SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

TECHNOLOGIES AND ALGORITHMS FOR AUTONOMOUS GUIDANCE ON RPAS

Supervisor:	PhD Candidate:
Prof. Marco Balsi	**Carmine Giannì**

Academic Year 2017/18

SUMMARY

1 INTRODUCTION

The use of aircraft as an aid to human activities has become indispensable in many areas. For some years now, next to traditional aircrafts, a steep growth is evident of a new class of aerial vehicle: drones, properly defined as (RPAS Remote Piloted Aircraft System), UAV (Unmanned Aerial Vehicle) or UAS (Unmanned Aerial System). As often happened with other technologies, such devices, first-born with military purposes, over time, have found a cost-effective use in civil applications.

Figure 1-1 DJI Mavic Pro RPAS

The advantages of using a drone with respect to a traditional aircraft are manifold: economic, safety, use in hazardous environments, just to name a few. If the first drones, those used for military purposes, had the size and weight of a conventional aircraft, in recent years, thanks to the evolution of onboard electronics (e.g. the spread of MEMS technology for the construction of gyroscopes, accelerometers, barometers and magnetometers) and materials used for the mechanical structure (composite materials, carbon fiber), it has been possible to make UAVs of dimensions of the order of tens of centimeters and weights even below the kilogram.

They thus created fields of application and use that until a few years ago were unthinkable, their use for indoor applications is an eloquent example. Currently the use of drones, ranges from agriculture to archaeology, drones are also used in scenarios of natural disasters, surveillance or for filming.

Together with the spread of applications in which drones are demanded another need is growing up, the need of an autonomous guided RPAS. In fact, a plethora of applications would be feasible if only a RPAS could get rid of the human pilot.

During these three years of doctorate a lot of steps have been made by the international players in that direction and in the last two years some firms started to publicize drones that can do some automatic tasks, like obstacle detection and target following. However these task are performed correctly only under certain environmental conditions and this pose a limit to the usage of such gizmo in a critical application.

If the research and development divisions of the greatest drone manufacturers are promising always more accurate versions of autonomous RPAS, what keeps this market from take-off is the lack of common regulations between countries.

Being this a newborn sector both national and international regulators don’t know how to move: RPAS are aircrafts, even if their size and weight can be relatively small they represent a hazard to the other flying objects populating the common airspace and moreover, because they fly typically at very low heights they are also hazardous for human ground activities.

So a lot of countries decided for a severe restriction of potentially hazardous RPAS activities like for example autonomous flight.

Anyway the duty of research here is to provide a technology that can be universally considered secure in order to earn trust and to be freed by the weights of bureaucracy that keep them from flying free.

In this work I firstly analyzed the state of the art in autonomous guidance, to be honest in three years the scientific progress has been important, so some limits that where true in 2014 are now trespassed. However when I started the doctorate project one of the main limitation that the solutions proposed by research groups had, was the huge weight and size that obstacle detection systems occupied on drones.

So I decided to design an autopilot platform bases on FPGA and DSP to bring a lot of computing power onboard the RPAS with a very low weight impact.

The presence of FPGA guaranteed the possibility of making tough tasks like FFTs calculations.

To test the autopilot capabilities, I had to learn how to pilot a RPAS, so I attended a course of 33 hours held at L'Aquila that gave me the fundamental notions needed by a RPAS pilot. After the course I also learned how to design a RPAS, and I designed a prototype on which I mounted Leonardo.

After the design of Leonardo autopilot I made a research on the current available sensors technologies for obstacle detection

I then implemented a system for altitude above ground level detection and avoidance, in chapter 5 I describe thoroughly the experiments made.

I then focused on integrating more sensor technologies on a single multi-sensor system to exploit the particular feature of each technology. The research activity on the multi-sensor system is described thoroughly in chapter 6, several experiments has been done on the ground and onboard a RPAS to test and validate the developed obstacle detection system.

Another research effort has been made also in the direction of integrating RPAS in common airspace, in chapter 7.2 I describe how I, together with Oben s.r.l. (a spin-off of University of Sassari) participated to a call for demonstration made by ENAC for a system to electronically identify and tracking of RPAS in the common airspace.

2 RPAS

RPAS stands for Remotely Piloted Aircraft System, these vehicles also known as drones today are widespread and there are countless applications in which they can be employed.

2.1 RPAS Classifications

RPAS are divided in two main families:

- Multicopters
- Fixed Wing

Multicopters are the heirs of the Helicopters RC models, with their parents they share the ability to hover and to fly freely in the three dimensions. What made the fortunes of these devices with respect to helicopters is the simplicity of the mechanics, because multicopters to move, don't need to vary the angle of the propeller. Moreover having multiple rotors allowed for a smaller diameter of propellers thus making these devices less dangerous to operate than standard helicopters.

Figure 2-1 Multicopter

Fixed Wing are very similar to standard airplanes, their main advantage with respect to multicopters is the flight autonomy, this is due to the fact that they fly thanks to the lift, so they don't need the great amount of energy to keep flying. On the other side however this advantage turns to be a culprit when hovering or 3D maneuvers are needed. A Fixed Wing RPAS like any airplane simply can't hover, because this would take the vehicle to stall and then to fall.

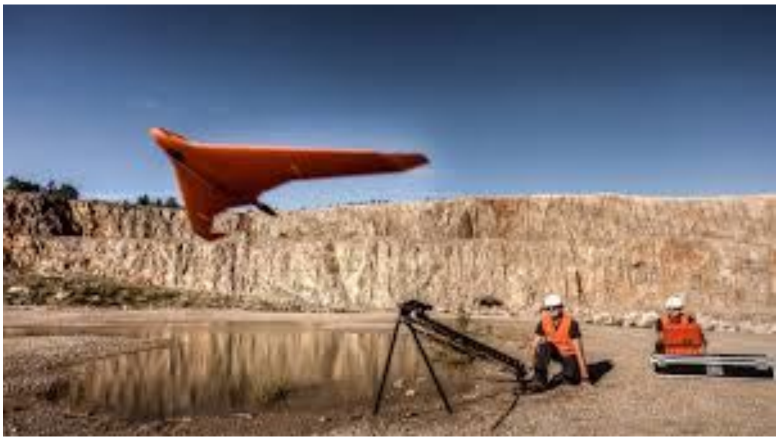


Figure 2-2 Fixed Wing

These characteristics make the two families of RPAS very complementary each other so they are both used time to time depending on the application.

2.2 RPAS Applications

The RPAS market is in continuous growth, like shown in Figure 2-3 the trends are for a doubling of the actual values (2017) only in the next five years, such a pace is incredible for a technology that just five years ago was only known by military.

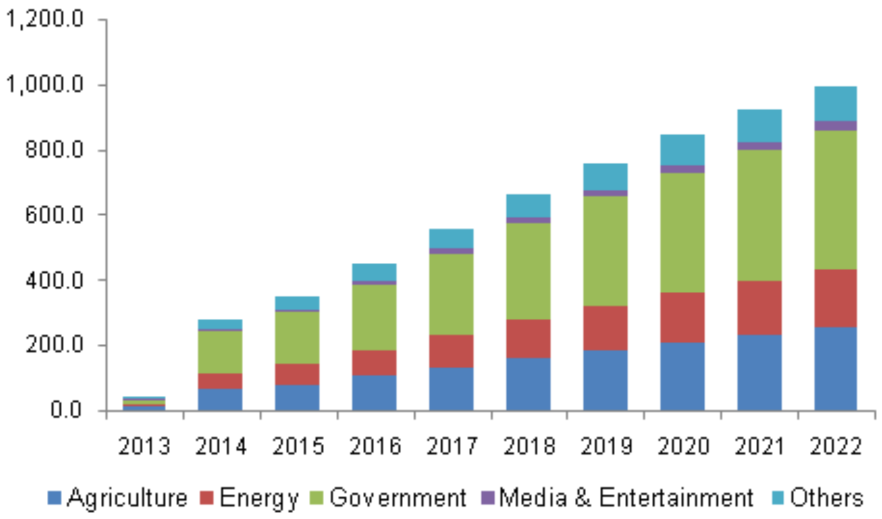
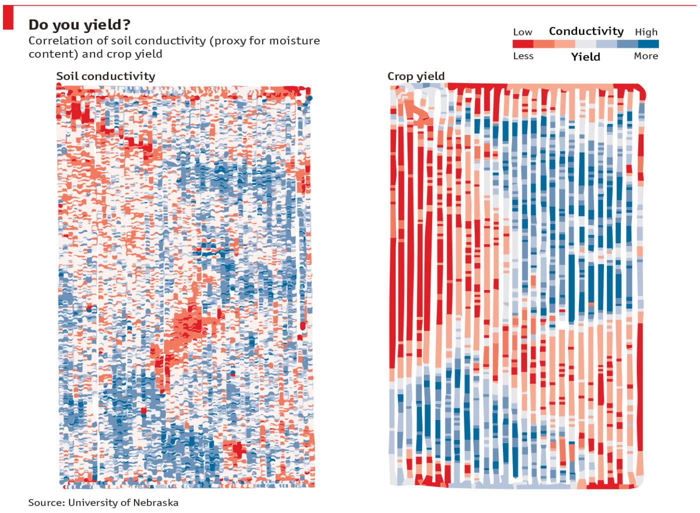


Figure 2-3 U.S. RPAS Market Trends

2.2.1 Agriculture and forestry

Agriculture is one of the most promising fields in which drones are expected to be used more and more intensively. With drones, especially with fixed wings models, it is possible to map extended areas using for example multispectral cameras that can give to farmers important information for example on the soil moisture, this can reduce a lot the waste of water, pesticides and allow to do what is often called “precision agriculture”


Do you yield?
Correlation of soil conductivity (proxy for moisture content) and crop yield
Low
Conductivity
High
Less
Yield
More
Soil conductivity
Crop yield

Source: University of Nebraska

Figure 2-4 Soil analysis with agriculture drone

One important application of drones is in forestry. For example, European Union is financing the project FreshLIFE demonstrating remote sensing integration in sustainable forest management aimed at demonstrating, in four study sites in Italy, integration of data obtained through ground-based forest inventory techniques with remote sensing data, for spatial estimation at small-plot scale of indicators of sustainable forestry management. Our department (DIET) is participating at this project and interesting results are coming from the usage of RPAS and LiDAR.

In 2016, the research group with which I am working published a work within this project, titled "Airborne LiDAR scanning for forest biomass estimation" (Balsi et al., 2016). In this work, we demonstrated that with airborne LiDAR scanning it is possible to reconstruct the Canopy Height Model (CHM) obtained by the difference between the Digital Surface Model (DSM) and the Digital Terrain Model (DTM).

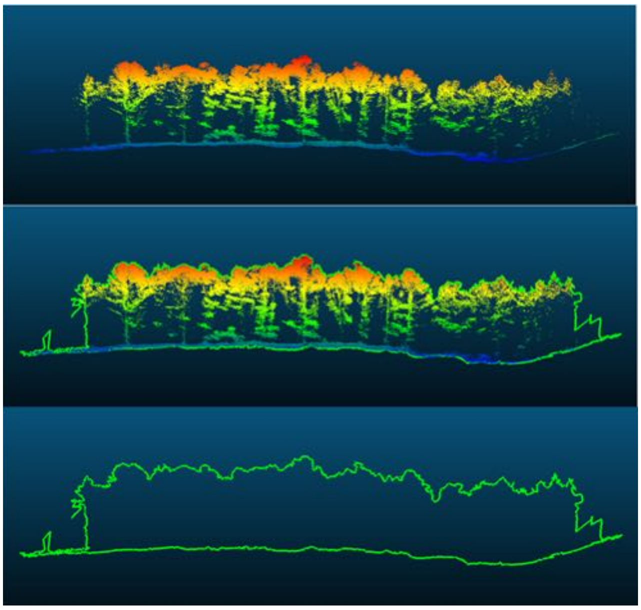


Figure 2-5 CHM extraction

2.2.2 Energy

Another growing sector is that of powerlines monitoring. With thousands of kilometers covered by power lines, it is very important for electrical companies to monitor the status of the lines in order to prevent harmful situations like vegetation falling on the lines like shown in Figure 2-6. Traditionally this work has been carried on with manned aircrafts with inherently high costs, today there is a growing interest in doing such tasks with RPAS thanks to the relative low cost of operation.

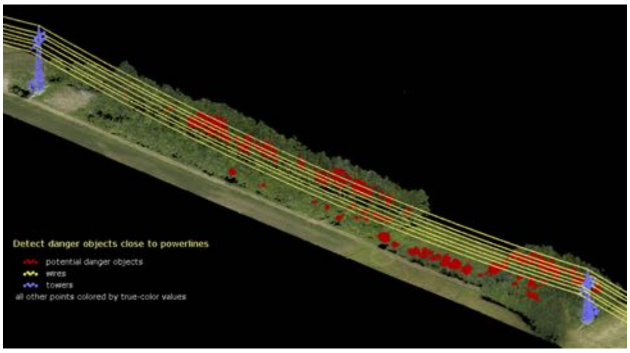

Detect danger objects close to powerlines
potential danger objects
wires
towers
all other points colored by true-color values

Figure 2-6 Powerline monitoring

However, it is not easy to pilot a RPAS in such a critical application, and moreover with the actual regulations the movements of the RPAS are limited to a strict radius around the pilot, so mapping a powerline which spans for kilometers is a very slow task if the RPAS can't act autonomously.

2.2.3 Surveillance

One of the promising fields of applications of RPAS is that one of surveillance – security. As of today, several police forces all over the world are starting to use small UAVs as tactical devices in applications that only few years ago required the use of a manned helicopter.

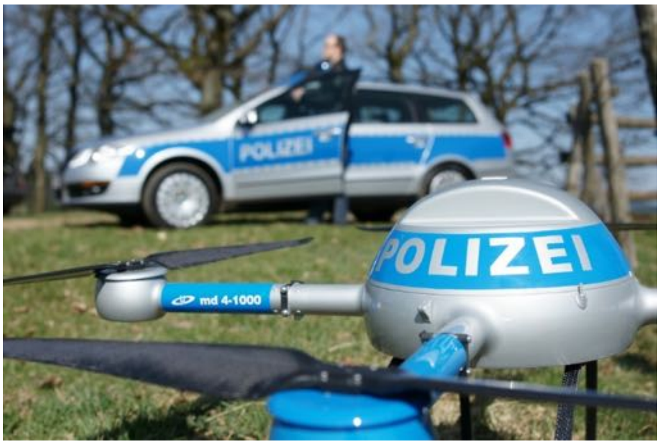


Figure 2-7 RPAS used by Police

RPASs are also starting to be used by watchmen in surveillance applications.

In this field, autonomous flight would be an enabler for applications of home surveillance. One of the issues with video-surveillance today is the inherently high cost of the system when used on extended areas. Indeed, to monitor large areas it is necessary to deploy complex systems with tens of cameras, that leads to high costs of installation and it is also difficult to manage.

An interesting approach is for example that of Sunflower Labs (Sunflower Labs, 2017), it is a Swiss startup that aims to diffuse home UAV surveillance integrated with a traditional motion detection system. They propose to place these sensors in the garden where it is desired to detect intruder's presence. When a sensor detects a person, it sends wirelessly this information to the idle UAV that takes-off and flies to the position of the sensor. The UAV starts to transmit the video signal and the system notifies the owner through an app that will show real time video.

