
Alt:	0
Climb Rate [m/s]:	0

GPS Params:

x:	
y:	

Scaled Pressure:

Time to boot [ms]:	
Press abs [hectopascal]:	
Press diff [hectopascal]:	
Temperature [0.01 Deg C]:	

Current Mission:	1

Figure 8 - The laser measurment

On-Board Communication

An Arduino computer with built in software is installed on the quadcopter alongside the odroid computer, its goal is to function as the flight controller.
The two computers (the Arduino and the Odroid) are connected via USB cable, using the communication protocol Mavlink.
Mavlink is used mostly for communication between a Ground Control Station (GCS) and unmanned vehicles, and in the inter-communication of the subsystem of the vehicle. It can be used to transmit the orientation of the vehicle, its GPS location and speed.

When switching from manual control to autonomous control, the Odroid computer takes the control over the quadcopter, giving commands to the arduino computer, using data from the on-board algorithm and the GCS commands.
Messages from the Odroid computer to the Arduino computer go through the following steps:

- The Odroid computer generates a serial command.
- The Odroid computer sends the serial command to the Arduino computer.
- The Arduino computer parses the command received into Pulse Width Modulation (PWM), which is an Arduino-based message structure.
- The Arduino computer executes the command.

Testing and Conclusions

In order to test our algorithm we experiment flight cases in a secured area which was done in order to avoid damage to people and property.

i. In order to examine the navigation algorithm without obstacles we took the following steps:
    1. We checked the coordinates of sample point using the Arduino computer. The data was received as previously described, from the Arduino computer, through the Odroid computer, which sends the data to the GCS from which we can read it.
    2. Using the samples we collected in step (1) we assembled a list of waypoints on our GCS.
    3. We commanded the quadcopter to take off manually. Once the quadcopter was hovering, we switched control to autonomous.
    4. We sent the list of waypoints from the GCS to the quadcopter, and pressed the "START MISSION" button.
    5. We evaluated the Quadcopter's flight, comparing it to the waypoints mentioned above.

    Results:
    The quadcopter passed the waypoints, being inaccurate by 5-10 meters. This did not match the expected results, as we had predicted an inaccuracy of up to 30 cm.
    After reexamining the algorithm repeating the experiment, we received accuracy no

better than 1-2 meters. We conclude that there is some GPS inaccuracy, and that the best results were given when the GPS antenna was set away from the different electronic devices. This might be due to electronic or magnetic fields caused by the different devices.

ii. The second part of the experiments was to examine the obstacles avoiding algorithm, this was done as follows:
1. Sections 1-4 were repeated from the first experiment.
2. An obstacle was held high enough in order for it to be in the quadcopter's route.
3. Using the transmission of the laser sensor from the quadcopter we followed and confirmed the distances correspondence to the actual distances.

Results:
The quadcopter stopped 3 meters before reaching an obstacle. It then bypassed the obstacle from the right side and continued the flight towards the target. This was all done according to the algorithm and as expected.

Improvements:

In order to increase navigation accuracy, we have to increase the GPS receiver accuracy. It can be done simply by buying a better GPS receiver, with better accuracy.

Future work:

Combine the GPS navigation with an image processing algorithm:
At this way, we can navigate with the GPS to the area, but the recognition of an object and the ability to hover above it with great accuracy will be achieved by a camera and a real time image recognition and processing algorithm.