Figure 2-8 Sunflower Labs UAV Home Surveillance

Such a product is interesting even if it is just a concept, the main concerns that are moved today to such applications is that commonly no autonomous flight is allowed, so in order for these applications to spread throughout the world it is necessary to have a common regulation on autonomous flight.

2.3 RPAS Anatomy

To better understand the next chapters is now important to discuss briefly the "anatomy" of an RPAS with its main blocks.

Almost every RPAS is composed by these blocks:

- FCU (Flight Control Unit)
- Battery
- GPS
- IMU (Inertial Measurement Unit)
- RF Receiver/Transceiver

- Motors
- ESC (Electronic Speed Control)
- Propellers

2.3.1 Flight Control Unit

Flight Control Unit also known as Flight Management Unit is the brain of the RPAS.

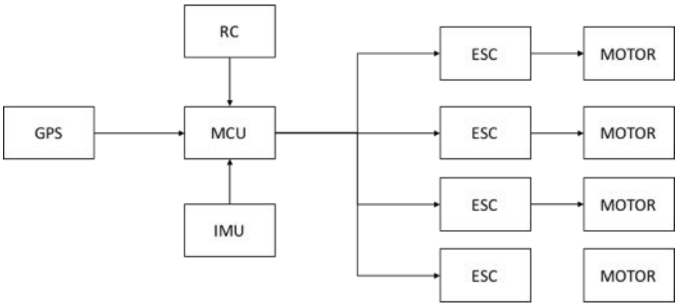


Figure 2-9 Flight Management Unit block scheme

Figure 2-9 shows the typical block scheme of a RPAS FMU.

The Microcontroller Unit (MCU) gets the data coming from the Inertial Measurement Unit (IMU) to calculate the actual attitude of the RPAS, while the GPS is used to get the correct position and speed of the vehicle. With these information, the MCU is capable by means of flight mechanics calculations to establish the right torque to give to each motor in order to get the drone in the desired position.

From the Radio Channel (RC) the MCU gets the information coming from the Ground Station or from the pilot Remote Controller and can also send data out for telemetry purposes.

As of today, there are on the market plenty of Flight Control Units, both Open Source and proprietary ones.

Most famous commercially available Flight Control Units are those from DJI, a chinese manufacturer which produces FCUs spacing from cheaper ones (Naza) to more advanced types (A3, Matrice).

In the Open Source world, also there are a lot of products, but as of today the most diffused are Ardupilot and PX4 (Pixhawk).

While the former was among the firsts FCUs for RPAS the latter is very interesting, because has a more powerful platform (in terms of computation power) and is a project born in the ETH of Zurich. As of 2017 is a very active project with a big community of developers and actually is moving to an even more powerful platform (Pixhawk 2 including and Intel Edison board)

When I started this doctorate project PX4 was in its firsts days and studying that project I found it very promising even if in my opinion it still lacked elaboration power for Sense and Avoid, in fact the STM32 on board is a good MCU/DSP, but it is mostly used for the Kalman Filter IMU calculations leaving the other applications with not so much computing power.

So, I decided to design Leonardo, my FCU based on PX4 but with a FPGA surrounding the STM32 to do the other tasks that the MCU alone couldn’t afford to do.

2.3.2 ESC

The ESC (Electronic Speed Control) has the important task of varying the motor rotation speed. It is powered by the main supply (battery) and provides a three-phase electric power to the motor.

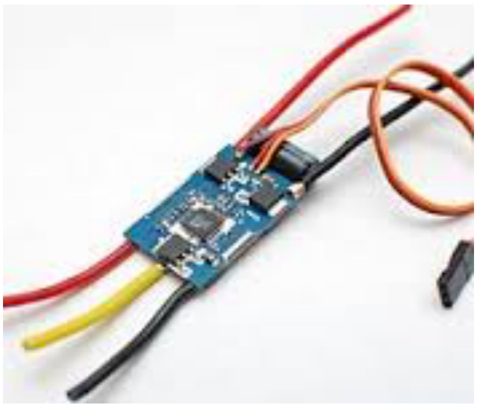


Figure 2-10 ESC, Electronic Speed Control

It is typically driven by a TTL PWM signal coming from the FCU and it can sustain currents in the order of tens of amperes.

Some ESCs are also provided with BEC (Battery Eliminator Circuit) circuitry that allows them to provide regulated supply to other components of the UAV, for example the FCU.

Modern ESC inputs are optically-isolated, this to prevent that noise coming from motors could compromise the signals.

Actually, modern ESCs not only provide power to motors and other circuits but are also provided with safety features: for example, if an ESC detects low battery voltage it will reduce power to preserve the battery, this feature is particularly useful on fixed-wing RPAS, but not desirable on multi-rotors, because could take to odd behavior of the drone.

In order to do such tasks ESCs are usually provided with a microcontroller that reads the pulses arriving from FCU and monitors other parameters like battery voltage and board temperature.

An important parameter on ESCs is the frequency of the PWM signal, on multi-rotors it is particularly important to have a high PWM frequency (400 Hz typical) this is because on multi-rotors the motor rotation determines the attitude of the vehicle. A slow reaction time to the signals coming from the FCU can be fatal to the RPAS.

It seems clear that the ESC in RPAS is an intelligent device and not merely a speed controller, I think that this should take to a next generation of ESCs that is capable of actively monitoring the health of a motor and that one of the propellers. For example, an ESC could be equipped with an optical sensor to monitor the actual speed of the propeller and to transmit back this data to the motor, this could bring to a more accurate speed control and failure detection.

2.3.3 Motor

The motor, together with the propeller is the last ring of the chain, it is the responsible to transform the electric power in air movement.

Today RPAS mount prevalently brushless motors, because they have several advantages over brushed ones, they are more reliable and more efficient, this technology as in fact allowed the spread of multi-rotors, because reliability of a motor is crucial in this RPAS design since in quad-copters a failure of a single motor makes the RPAS to fall down.

Brushless motors divide in:

- Outrunners
- Inrunners

In Outrunners the outer shell spins around its windings which are firm in the center.

In Inrunners the rotational core is contained inside the motor’s can, like a standard motor.

Compared to Outrunners, Inrunners run very fast (well over 10000 rpm), but lack of torque, so to be used they need gears that reduce the spinning speed but increase the torque.

Exactly for this reason Outrunners are the prime choice on multi-rotors, because the main parameter to drive big propellers is the torque.

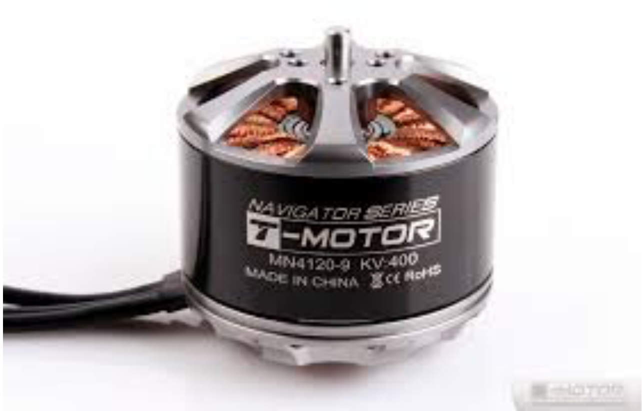


Figure 2-11 Brushless outrunner motor

One important parameter that characterizes a brushless motor is the KV. It indicates the number of rotations that the motor can do for every Volt applied to it, so for example a 400KV motor powered with a 3S LiPo battery will spin at maximum 11.1 * 400 = 4440 rpm.

In Table 1 are shown some experimental data on the AX-4005 D brushless outrunner motor powered with a 3S or 4S batteries and tested with different propellers (Masali, 2015).

Propeller	Voltage(V)	Current(A)	Power(W)	Pull(g)	RPM	Pull efficiency (g/W)
9050	11.5	4.45	51.2	395	6789	7.72
9050	15.3	7.63	117	682	8750	5.84
1047	11.5	7.80	89.7	605	6077	6.74
1047	15.3	11.9	182	978	7565	5.35
1147	11.5	10.5	120	800	5765	6.65
1147	15.3	15.0	230	1160	6853	5.05

Table 1 AX4005D 650 KV motor test data

As it can be seen the consumed power increases with the dimension of the propeller, for a 9 inch propeller at 11.5V the consumed power is 51.2 W, if we put a 10 inch propeller without changing voltage the consumed power increase to 89.7 W.

Another data that is clear from the table is that the higher the propeller diameter, the higher the pull of the motor. This however clashes with the power rating of the motor, a brushless motor, no matter of how big the propeller is, will always try to pull at the maximum speed, so if its power rating is lower than necessary it will go on and heat up until its destruction.

2.3.4 Battery

Batteries are one of the key components that enabled the multi-copter revolution, it is only a few years since the lithium batteries evolved with always more energy density and capability to supply very high currents. The lithium batteries usually found on RPAS are LiPo ones.

LiPo stands for Lithium Polymer, unlike standard Lithium ions battery LiPo batteries use a solid polymer electrolyte (SPE) like Poly Ethylene Oxide (PEO), Poly Acrylonitrile (PAN), Poly Methyl Methacrylate) (PMMA) or Poly Vinylidene Fluoride (PVdF).

If compared with a NiCd or NiMH battery a LiPo battery have about four time the energy density they are also more resistant to impact or punctures w.r.t. other technologies however they are very sensitive to wrong charging/storage, if overcharged or bad stored can easily take fire or even explode.

In Table 2 are shown the main features of LiPo batteries

Energy/weight	130/200 Wh/kg
Energy/volume	300 Wh/l
Power/weight	Up to 2800 W/kg
Energy/price	0,5Wh/US$
Auto-discharge speed	5% a month
Life cycles	>500
Nominal voltage	3.7V

Table 2 Lipo battery data

The charging time is a limit in LiPo batteries because to safely charge a battery one must not exceed 1C (one time the nominal Ah capacity of the battery), this means that a battery charged at 1C will be fully charged in no less than 1 hour. This is a strong limitation, especially with multi-rotors, in fact the typical endurance of a multi-rotor does not go over 30 minutes (optimistically) with a single battery pack, thus forcing the pilot to halt the mission and change battery.

Another main defect of LiPo batteries is their limited endurance in terms of charge/discharge cycles typical batteries don't last over several hundred cycles.

2.4 RPAS Regulations

In order to allow drones to be autonomously guided, the obstacles to dodge are not only physical nor technological but also legal ones.

In the world, the role of regulating flight of RPAS is the responsibility of the various agencies that also regulate the flight of ordinary aircrafts. ICAO is the International Civil Aviation Organization and 191 countries are members of it. The other main organizations are regional ones and are FAA (Federal Aviation Administration) for the USA, EASA (European Aviation Safety Agency) for European Union.

At the moment of writing this thesis the world does not still have a common regulation for the flight of RPAS. Every country has its own rules, that differ a lot each other.

Current common European rules only cover drones weighing above 150 kg, under this threshold national regulations pose a lot of fragmentation and limits to a broad takeoff of the drone market.

However, the scenario is continuously changing and the international organizations are struggling to find a common point on this issue.

For what concerns to Europe, the Single European Sky Air Traffic management Research Joint Undertaking (SESAR) is working for a common regulation on drones, in particular they are working to make drone use in low-level airspace safe and secure.

Just in 2017 SESAR announced that it will put in operation starting from 2019 the "U-Space" project, it will cover the altitudes of up to 150 meters and hopefully will open the way to a common European drone services market. Operators will be requested to electronically identify their drones and they will need to have mechanisms of geo-fencing in order to make the drone fly in a well constrained airspace.

From the paper published by SESAR the U-Space should be:

- **Safe:** safety at low altitude levels will be just as good as that for traditional manned aviation. The concept is to develop a system similar to that of Air Traffic Management for manned aviation.
- **Automated:** the system will provide information for highly automated or autonomous drones to fly safely and avoid obstacles or collisions.
- **Up and running by 2019:** for the basic services like registration, e-identification and geo-fencing. However, further U-Space services and their corresponding standards will need to be developed in the future.

2.4.1 Italian regulations

As stated before, as for 2017 the regulations for drones under 150 kg still fall under national laws. In Italy, the entity that regulates the civil air traffic is ENAC (Ente Nazionale per l'Aviazione Civile).

ENAC classifies RPAS in two main categories:

- Aircrafts with operational weight at takeoff below 25 kg;
- Aircrafts with operational weight at takeoff over 25 kg and under 150 kg.

ENAC distinguish operations in:

- Not critical
- Critical

Are classified as not critical all the operations not involving the flight above congested areas, crowds of people, urban agglomerations or sensitive infrastructures. Is up to the operator to verify that these conditions don't change during flight.

If just one of the conditions cited above changes the operation becomes "critical".

For critical operations, the operator must achieve the authorization from ENAC.

The RPAS must be equipped with a primary command system which software is certified to be compliant to the EUROCAE ED-12 or equivalent specification. The RPAS must also be equipped with systems that keep the control of the vehicle even if datalink should fail.

Moreover, it is important that the RPAS must be equipped with a flight terminator device which is totally independent from the primary control system. In case of problems the flight termination must be done inside the buffer area. The buffer area is not easy to calculate because the operator, knowing the maximum speed of the RPAS with respect to the ground must consider that the flight terminator can be activated by the pilot in no more than 5 seconds from the damage and the possible ballistic trajectory and the effect of the wind.

ENAC also classifies operations in:

- VLOS – Visual Line Of Sight
- EVLOS – Extended Visual Line Of Sight
- BVLOS – Beyond Visual Line Of Sight

The first kind of operations is when the pilot directly sees the RPAS while operating it. For "directly" is intended that the altitude above ground level can't be more than 150 meters and that distance from the pilot can be more than 500 meters.

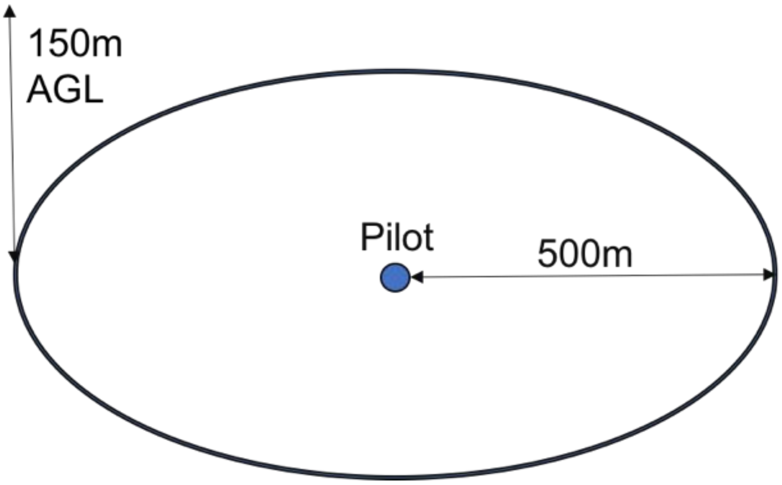


Figure 2-12 VLOS operations

The second type of operation, EVLOS, regards operation where the pilot doesn't directly see the RPAS, he uses for example FPV goggles, but other operators can see the RPAS and can take the control of the vehicle should the pilot have any problems controlling it. EVLOS operations must be directly approved by ENAC even if are not classified as "critical". The flight area is the same of VLOS, theoretically a EVLOS operation could extend for hundreds of kilometers provided that the pilot has got enough observers to follow the RPAS along its path.

The last type of operation is BVLOS, in this case the pilot operates the drone totally out of his sight. This kind of operation was totally forbidden just a few years ago, even today the operations in BVLOS must be approved by ENAC time by time, and can be done only in well fenced areas where no air traffic is possible.

Nevertheless, this last kind of operation is the most interesting and promising, because for the mass diffusion of RPAS services it is fundamental that flight BVLOS is legally possible. The regulators are doing small steps in this direction, but first there is the need for enabling technologies before these operations go further.

3 AUTONOMOUS GUIDANCE

Autonomous guidance of a RPAS refers to its ability to determine its trajectory from one location to another, adapting it in the presence of obstacles.

3.1 State of the Art

When we talk about autonomous driving UAV, one must first specify the environment in which the UAV will operate. To date there is no universal solution for both indoor and outdoor operations. For Indoor Operations an interesting study is that one of Grzonka et al. (Grzonka, 2012) where they propose a fully autonomous quadrotor, they address the issues of operating in GPS denied environments and propose an approach based on SLAM (Simultaneous Localization and Mapping) using LIDAR technology as the main sensor. This approach based on SLAM is interesting, however as they underlined in the conclusion of their study it has a drawback on the computation power required to conduct all the required tasks, they partially solved the problem by establishing a radio link between the quadrotor and the ground station in order to execute the toughest part of the algorithm on a Desktop Computer. Outdoor operations pose different issues to autonomous guidance because while in this case the GPS is typically available, other problems come into play, like light conditions, wind just to mention a few. Several works try to address a specific aspect of Outdoor autonomous guidance I tried to find the ones that try to fully address this problem and I find interesting the works of Nieuwenhuisen et al. (Nieuwenhuisen, 2013) (Nieuwenhuisen, 2014). They propose an autonomous octocopter with different kind of sensors on-board in order to try to cope with the different kind of obstacles present in an outdoor environment. A multi-resolution map is created and updated by the information coming from the different sensors available on board (Stereo Cameras, Laser Scanner and ultrasonic sensors, a reactive collision avoidance layer accounts for fast UAV and environment dynamics. Their issue here is that for all the calculations required they carry a Core-i7 board at 2.7GHz, that force them to mount it on an octocopter because of its weight. Being this research field very active at the moment of writing this thesis other promising works have to be mentioned to possibly provide a more complete image of the scenario, a very promising technology that can reduce the resources needed for autonomous guidance is that one of the Event Cameras, described in section 3.4.5.

3.2 Sense-And-Avoid

Sense-And-Avoid ability for UAVs is the unmanned equivalent of the See-And-Avoid ability for human pilots. The FAA (Federal Aviation Administration) responsible of civil aviation in the USA defines See-And-Avoid in its regulations (14 CFR Part 91.113): “When weather conditions permit, regardless of whether an operation is conducted under instrument flight rules or visual flight rules, vigilance shall be maintained by each person operating an aircraft so as to see and avoid other aircraft”.

See-And-Avoid is not an easy task a pilot is required to:

- Detect conflicting traffic

- Determine Right of Way

- Analyze Flight Paths

- Maneuver
- Communicate

Sense-And-Avoid is to date an open issue and is subject of study and research. On UAVs this is a challenge even more competitive because of the limitations that a drone imposes to the usable technologies. On board a small UAV in fact there are space, payload (the maximum payload transportable) and power supply limitations. These constraints force you to find a trade-off in the choice of sensors, algorithms and available computing power.

Sense And Avoid skills required for UAS are even more restrictive, in fact a UAS typical application is usually in aerial spaces where obstacles are present almost always, and are often moving, not-collaborative ones. For all these reasons the issue of SAA on this type of devices is a challenge from both algorithmic and technological point of view, UAVs are required to deal with obstacles with characteristics very different from each other, and very often the kind of obstacle that presents itself in front of the vehicle affects the choice of the sensor.

In fact, the detection of a small obstacle presents completely different problems than an extended obstacle. In addition to the size of the obstacle other parameters come into play such as its color, its movement and its morphology.

In order to deal with different scenarios and obstacle types, often a single kind of sensor is not enough, radars for example are good active devices for obstacle detection but show limitations dealing with very small obstacles, optical flow sensor on the other side are very good in small obstacle avoidance but not behave as well with larger obstacles and suffer in poor light conditions.

3.3 AMSL vs AGL

A less obvious kind of obstacle is the terrain itself, in order to know their altitude nowadays UAVs are equipped with a barometer that measures pressure which indirectly relates to the altitude above mean sea level (AMSL). A problem with this measurement is that it doesn't take into account the terrain morphology and orography. In order for the UAV to avoid this kind of obstacle it is required that it keeps the correct distance from the ground by means of a sensor that measures its altitude above ground level (AGL).

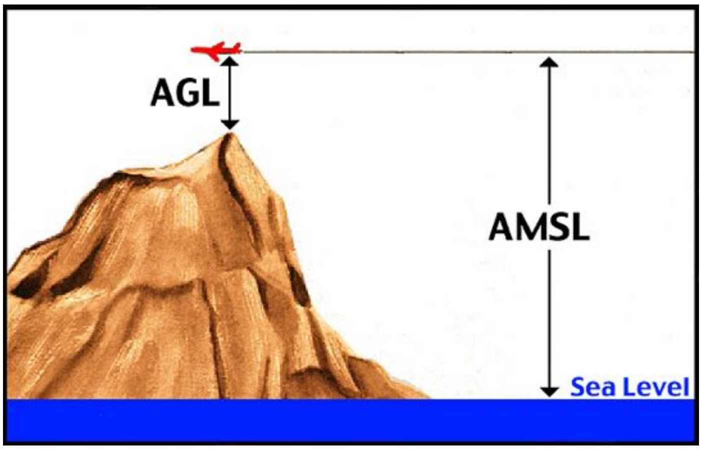


Figure 3-1 AGL vs AMSL

3.4 Sensors Technologies

The heart of every Sense and Avoid system is undoubtedly constituted by its sensors,

3.4.1 SONAR

SONAR (SOund Navigation And Ranging) uses sound waves reflection to detect a target. It is maybe the first modern technology used for ranging purposes, its use is very diffused in a lot of applications. Its principle of operation bases on knowing the speed of sound in the air. By measuring the time a sound wave takes to travel towards an object and to come back when reflected one can easily measure the distance from the object. In practical implementations however SONAR are also provided with a temperature sensor to compensate the change of the speed of sound in the air due to temperature drift.

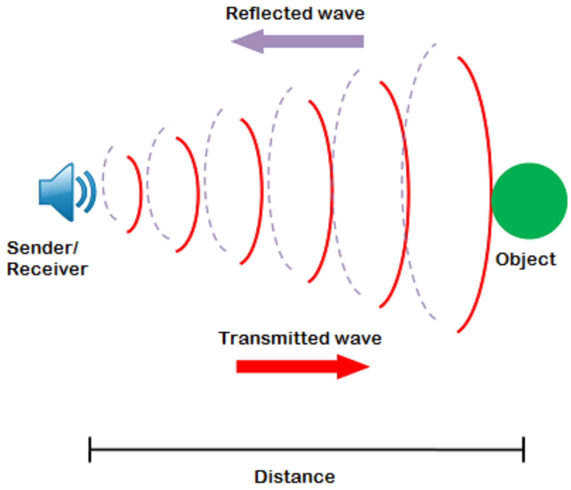


Figure 3-2 SONAR principle of operation

Using a SONAR with quadcopter seems a viable choice since it is cheap, low in weight and able to detect obstacles that are difficult to see with visual technologies like mirrors and windows. However, SONARs must be used carefully since they suffer the turbulences induced by propellers in flying drones. I personally experimented this problem mounting a SONAR on a quadcopter and seeing that when the quadcopter was flying the SONAR always detected an obstacle at 2 meters even if that obstacle didn’t exist at all.

It is very important that the placement of the SONAR is made far away from the propellers airflow and that the SONAR is designed to filter that noise.

A good choice for drone applications are MaxBotix SONARs, they use ultrasonic waves in the range of 40 KHz and can detect obstacles up to 7 meters depending on their size.

Figure 3-3 MaxBotix SONAR

For example the MB1242 is characterized by a high FOV greater than 90° of aperture. SONARs are most efficient on hard surfaces, and have relatively wide main lobe. Normally, they are rather limited in range.

3.4.2 RADAR

RADAR (RAdio Detection And Ranging) is a technology that uses electromagnetic waves to measure the range, the velocity and the angle of a target. At a first glimpse the working principle is quite similar to that one of the SONAR, but the usage of radio waves takes with it a lot of differences with respect to the SONAR technology.

RADAR technology is based on the physical phenomena of backscattering of the electromagnetic radiation when it hits an object whose dimensions are greater that the wavelength of the incident radiation. The backscattered radiation can be detected by a receiving antenna after an amount of time equal to two times the time that the electromagnetic wave takes to go from the antenna to the target. By knowing that electromagnetic waves propagate at the speed of light it is clearly possible to calculate the distance to the target.

The equation that describe the relation between transmitted and received signal is called RADAR equation:

$$P_r = \frac{G_t G_r \sigma \lambda^2}{(4\pi)^3 R^4} P_t$$

where:

P_r is the received power

P_t is the transmitted power

G_r is the antenna gain of the receiving antenna

G_t is the antenna gain of the transmitting antenna

σ is the RCS (radar cross section)

λ is the wavelength of the electromagnetic wave

R is the distance between sensor and object

As it is clear from the equation the receive signal power is reverse proportional to the fourth power of range and it is directly proportional to the RCS of an object.

RCS is a parameter that indicates how much a target is a “good target” for a RADAR.

RCS is measured in square meters (is a surface) and it is frequency dependent.

In Table 3 is shown the RCS of several kind of targets at 24 GHz, a typical frequency for RADAR devices.

Object	**RCS (Radar Cross Section)**
Automobile	10 m^2
Metal sheet of 1m^2	More than 100 m^2
Ship	More than 1000 m^2
Human Being	0.5 – 1 m^2
Tree	More than 1 m^2

Table 3 RCS of various objects at 24 GHz

As it is clear from the Table a human is not a good RADAR target if we compare it to a metal sheet or a car. The physical explanation of such differences stems in the penetration of the electromagnetic wave through different kind of materials.

While metal is not penetrated by microwaves plastic, wood clothes are easily penetrated, and so are not "seen" by a RADAR. While water is not a good target, with microwaves being adsorbed by it. This explains why in submarine applications RADARs are not used and SONARs are preferred.

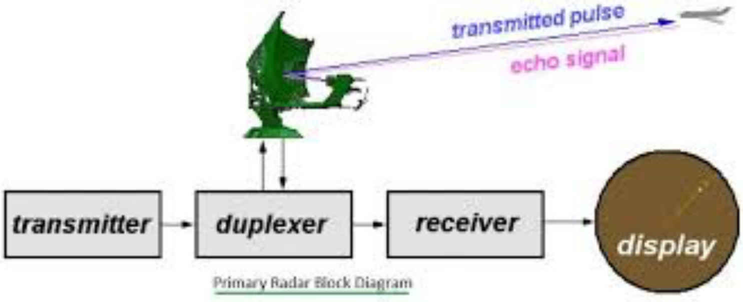


Figure 3-4 RADAR block diagram

Like shown in Figure 3-4 the radar system is basically composed by a transmitter, a receiver, one (monostatic) or more antennas (bistatic/multistatic) and a duplexer, a device that alternately switches the antenna between transmitter and receiver (in monostatic RADAR), to prevent the receiver from being destroyed by the high-power signal coming from the transmitter.

RADAR can be divided in:

- Continuous Waveform
- Frequency Modulated Continuous Waveform
- Pulsed

3.4.2.1 Continuous Waveform RADAR

This kind of RADAR is the simplest one, since it simply transmits a continuous electromagnetic wave at a known frequency and measures the perturbation of the received wave. The received “perturbation” is at a frequency due to the Doppler effect:

$$f_d = 2f_c \frac{v}{c} cos\theta$$

where:

f_d is the Doppler frequency

f_c is the carrier frequency

v is the speed of the moving object

c is the speed of light

θ is the angle between the RADAR-Target line and the actual direction of motion of the Target.

As it is clear from the equation the information produced by this kind of RADAR is relative to the speed of the target and not to its distance. Moreover, since the Doppler effect changes with the direction of motion, continuous wave RADAR give also information about a target approaching or leaving the RADAR. For this kind of information to be produced however it is necessary to have an additional electronic onboard able to recover the sign of the radial speed between RADAR and Target, like shown in Figure 3-5.

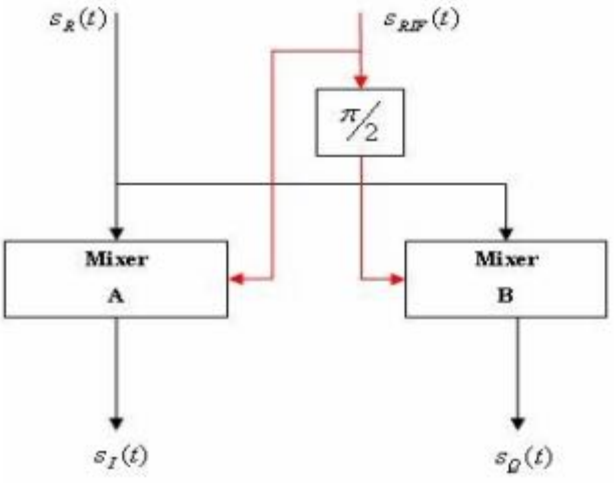


Figure 3-5 Double channel converter

$$S_R(t) = A\cos(2\pi(f_{IF} \pm f_d)t + \phi_0),$$

$$S_{RIF}(t) = B\cos(2\pi \cdot f_{IF}t).$$

Where S_R is the received signal and S_{RIF} is the reference signal at intermediate frequency. At the output of the converter we will have both the in-phase component and the quadrature component. So, if the Doppler frequency is positive the quadrature component will precede the in-phase component and vice versa.

Continuous Wave RADAR can be very compact and require low power to function, so they are used in a lot of applications where is important to detect movement. They are largely used for example in alarm systems, to detect intruders.

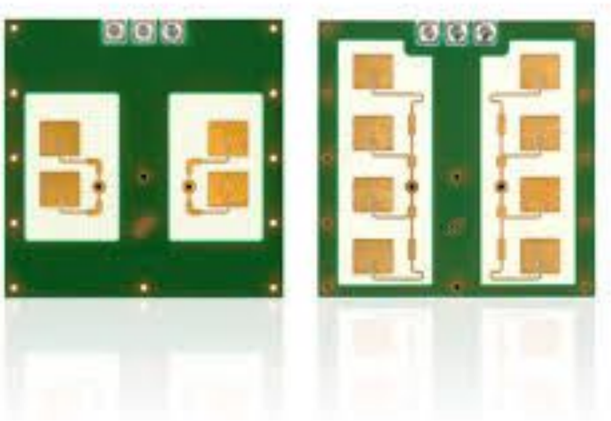


Figure 3-6 CW low-cost RADAR

3.4.2.2 FMCW RADAR

The FMCW RADAR is an evolution of the CW one. In FMCW the Continuous Wave gets modulated in frequency with a particular modulating signal. With this approach, it is possible to detect not just the speed and direction of an object, but even its distance.

FMCW principle to calculate distance is in the fact that frequency of the transmitted waveform continuously change as a function of time, since the reflected signal is delayed with respect to the transmitted one, by simply comparing the received frequency versus the transmitted one distance can be measured.

The mathematics behind FMCW is in this equation that describes the dependence of the measured differential frequency from distance:

$$R = \frac{cTf_D}{2\Delta f}$$

where:

f_D is the differential frequency

Δf is the frequency deviation

T is the period of the modulating wave

R is the distance of the reflecting object

c is the speed of light

To get a linear dependence between differential frequency and distance, the modulating signal must change linearly.

The typical modulation patterns are Sawtooth or Triangular, with the former used for range measurements and the latter for both range and velocity.

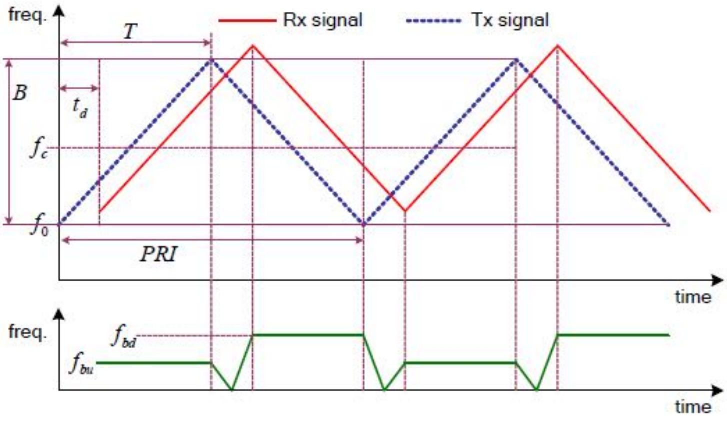


Figure 3-7 FMCW Triangular Wave

Like shown in Figure 3-7 in triangular modulation the transmitter frequency linearly changes, this allow to measure the distance on either the rising front or the falling one, without Doppler effect, the amount of frequency difference Δf during the rising edge is equal to that one in the falling edge for a certain target. When the target starts to move the detected frequency difference changes between rising edge and falling edge, this is due to the Doppler effect that we have yet seen in CW RADAR

$$f_R = \frac{\Delta f_{ud} + \Delta f_{bd}}{2}$$

$$f_D = \frac{|\Delta f_{ud} - \Delta f_{bd}|}{2}$$

In this way f_R, the frequency relative to the range, is given by half the sum of the rising edge frequency deviation (Δf_{UD}) and the falling edge frequency deviation(Δf_{BD}), while f_D, the frequency relative to the speed, is given the modulus of half the difference between Δf_{UD} and Δf_{BD}. One of the limitation of this method is that in presence of multiple reflective object, the measured Doppler frequencies cannot be uniquely related with a single target. This in practical scenarios leads to ghost targets like shown by arrows in

Figure 3-8. To overcome this problem one method is to measure cycles with different slope of the modulation pattern, and to show only the target position that doesn't vary between a cycle and another.

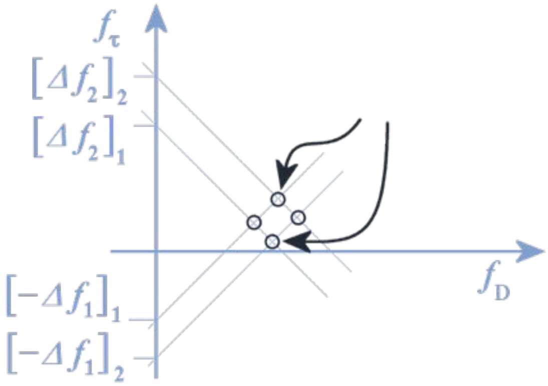


Figure 3-8 Ghost targets in FMCW

An important point when talking of FMCW RADARs is the bandwidth of the modulating signal, the more the bandwidth is, the smaller is the range resolution of a RADAR. From the relationship between signal delay and range, we can easily calculate the range resolution.

$$R = \frac{c\Delta t}{2}$$

To better explain this point we can take as example the 24 GHz ISM Band available in Europe for RADAR applications, this band is 250 MHz wide, so a RADAR operating in these frequencies must be limited at a 250 MHz Bandwidth, with simple calculations we can calculate the range resolution of such a radar:

$$R_{min} = \frac{c\frac{1}{\Delta f}}{2} = 0.6 \text{ m}$$

Figure 3-9 shows the modulating wave of a FMCW RADAR in green and the signal at the IF output, in cyan. In the envelope of the IF output signal is clearly visible a signal at higher frequency, that signal is produced by an obstacle lying at about 6m distance.

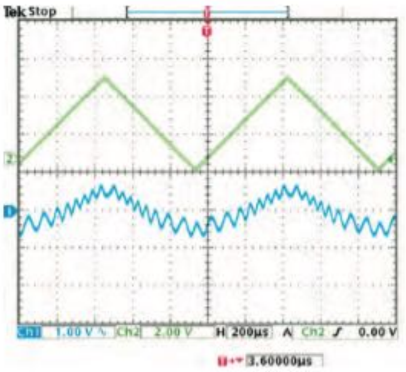


Figure 3-9 FMCW output signal

When dealing with FMCW Radar it is important to estimate the range of frequency that the elaboration circuitry must accept in a specific range of distances. In Figure 3-10 it is shown how output frequencies relate with distance using the FMCW bandwidth as parameter.

As expected the more the bandwidth the more the output frequency at the maximum distance. This fact then affects all the processing chain, because if a higher bandwidth is desirable because its increased range resolution however it requires higher acquisition rates of the Analog to Digital converter stage in the phase of elaboration of the signal.

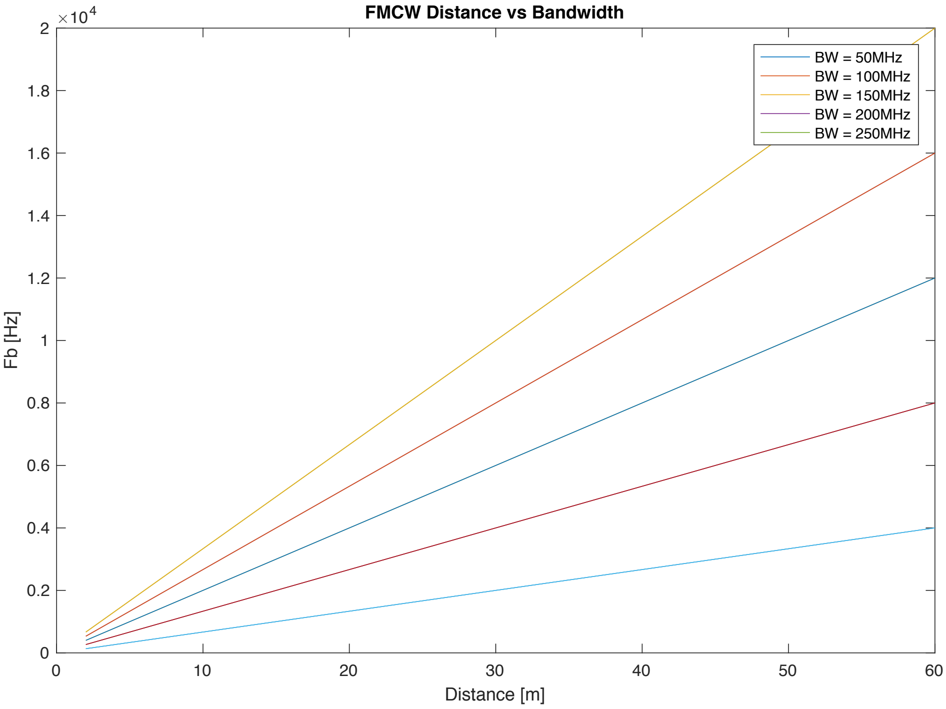


Figure 3-10 FMCW Distance vs Bandwidth

One of the non-idealities that must be accounted when working with RADARs is the noise at the output, produced by mixers that produce some of the input signal in the output, this is well visible in the cyan wave, this noise can be a problem when the target signal is feeble, because in this case a higher gain it is necessary, however the noise can easily saturate the output. To limit this problem, it is necessary to filter out the frequency of the modulating signal.

3.4.3 Stereo Camera

Stereo camera is a well know technology used in obstacle detection systems to create a depth map of the scene, it is a passive system, this fact implies several advantages and disadvantages with respect to other technologies. The main advantage is that the detectable range is not limited by an active illuminator, on the other side it is more prone to errors due to artifacts introduced in the acquired images by environmental lights. The work of Majumder et al. (Majumder, 2015) made an accurate study on the usage of such technology on RPAS applications, they tried the stereo-camera technology both in outdoor and indoor applications and they found that it is a good choice especially in indoor applications, in outdoor ones is still fairly good although presenting some errors in detected object size and distance particularly on small objects.

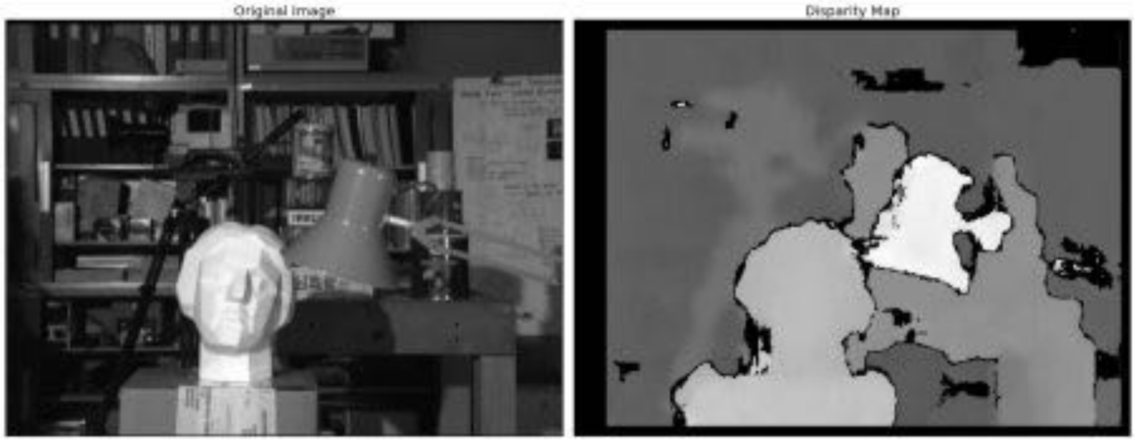


Figure 3-11 Stereo-Cameras disparity map

Figure 3-11 shows on the left a scene shot by two stereo-cameras, on the right side there is the disparity map calculated starting from the two images taken by the left and the right camera. As it can be seen in the figure the disparity map shows with brighter pixels the objects that lie nearer the camera thus producing a depth map.

In order to produce a disparity map of a scene there are several algorithms that can be used, like:

- SAD Sum of Absolute Differences
- SSD Sum of Squared intensity Differences

A limit of stereo-cameras due to the construction of the system of cameras is the maximum depth resolution strictly linked to the distance between the cameras (baseline) and the FOV of the optics used. For a good obstacle detection, it is necessary to have an high baseline distance, and it is clearly not feasible on small RPAS.

Not last another disadvantage is the great amount of computing power necessary to reconstruct the depth map.

3.4.4 Monocular Camera

By simply comparing two frames in time instead of in space (like in the case of stereo-cameras), it is possible to analyze the tridimensional environment using algorithms of Visual Odometry. For these algorithms to work it is necessary to associate the images taken from camera to inertial data coming from IMU. In this way, it is possible to deduce the spacing between points of an image and another taken an instant later.

One of the most effective algorithm is called SVO (Semi-Direct Visual Odometry), it has been developed by the ETH of Zurich (Forster, 2014).

This algorithm is in the middle between feature based methods and direct methods that can exploit information from all image gradients but are too slow to be used in real time applications. This algorithm analyzes the frames coming from camera and uses a sparse approach to select pixels recognized to be at corners or along intensity gradient edges thus reducing a lot the computing time of dense approaches where every pixel is analyzed. The advantage of this algorithm is that it is fast and so can be used even on modern smartphones and still have a good response time down to 10 ms.

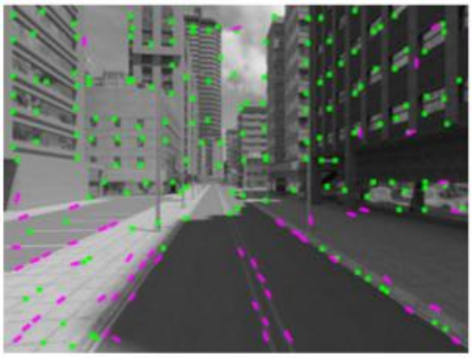


Figure 3-12 SVO algorithm

Experiments have been done with this algorithm of navigation in GPS denied environments and it have proved good, however 10ms elaboration time per frame are still not good for a robust system.

3.4.5 Event-Based Camera

This kind of technology is promising and can have applications in a lot of fields. It is a very recent technology and important companies like Samsung and IBM are investing a lot in it. The first research group to exploit this kind of sensor for RPAS applications is the Robotics and Perception Group of the university of Zurich (Mueggler, 2017). In Figure 3-13 is shown the comparison between standard frame based cameras and event based cameras. From the figure it is clear how and event-based camera works, instead of producing as output a series of frames at a specified frame rate, the event-based camera outputs just the pixels that have changed from a sample to the next one.

An advantage of this kind of cameras is the absence of motion blur during rapid motion, they respond to pixel-level brightness change with a microsecond latency.

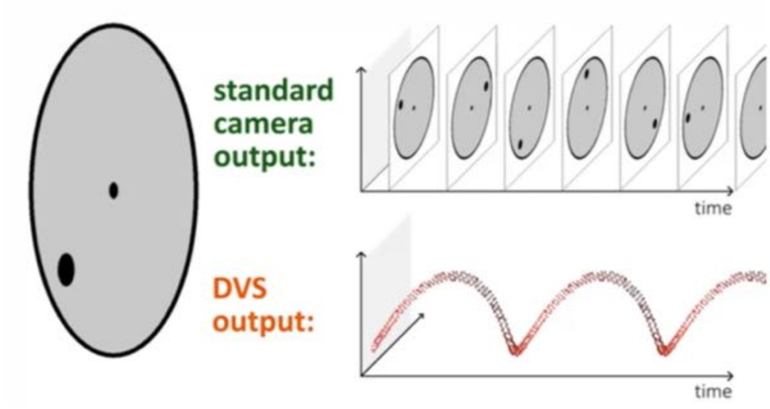


Figure 3-13 Standard vs Event Camera

The heart of an event-based camera is a DVS (Dynamic Video Sensor), in it every pixel has a comparator circuit that compares the brightness seen with a stored one, due to its circuitry, shown in Figure 3-14, the dimension of the single pixel is greater than that of a standard camera, the main part of the space is taken by the capacitance needed to differentiate.

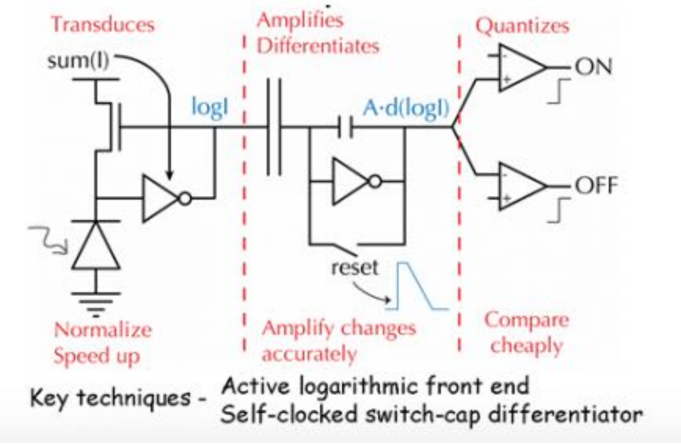


Figure 3-14 Pixel circuitry of DVS

So at least for now these cameras can have relatively small resolutions up to 240x180, but with research this limit is expected to increase.

Another advantage of this kind of camera is its large intra-scene dynamic range, this is because every pixel responds to relative change of intensity. This leads to a dynamic range of approximately 120 dB that dwarves the typical limited dynamic range of normal cameras that is approximately 50 dB.

This is clearly shown in Figure 3-15, where an Edmund gray scale chart is partially exposed to sunlight, the exposed part is at 780 lux, while the shadow is at 5.8 lux. The right figures show pictures taken by a normal camera: as it can be seen the camera must either increase the exposure time leading to an over-exposed scene in the 780 lux zone, or can decrease the exposure leading to an under-exposed scene in the 5.8 lux zone.

The left figure shows the output of a DVS sensor under the same light conditions, the sensor has no problem to detect the whole scene.

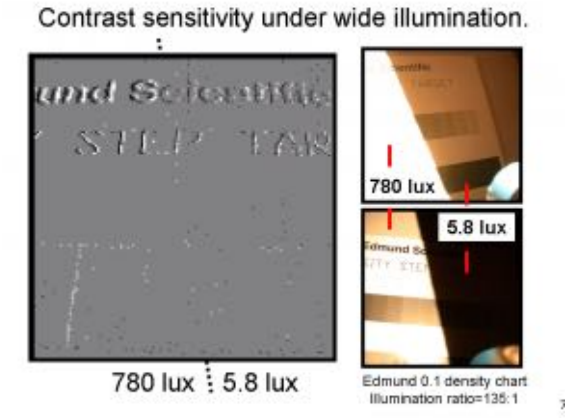


Figure 3-15 DVS contrast sensitivity

3.4.6 Optical Flow

A good definition of optical flow (or optic flow) is that of Horn (Horn, 1981):

“Optical flow is the distribution of apparent velocities of movement of brightness patterns in an image. Optical flow can arise from relative motion of objects and the viewer”.

So the optical flow can be used to detect obstacles, because it gives information about the spatial arrangement of the objects viewed.

Several techniques have been developed to calculate optical flow, one of the widely used is the Lucas-Kanade algorithm (Lucas, 1981).

The algorithm assumes that when observing a sequence of images, if two images are separated by a small time Δt the objects have not displaced significantly, so it looks at the second image looking for a match of a pixel, and it works trying to guess the direction of motion of an object.

So the optical flow equation can be assumed valid for pixels in a window centered at around a pixel P.

$$\nabla I(x,y) \cdot \vec{v} = -\frac{\delta I(x,y)}{\delta t}$$

In nature, optical flow is widely exploited by living beings to move in the surrounding environment. If look at flying insects their senses relay a lot on optical flow, several studies have been made on their behavior and how it could be transferred on robots (Zufferey, 2008) (Fallavollita, 2012) with very promising results.

An interesting device that exploits the optical flow is the PX4FLOW, it is developed by the PX4 team and is thought mainly as a positioning device in zones where the GPS signal is not available.

In the Honegger et al. work (Honegger, 2013) they explain the idea of pointing the optical flow camera towards the terrain, they study the optical flow and guess the direction of movement of the camera, and consequently of the RPAS.

The camera however can't work alone because it needs a distance sensor to quantify the movement in any direction, in their work they used a SONAR, thus limiting the maximum height at few meters above ground level. To estimate the optical flow they adopted the less costly algorithm based on SAD (sum of absolute difference) block matching. Onboard the PX4FLOW is present a Cortex M4 DSP, it comes with integer vector instructions, thus allowing for the computation of the SAD value of four pixels in parallel in a single clock cycle. The SAD value of a 8x8 pixel block is calculated within a research area of +/- 4 pixels in both directions.

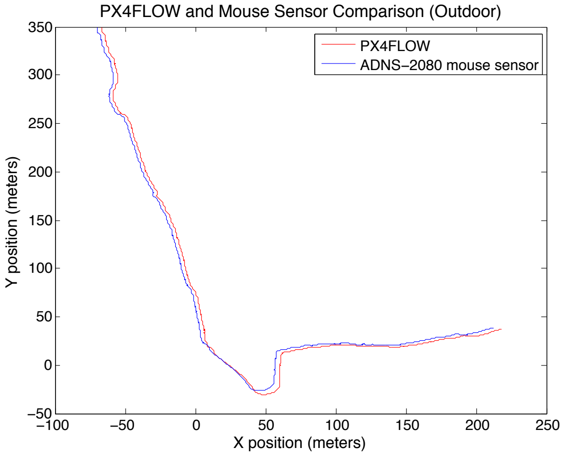


Figure 3-16 PX4FLOW position estimation

In Figure 3-16 is shown a comparison between the PX4FLOW and a mouse sensor in position estimation. As it can be seen the results are comparable with a little drift of the PX4FLOW caused by brightness noise induced by obstacles on the ground.

The Honegger et al. work with the PX4FLOW is a good starting point for this technology and shows interesting results expecially in indoor applications.

Figure 3-17 PX4Flow Camera

3.4.7 LiDAR

LiDAR sensors use a LASER source and analyze the reflected light to measure the distance of an obstacle.

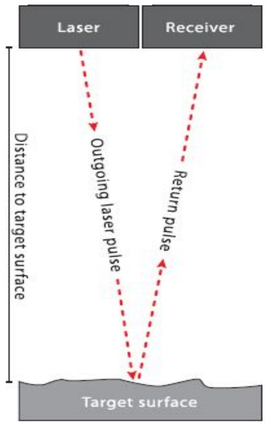


Figure 3-18 Principle of working of a Laser range finder

Due to the shorter wavelength of light LiDARs can be a lot more accurate than RADARs in detecting obstacles, at most the limitations are posed by the reflectivity of the target and by its form factor.

A LiDAR is composed by a Laser generator that can be pulsed or continuous wave (CW), in the former the distance is calculated by estimating the round-trip time of the pulse, in the latter the distance is calculated by estimating the phase deviation between the source signal and the reflected.

A LiDAR can measure a single spot distance, in this case it is also referred as Laser Range Finder, or it can provide a linear scan with several thousands of points per second. The scan is typically performed by using rotating mirrors. Today it is possible to mount this kind of LiDAR on an aircraft and obtain accurate point cloud surveys like the one shown in Figure 3-19 where a building is mapped by a LiDAR mounted onboard a Multirotor RPAS.

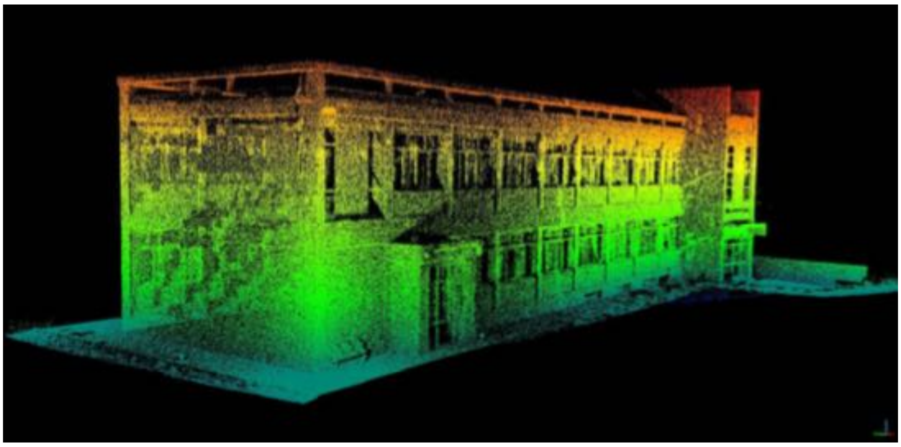


Figure 3-19 LiDAR point cloud of building survey

While the scanning LiDAR would be very good in detecting obstacles, however one of its main limitation is the weight, the Yellowscan Mapper for example is a good scanning

LiDAR with a field of view of 100° and 18500 shots per second, however its weight is of 2.1 kg. This aspect clearly limits the use of scanning LiDAR as obstacle sensing devices on small RPAS, however on bigger aircraft some research has been done with good results (Sabatini, 2014).

The evolution of the LiDAR technology is leading to solid state scanning LiDARs, this technology will provide lighter and faster devices and will allow LiDARs to be used in obstacle avoidance systems even on small drones.

3.4.8 Time-of-Flight Camera

Time-of-flight (ToF) camera is a particular kind of camera that operates by illuminating an area with modulated IR light. By measuring the phase change of the reflected signal the camera estimates the distance for every pixel in the scene thus creating a 3D depth map of the illuminated area. This kind of technology is relatively new compared to the others, it is very similar to LiDAR technology, in fact a time-of-flight camera is simply a class of scanner-less LiDAR, because ToF cameras acquire the entire image in a single shot while LiDAR systems scan the scene point by point. The clear advantage compared to LiDAR is that the frame rate is very high, over than 160 fps.

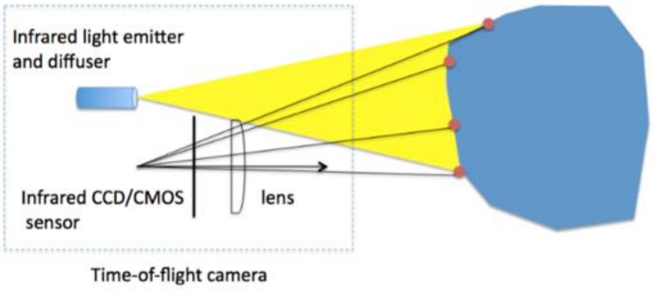


Figure 3-20 ToF camera concept (courtesy of perception.inrialpes.fr)

As of today there are several technologies of ToF cameras:

- **RF-modulated light sources with phase detectors**

  In this kind of cameras, the outgoing light beam is modulated by a RF carrier, then the range is estimated by measuring the phase shift between the carrier and its copy on the receiver side. With this approach, the maximum range is limited by the RF carrier wavelength, that introduces ambiguity over one wavelength distance.

- **Range gated imagers**

  These cameras have built-in electronic shutters that opens and closes at the same rate of the output pulse. At the received side the amount of light received is proportional to the distance between the target and the camera.

- **Direct Time-of-flight imagers**

  These cameras measure directly the time-of-flight of a single Laser Pulse in the path from the camera to the target and back to the camera.

3.4.9 GNSS

It may sound strange to have GNSS between the sensor technologies for Sense and Avoid, however, this technology is ubiquitous when positioning is needed, and in Sense and Avoid applications is very important to know the actual RPAS position in order to avoid obstacles.

GNSS is the acronym for Global Navigation Satellite System, it is the global name that refers to the systems that provide navigation data to users equipped with the appropriate receivers.

Actually, worldwide there are several systems that implement GNSS:

- GPS (Global Positioning System)

  It is maybe the most famous and is provided by the USA

- GLONASS (Global'naja Navigacionnaja Sputnikovaja Sistema)

  It is provided by Russia

- GALILEO

  It is a European project that has entered in full operation in December 2016, it is more accurate than NAVSTAR GPS and should guarantee Europe an alternative to the GPS.

- BeiDou (that stands for Big Dipper in Chinese)

  It is the Chinese Satellite Navigation System, however it is for now just a regional system since with 4 satellites it works only in China

Other countries like India are developing Regional positioning system, because this technology is of military strategic interest.

In fact, the first satellite navigation systems were developed by USA during the cold war, the first system was the Transit and it was thought for military use for navigation of Americans vessels, it provided a 2D positioning, but was inherently slow and impractical for dynamic uses like onboard airplanes. So USA decided to start the GPS project.

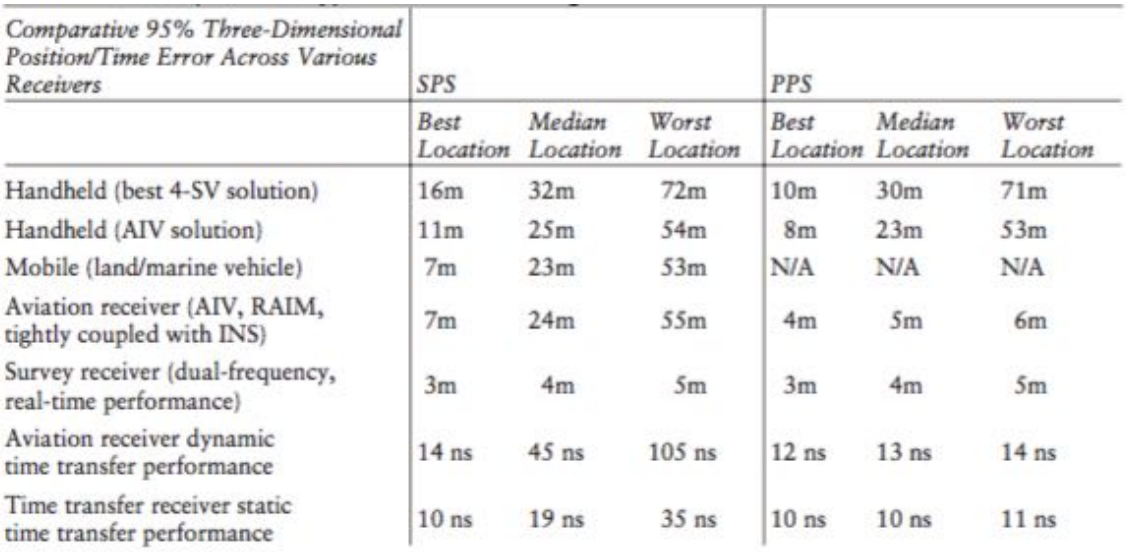
GPS is a complex system formed by a satellite constellation of 24 satellites arranged in 6 orbital planes with 4 satellites per plane (Kaplan, 2006). A worldwide ground control/monitoring network monitors the health and status of the satellites.

GPS can service unlimited users since the receivers operate passively, the system utilizes the concept of one-way time of arrival (TOA) ranging.

In order to lessen the costs of receivers a very accurate time-base atomic clock is onboard GPS Satellites. The GPS signal can work on two frequencies L1 and L2 (1572.42 MHz and 1227.6 MHz) with a technique of CDMA (Code Division Multiple Access). Civil receivers like the ones included in our smartphones use only L1 frequency, while L2 frequency is reserved for military and professional uses.

For a long while, during the cold war and even after the GPS signal for civil use was intentionally degraded in order to guarantee military supremacy to US Army. From year 2000 with an order of then president of the USA Bill Clinton this degradation of the signal was removed, so nowadays military have few advantages with respect to professionals in the use of GPS.

As shown in Table 4 the accuracy of GPS in good civilian receivers can achieve up to 3 meters.



Comparative 95% Three-Dimensional Position/Time Error Across Various Receivers	*SPS*			*PPS*		
	Best Location	*Median Location*	*Worst Location*	*Best Location*	*Median Location*	*Worst Location*
Handheld (best 4-SV solution)	16m	32m	72m	10m	30m	71m
Handheld (AIV solution)	11m	25m	54m	8m	23m	53m
Mobile (land/marine vehicle)	7m	23m	53m	N/A	N/A	N/A
Aviation receiver (AIV, RAIM, tightly coupled with INS)	7m	24m	55m	4m	5m	6m
Survey receiver (dual-frequency, real-time performance)	3m	4m	5m	3m	4m	5m
Aviation receiver dynamic time transfer performance	14 ns	45 ns	105 ns	12 ns	13 ns	14 ns
Time transfer receiver static time transfer performance	10 ns	19 ns	35 ns	10 ns	10 ns	11 ns

Table 4 Comparison of typical performance range across various receivers (from Kaplan)

To achieve better accuracies, it is necessary to go for the GPS augmentation systems that can take GPS accuracy to centimeters, and even millimeters.

The augmentation systems available for GPS are based on DGPS (Differential GPS), it is a method that increase the accuracy of the GPS receiver by taking as reference one or more ground stations at known locations, each station sends GPS information to the receiver through a datalink.

DGPS services can be over a local area or also over an entire continent. WAAS (Wide Area Augmentation System) is a USA service, while for Europe there is the EGNOS (European Geostationary Navigation Overlay Service).

For local area DGPS a promising technology is the RTK (Real-Time Kinematic), contrary to other DGPS technologies it can achieve centimeter-level accuracy not just using the information content of the signal, but through the measurements of the phase of the carrier of the GPS signal.

One strong limitation of GPS that limits its use in critical applications is that the signal can be corrupted/altered voluntary or not, for example due to jamming or simply because of an error in the ground control section. Jamming, intentional or not, are common and are a well-known problem, even if jamming GPS signal is illegal.

However, GPS is evolving with added frequencies and anti-jamming capabilities.

Galileo system seems to be a good alternative to GPS since it addresses right at the limitation of the actual GPS system.

For what concerns to RPAS application this problem is obviously of primary importance since a completely autonomous system must be confident on reliable GNSS signal.

4 LEONARDO AUTOPILOT

4.1 The idea

As I stated before, the idea behind Leonardo Autopilot was born after viewing the state of the art of open source autopilots at that time (2013-2014). I decided to design Leonardo to provide an autopilot for drones with FPGA on board.

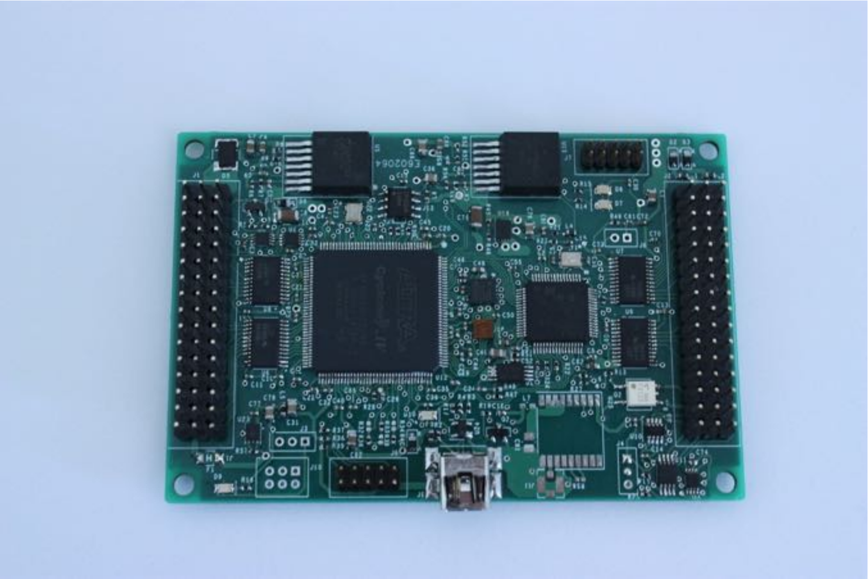


Figure 4-1 Leonardo Autopilot PCB

Let’s see what are the main characteristics of Leonardo:

- FPGA EP4CE22E144 Altera Cyclone IV FPGA with 22k Logic Elements
- DSP STM32F405RGT6 32bit ARM Cortex M4 running at 168 MHz
- IMU LSM9DS0 with 9 degrees-of-freedom containing both Accelerometer, Gyroscope and Magnetometer on a single chip
- LPS331 Barometer

- On board GPS MAX7 from U-Blox
- Up to 12 buffered Outputs for driving ESCs and 12 Inputs compatible with R/C interfaces
- MicroSD card slot
- CAN Transceiver
- RS485 Transceiver
- Up to 3 UART interfaces
- USB interface with Micro USB connector

All these features are integrated in a 60x90mm board in order to be easily mounted on a small RPAS.

When I designed Leonardo I decided to make it compatible with the PX4 stack, in order to exploit the features and the advances yet made by the PX4 team, and that soon revealed as a good choice, because after a work of porting their software on my hardware I could test and debug my Leonardo more easily.

Having an FPGA companion chip brings several advantages:

- Computation loads can be shared between the units
- System is more reliable, since if the DSP fails or locks up, the FPGA can trigger a failsafe procedure
- FPGA being a programmable hardware can easily do calculations that are very heavy for standard CPU architectures

Obviously, there are also disadvantages – drawbacks:

- Overall increase of power consumption, this is true only partially because it depends on the algorithms running on FPGA/DSP
- Increase of development times, because programming an FPGA requires specific skills and debugging FPGA code can be time consuming because of the compilation/simulation times
- It becomes necessary to manage the communication between the two ecosystems (FPGA – DSP), this can be a bottleneck and must be carefully evaluated.

On this last point in particular I concentrated my attention, because there was the risk of having two powerful units not communicating each other. So I included ten lines of interface between the DSP and FPGA, four of these lines are dedicated to an SPI interface, while the others are GPIO. The fulcrum of the communication between the FPGA and the DSP is precisely the SPI, this choice has been taken mainly because the SPI interface is the fastest one available on STM32.

The PCB layout of the Leonardo autopilot also has been somewhat complicated to route because of its density, the choice has been to use a 4 layers PCB with inner layers of ground and power routing, the technology used for passive components is 0402, this has been for me a bit challenging because then I soldered every passive component by hand. I also had to buy a hot air station to solder QFN components (IMU and barometer) that else couldn't be soldered.

In order to get everything working I had to do a porting of the PX4 stack to Leonardo, the porting has to be made since Leonardo has got some hardware differences when compared to Pixhawk. The main difference is the IMU that in Leonardo is all included in a single module, LSM9DS0, the other difference is the barometer, at the moment of design I chose

LPS331, a new MEMS chip from ST that on paper had better performance than the barometer used by Pixhawk. This choice has revealed not so good indeed, because the component has got several accuracy problems and has been removed by ST from its linecard.

So I had to write the drivers for IMU and Barometer.

4.2 FPGA code

On the FPGA side I had to write the interface between the remote control and the STM32, the receiver that I used (Turnigy 9xr) has got separated output for every channel input, since I have a limited number of pins for communication between FPGA and STM32 I chose to write the interface for conveying single channel information into a PPM stream. PPM, also known as Pulse Position Modulation, is a standard often used in RC receivers to pass multiple channels into a single channel, a good representation of this modulation is depicted in Figure 4-2, as you can see the distance between one pulse and the sequent is the PWM duty cycle of that channel.

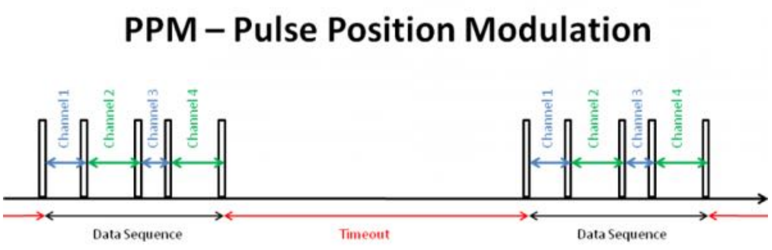


Figure 4-2 Pulse Position Modulation

The advantage to have this step integrated in FPGA is that you can eventually choose to change the behavior of the RC, or check that all data is good before routing it to the DSP.

4.2.1 NIOS 2 experiment

During the first debugging phase I decided to test the FPGA capabilities and pilot all the drone from it. It has been a good exercise and helped me to prove that everything was working as expected.

Cyclone IV FPGAs come with an embedded RISC 32 bit microcontroller IP core that can be programmed inside the logical elements alongside with the other logic its name is Nios 2.

The IP core is generated inside the Altera Quartus II interface (QSys) and the user can choose the features to enable in it. Obviously the more the features the more the space occupied inside the FPGA, so it is important to constrain opportunely the dimensions to leave enough space for other eventual applications. For this application, I chose to enable just the minimum features required to test the basic Leonardo functionalities:

- CPU
- 32KB On Chip Memory
- 32 bit Timer
- 2 SPI interfaces

The SPI interfaces are required to read the IMU values, one is for the barometer, the other is shared between the accelerometer, the gyroscope and the magnetometer.

While the 32 bit timer is necessary to handle the control tasks.

So, I’ve written a sample C code with a PID controller to drive the ESCs a SPI interface to read the IMU and an interface to read the input RC commands.

Total logic elements	2676/22320 (12%)
Total combinational functions	2433/22320 (11%)
Dedicated Logic registers	1642/22320 (7%)
Total registers	1644
Total pins	30/80(38%)
Total memory bits	273408/608256(45%)
Embedded Multiplier 9-bit	0/132 (0%)
Total PLLs	1/4 (25%)

Table 5 FPGA Occupation summary

4.3 PX4 Autopilot stack

The PX4 autopilot as already said is an open-source autopilot designed in the ETH of Zurich by the research team of Lorenz Meier. I decided to make my autopilot compatible with this stack because it is thought for research applications, it is fully editable and there is a wide community working everyday on it.

The block scheme on how the autopilot works is shown in Figure 4-3. The autopilot estimates its position and attitude through inertial sensors, GPS and eventually computer vision, a position estimator then is used to feed the navigator block and the attitude estimator block (actually a Kalman filter). The estimated attitude and position are then used as an input to the position and attitude control blocks, these two blocks go to the mixer block which distributes power between motors.

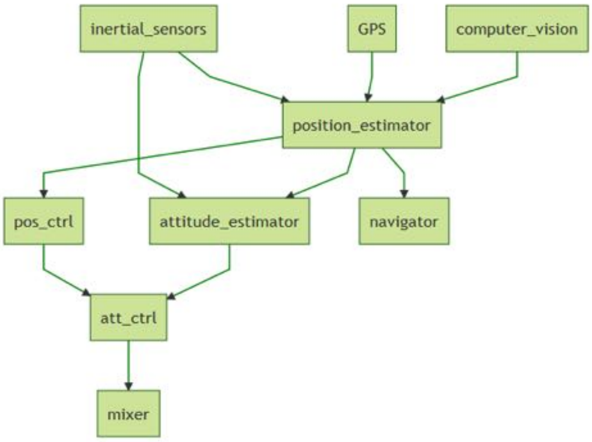


Figure 4-3 PX4 block scheme

As shown in Figure 4-4 the navigator block is responsible to define the route of the RPAS, it can be fed directly by the user through the commander block that translates the commands of the Remote Controller into instructions to the navigator, or it can be fed with instructions of a flight-plan defined by the user before the flight.

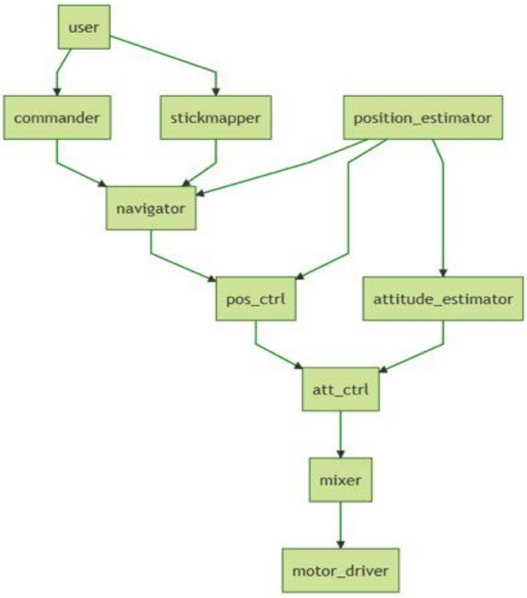


Figure 4-4 PX4 block scheme: navigator

As a feedback to the commander block and to the navigator block, the position and attitude estimators, and the mixer block send their live data to the Mavlink block like shown in Figure 4-5.

Mavlink is a protocol for telemetry thought for RPAS, it is better described in section 7.1. Through Mavlink all the information useful for tracking the drone are sent to the ground station.

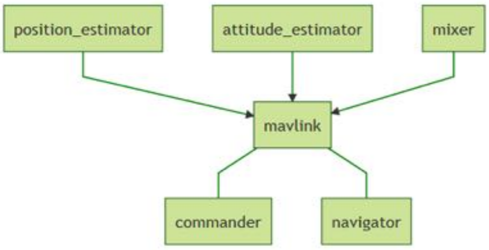
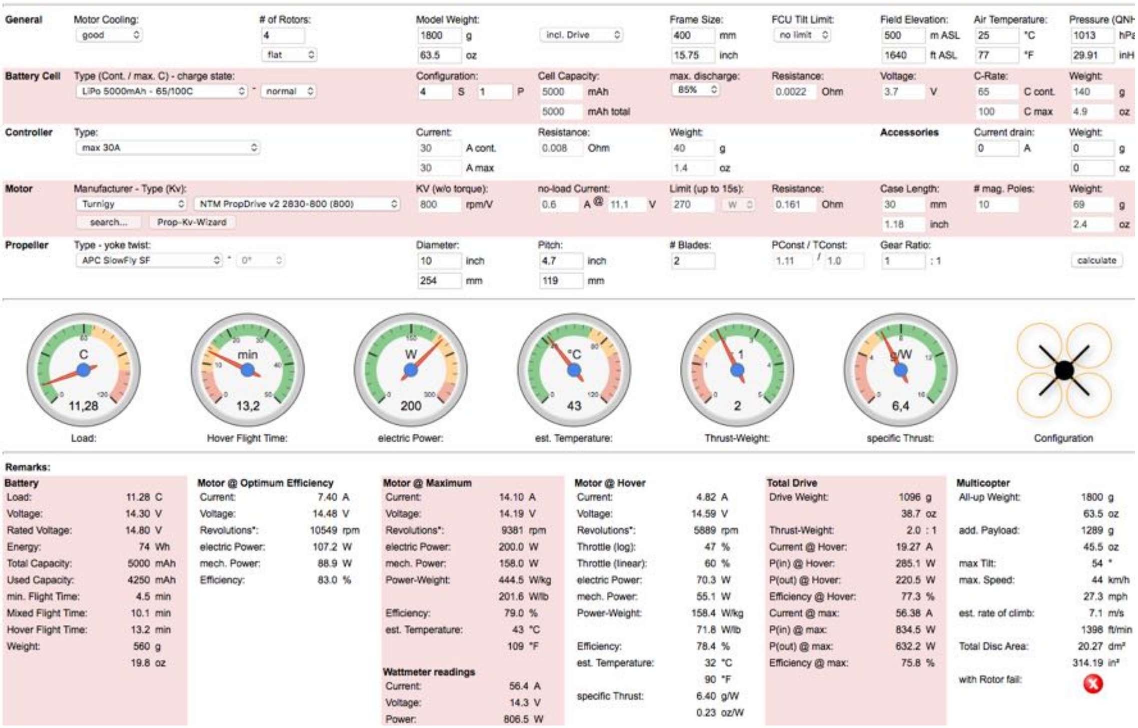


Figure 4-5 PX4 block scheme: Mavlink

4.4 Quadcopter design

To test the capabilities of the Leonardo autopilot I built a quadcopter, I started with the dimensioning of the system and, for the calculations I used **ecalc** an online tool that helps the dimensioning of a quadcopter, like shown in Figure 4-6 the tool allows to insert the desired configuration and through simple gauges indicates if a model is well dimensioned or will have problems in flight, problems may come from motors heating too fast, or ESCs too small for the current driven from motors.

The calculator gives also an idea of the maximum payload that the multicopter can carry with the chosen motors/propellers.



General
Motor Cooling: good
# of Rotors: 4 flat
Model Weight: 1800 g / 63.5 oz incl. Drive
Frame Size: 400 mm / 15.75 inch
FCU Tilt Limit: no limit
Field Elevation: 500 m ASL / 1640 ft ASL
Air Temperature: 25 °C / 77 °F
Pressure (QNH: 1013 hPa / 29.91 inH

Battery Cell
Type (Cont. / max. C) - charge state: LiPo 5000mAh - 65/100C - normal
Configuration: 4 S 1 P
Cell Capacity: 5000 mAh / 5000 mAh total
max. discharge: 85%
Resistance: 0.0022 Ohm
Voltage: 3.7 V
C-Rate: 65 C cont. / 100 C max
Weight: 140 g / 4.9 oz

Controller
Type: max 30A
Current: 30 A cont. / 30 A max
Resistance: 0.008 Ohm
Weight: 40 g / 1.4 oz

Accessories
Current drain: 0 A
Weight: 0 g / 0 oz

Motor
Manufacturer - Type (Kv): Turnigy - NTM PropDrive v2 2830-800 (800)
search... Prop-Kv-Wizard
KV (w/o torque): 800 rpm/V
no-load Current: 0.6 A @ 11.1 V
Limit (up to 15s): 270 W
Resistance: 0.161 Ohm
Case Length: 30 mm / 1.18 inch
# mag. Poles: 10
Weight: 69 g / 2.4 oz

Propeller
Type - yoke twist: APC SlowFly SF - 0°
Diameter: 10 inch / 254 mm
Pitch: 4.7 inch / 119 mm
# Blades: 2
PConst / TConst: 1.11 / 1.0
Gear Ratio: 1 : 1
calculate

Load: 11,28
Hover Flight Time: 13,2
electric Power: 200
est. Temperature: 43
Thrust-Weight: 2
specific Thrust: 6,4
Configuration

Remarks:

Battery		
Load:	11.28	C
Voltage:	14.30	V
Rated Voltage:	14.80	V
Energy:	74	Wh
Total Capacity:	5000	mAh
Used Capacity:	4250	mAh
min. Flight Time:	4.5	min
Mixed Flight Time:	10.1	min
Hover Flight Time:	13.2	min
Weight:	560	g
	19.8	oz

Motor @ Optimum Efficiency		
Current:	7.40	A
Voltage:	14.48	V
Revolutions*:	10549	rpm
electric Power:	107.2	W
mech. Power:	88.9	W
Efficiency:	83.0	%

Motor @ Maximum		
Current:	14.10	A
Voltage:	14.19	V
Revolutions*:	9381	rpm
electric Power:	200.0	W
mech. Power:	158.0	W
Power-Weight:	444.5	W/kg
	201.6	W/lb
Efficiency:	79.0	%
est. Temperature:	43	°C
	109	°F
Wattmeter readings		
Current:	56.4	A
Voltage:	14.3	V
Power:	806.5	W

Motor @ Hover		
Current:	4.82	A
Voltage:	14.59	V
Revolutions*:	5889	rpm
Throttle (log):	47	%
Throttle (linear):	60	%
electric Power:	70.3	W
mech. Power:	55.1	W
Power-Weight:	158.4	W/kg
	71.8	W/lb
Efficiency:	78.4	%
est. Temperature:	32	°C
	90	°F
specific Thrust:	6.40	g/W
	0.23	oz/W

Total Drive		
Drive Weight:	1096	g
	38.7	oz
Thrust-Weight:	2.0	: 1
Current @ Hover:	19.27	A
P(in) @ Hover:	285.1	W
P(out) @ Hover:	220.5	W
Efficiency @ Hover:	77.3	%
Current @ max:	56.38	A
P(in) @ max:	834.5	W
P(out) @ max:	632.2	W
Efficiency @ max:	75.8	%

Multicopter		
All-up Weight:	1800	g
	63.5	oz
add. Payload:	1289	g
	45.5	oz
max Tilt:	54	°
max. Speed:	44	km/h
	27.3	mph
est. rate of climb:	7.1	m/s
	1398	ft/min
Total Disc Area:	20.27	dm²
	314.19	in²
with Rotor fail:	☒	

Figure 4-6 ecalc screenshot

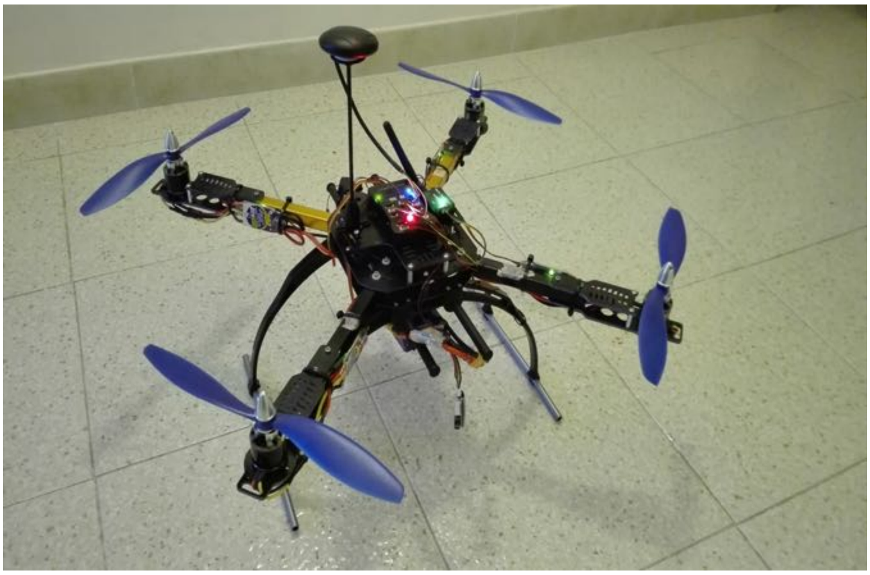


Figure 4-7 Quadcopter built for testing Leonardo Autopilot

In Figure 4-7 the quadcopter that I built is shown, in a first time I tested it with a PIXHAWK open-source autopilot, this gave me the possibility to test its functionalities with a well-known platform. The choice of PIXHAWK was made also because it allows the use of the PX4 autopilot stack, that is the same that I chose for Leonardo.

The main features of the quadcopter are reported in Table 6:

Propellers	10x4.5 inches
Motors	NTM 2830-800kV
Weight	1320g (without battery)
Battery	3700mAh 4S 400g
Maximum payload	1Kg
Hovering Time	11 minutes
Maximum Speed	43 km/h

Table 6 Quadcopter specifications

4.5 Modifications to the PX4 stack

As said above, in order to make the Leonardo autopilot work with the PX4 stack I had to add the drivers for the IMU and the BAROMETER, then I had to change the default output pins for driving motors routing them to the FPGA pins. The input signal coming from the RC controller has been converted with the FPGA in the PPM format and then sent to the STM32.

To comply with the coding standard of the PX4 stack the drivers are derived from the drivers of similar interfaces, the barometer driver for the LPS331 has been derived from the MS5611. The main difference is how the data is fetched from the sensor, here is the snippet of code:

```
/* fetch data from the sensor */
memset(&raw_report, 0, sizeof(raw_report));
raw_report.cmd = ADDR_PRESS_POUT_XL_REH | DIR_READ | ADDR_INCREMENT;
transfer((uint8_t *)&raw_report, (uint8_t *)&raw_report,
sizeof(raw_report));

report.timestamp = hrt_absolute_time();
report.error_count = 0; // not recorded
int32_t pressure_int =  (int32_t)(raw_report.press_h)<<16 |
(int32_t)(raw_report.press_l)<<8 | (int32_t)raw_report.press_xl;
int16_t temperature_int = (int16_t)(raw_report.temp_h)<<8 |
(int16_t)(raw_report.temp_l);
double pressure_dbl = (double)pressure_int/4096.0;//pressure in
millibar
```

```
double temperature_dbl = 42.5 +
(double)temperature_int/480.0;//temperature in C
```

Unfortunately, as said before, the LPS331 revealed to be noisy, so to reduce its noise I decided to filter its output data in this way:

```
/* START - Low Pass Filter */
const double alpha = 0.002;
static bool filter_starting = true;
static double pressure_av_prev = 0, pressure_av = 0;
if(filter_starting)
{
      pressure_av_prev = pressure_dbl;
      if(pressure_dbl > 200 && pressure_dbl < 1200)
      {

            filter_starting = false;
      }
}
pressure_av = alpha * pressure_dbl + (1 - alpha) * pressure_av_prev;
pressure_av_prev = pressure_av;


/* END - Low Pass Filter */
report.pressure= pressure_av;
report.temperature= temperature_dbl;
```

The IMU driver for the LSM9DS0 has been split in two drivers, one for the Gyroscope that is a modified version of the L3GD20 driver, and the other for the Magnetometer and the accelerometer, that is a modified version of the LSM303 driver. The full version of these modified drivers is too long to be included in this thesis, but it is available for the interested reader on my GitHub account https://github.com/gcarmix/leofmu_addons

The code available there is compatible with the PX4 stack and can be easily included in it by patching the original PX4 code.

5 AGL CONTROL SYSTEM

To control the Altitude above Ground Level a drone must be equipped with proper sensors and must elaborate the information coming from the sensor.

As stated before a barometer is useless in this task, so my attention focused on distance sensors, traditional airplanes are equipped with RADAR altimeters, they are good on long range, but for small UAVs since the distances are a lot smaller it is advisable to use a sensor with less range, but with a lot more precision like a laser range finder, the chosen laser range finder, the LidarLite as a range of approximately 40m with an accuracy of few centimeters.

One of the problems that I faced when thinking at an AGL control system is that, in order to calculate the correct trajectory to take, the distance sensor must be pointed forward with respect to the drone otherwise it is impossible to react correctly to abrupt AGL changes.

As a starting point, I decided to mount the LidarLite at 45° angle with respect to the ground in order to have good ground range and good foreseeing capabilities, anyway the LidarLite was mounted on a mechanical “arm” driven by a servo motor in order to change the angle of detection when needed like shown in Figure 5-1. Another advantage to have the sensor mounted on a servo motor is that the angle of measurement can be changed on the fly when the autopilot detects a variation in the pitch of the vehicle in order to detect the terrain always at the same angle.

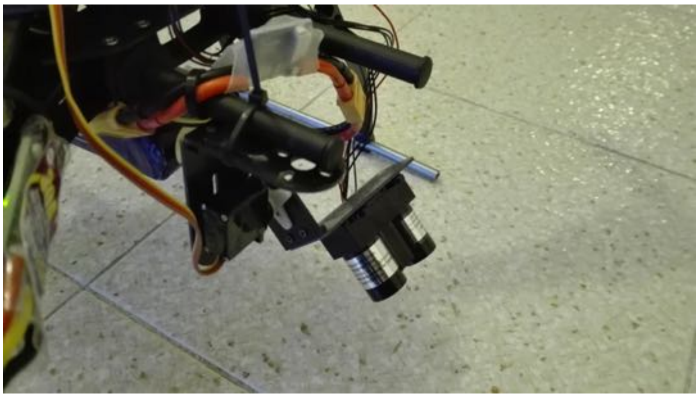


Figure 5-1 LidarLite mounted under a quadcopter for AGL experiment

To demonstrate the correct working of the system I tested it by passing over a pile of books arranged like a four steps stair on the ground, the results are good like shown in Figure 5-2.

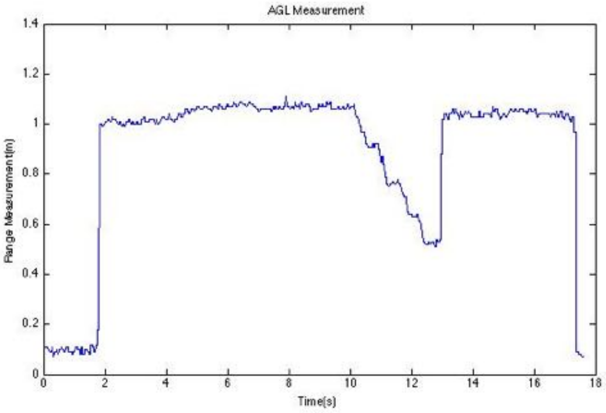


Figure 5-2 Test passage with AGL measurement

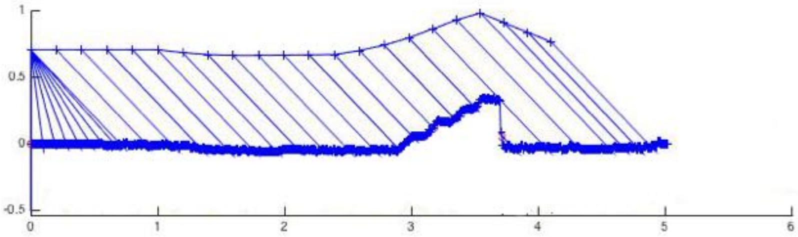


Figure 5-3 Model of Altitude correction

I made a second experiment with the AGL system and the RPAS passing over vegetation, this time I compared the barometer altitude with the AGL measurement.

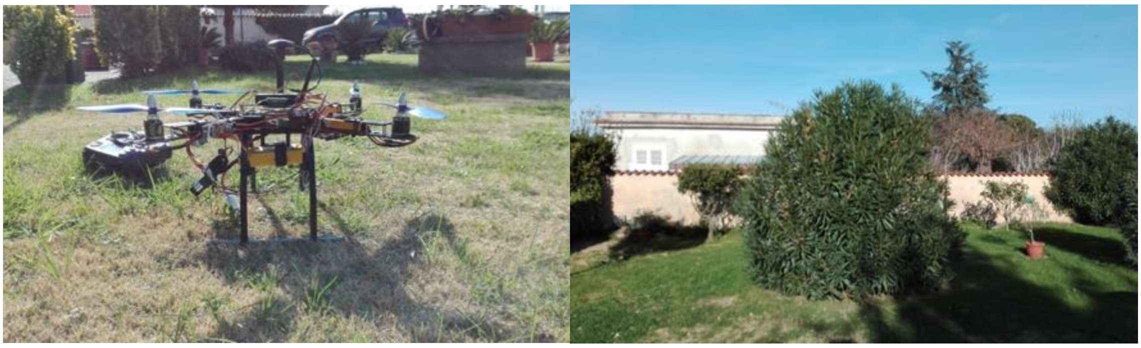


Figure 5-4 AGL Experiment scenario

In Figure 5-5 the comparison between barometer and AGL is shown. As it can be seen from the graph, the measured data of the AGL sensor corresponds to that of the barometer less than an offset that is due to the inclination of the AGL sensor.

In the experiment, I did approach the RPAS to the obstacle (the oleander shrub), and the moments when I approached the RPAS are well visible in the graph, they in fact correspond to a sudden loss in AGL distance not followed by a loss on the barometer height.

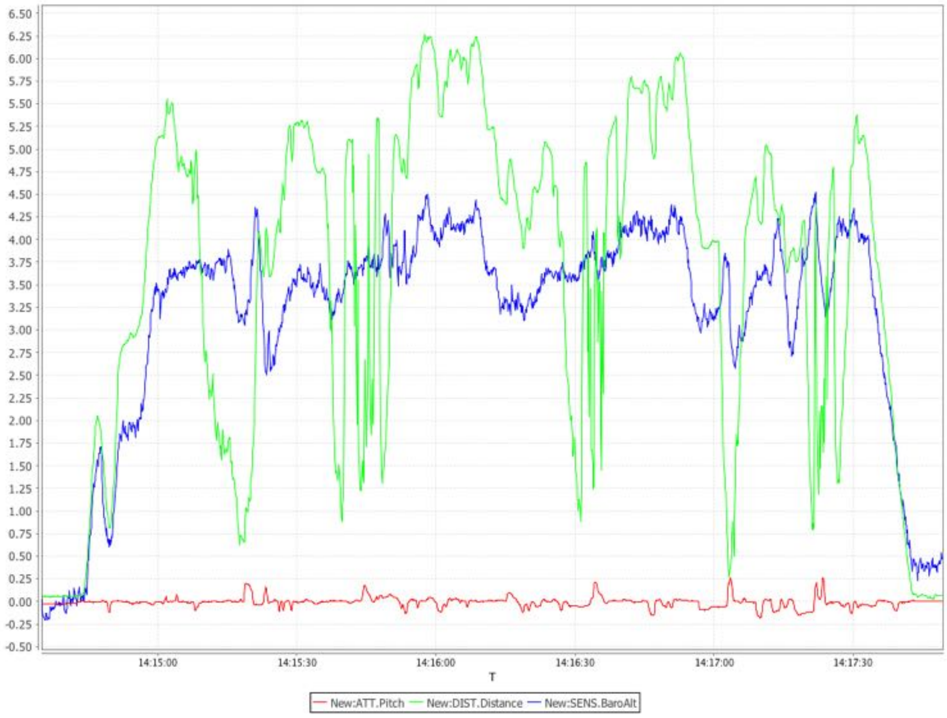


Figure 5-5 AGL System flight over vegetation

This is a good result, so I decided to go a step further, I've written a code on the autopilot linking the sensor distance with a change in the altitude set-point of the autopilot, to avoid odd behavior I made that this option could be disabled by the remote-control sticks.

In order to do this, I modified the pos_control module on the PX4 stack, by adding a new parameter: ALT_CTRL. I also modified the mavlink protocol to include the telemetry of the measured distance from ground.

The current distance measurement is sent directly to the position control module by the driver of the LidarLite, it is then compared with the altitude control parameter coming from the remote control in this way:

```
if(_range.current_distance < params.alt_ctrl && params.alt_ctrl > 0.5f)
{
        pos_sp(2) = _pos(2) – 1.0f;
}
```

pos_sp(2) is the set-point of the position on the z axis (x= 0, y=1, z=2). If the current distance is less than the distance set on the params.alt_ctrl and the params.alt_ctrl is greater than 0.5 meters (to avoid the drone to crash on the obstacle) then the position set-point is set to the actual position on the z axis _pos(2) minus 1 m.

Also, here the results were promising, with the drone increase of altitude whenever it went near the obstacle, the results are shown in Figure 5-6.

The red line shows when the AGL control is activated, it starts in a disabled state.

The Violet and the Cyan lines indicate respectively the Altitude set-point of the autopilot and the Vertical speed set-point of the autopilot.

The Green line is the distance measured by the AGL sensor, and the Blue line is the barometer altitude.

Starting from ground level and with the control disabled the RPAS takes-off, at 16:59:40 the control is enabled and the red line goes up. After the AGL sensor detects a low distance from ground the autopilot reacts by asking the motor to increase speed setting the set-point of the Altitude to a higher level (in the graph the altitude set-point is negative, so for a better understanding it must be interpreted as an absolute value). Also, the Vertical velocity set-point is changed and it controls how fast the RPAS should go up or down to avoid the obstacle.

The increase of altitude is promptly done by the RPAS as it can be seen in the blue line showing the barometer altitude. When the obstacle is passed the RPAS descends. I repeated the same task several times always getting roughly the same behavior as it can be seen by the successive passes.

The experiment gave good results, the behavior of the RPAS is a little crisp when it reacts to obstacles, but that can be smoothed by acting on the vertical velocity parameters.

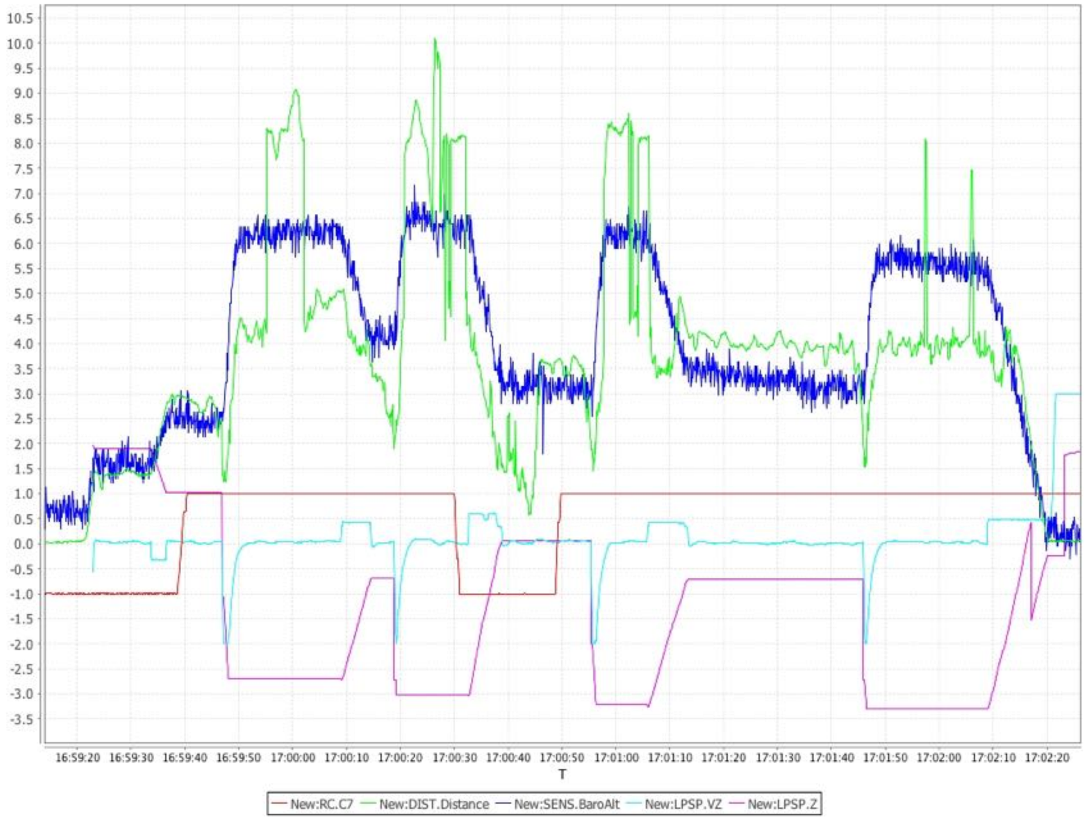


Figure 5-6 AGL altitude control experiment

6 MULTISENSOR SYSTEM

When I started the development of Leonardo the goal was to integrate in a single platform a complete system that could do complex tasks such as Sense and Avoid.

In this phase, I focused my attention on the “Sense” part.

As seen in chapter 3.4 several technologies are good candidates for obstacle detection, however every technology does only part of the job, due to the different characteristics of every obstacle. In order to have a reliable detection it is thus important to blend the information coming from different type of sensors.

They have been chosen to be practical for use on board a light and low-cost UAV. The three technologies integrated in the multi-sensor system are characterized by being complementary w.r.t. nature of targets, to get the widest spectrum of detected obstacles:

- RADAR
- SONAR
- LiDAR

Every sensor, to be mounted on a RPAS, needs to be lightweight, small in physical dimensions and with low power consumption.

6.1 Sensors

In this section I will show the characteristics of the chosen devices that will constitute the multi-sensor system.

6.1.1 IVS 167 Radar

The chosen RADAR device is a IVS-167 with integrated antenna, from Innosent.

This device is mainly meant to be used in Industrial applications, for traffic monitoring or for water level measurement. This device operates in the K band and is capable of FMCW/FSK, so it can measure distance of stationary objects, it is stereo (dual channel) so it can also detect the direction of motion.

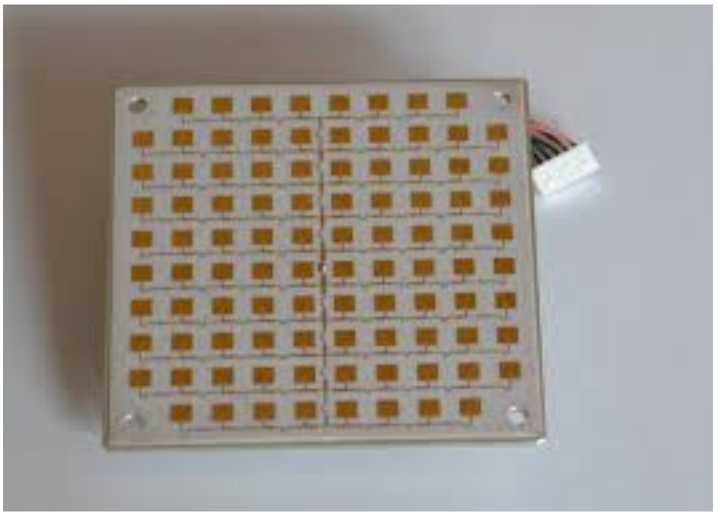


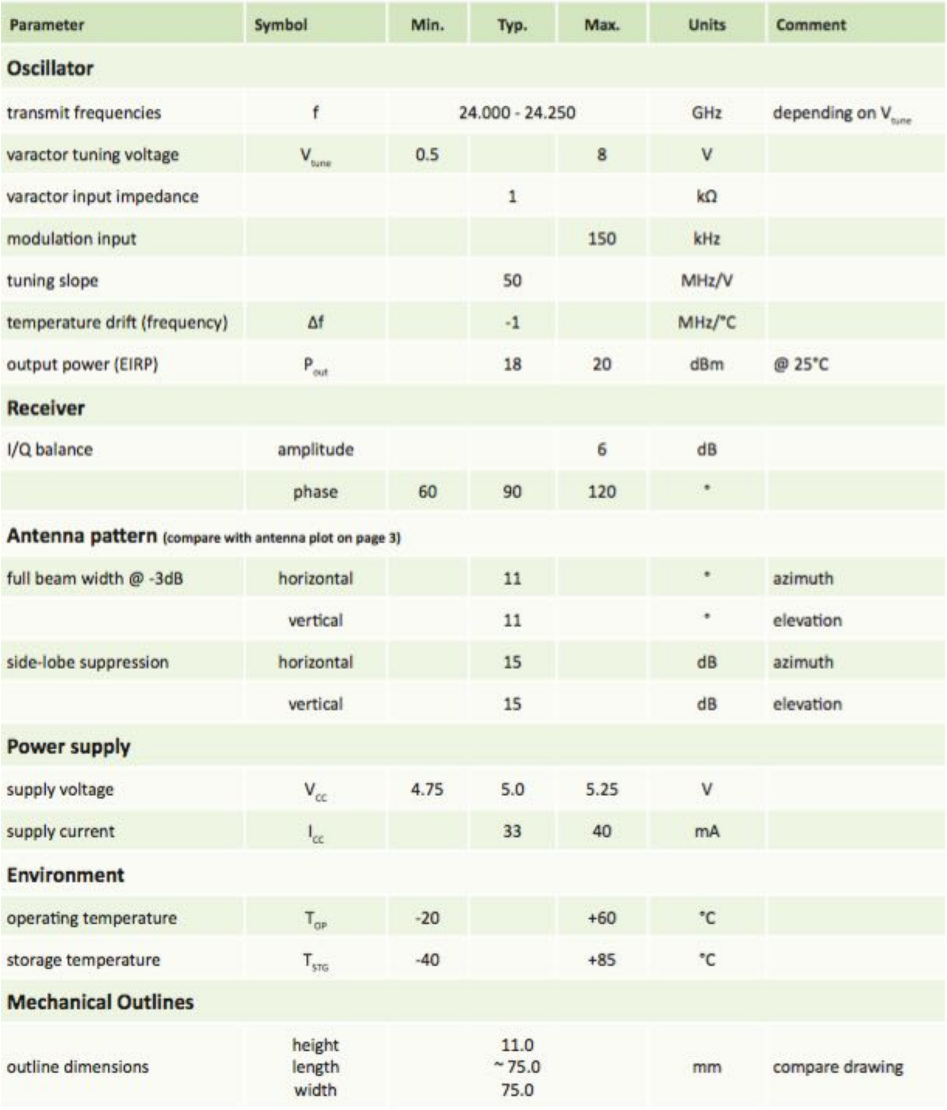
Figure 6-1 IVS 167 FMCW Radar

The IVS 167 is provided with a 5 pin connector with this pinout:

Pin	In/Out	Description	Comment
1	Out	IF1	Signal In Phase
2	Out	IF2	Signal Quadrature
3	In	GND	Analog Ground
4	In	VCC	Supply Voltage
5	In	Vtune	Varactor Tuning

Table 7 IVS 167 pinout

In Table 8 the RADAR main characteristics are shown.



Parameter	Symbol	Min.	Typ.	Max.	Units	Comment
Oscillator						
transmit frequencies	f	24.000 - 24.250			GHz	depending on V_{tune}
varactor tuning voltage	V_{tune}	0.5		8	V	
varactor input impedance			1		kΩ	
modulation input				150	kHz	
tuning slope			50		MHz/V	
temperature drift (frequency)	Δf		-1		MHz/°C	
output power (EIRP)	P_{out}		18	20	dBm	@ 25°C
Receiver						
I/Q balance	amplitude			6	dB	
	phase	60	90	120	°	
Antenna pattern (compare with antenna plot on page 3)						
full beam width @ -3dB	horizontal		11		°	azimuth
	vertical		11		°	elevation
side-lobe suppression	horizontal		15		dB	azimuth
	vertical		15		dB	elevation
Power supply						
supply voltage	V_{CC}	4.75	5.0	5.25	V	
supply current	I_{CC}		33	40	mA	
Environment						
operating temperature	T_{OP}	-20		+60	°C	
storage temperature	T_{STG}	-40		+85	°C	
Mechanical Outlines						
outline dimensions	height length width		11.0 ~75.0 75.0		mm	compare drawing

Table 8 IVS 167 Characteristics

The RADAR is powered with 5V and sinks about 33 mA of current, leading to a power consumption of roughly 165 mW.

To use it in FMCW operation it is necessary to drive the Vtune input of the VCO. No data (apart from the table shown above) was provided by Innosent on how the frequency varies with the Vtune voltage, however by using a spectrum analyzer I have seen that a 0 to 5 V voltage is enough to drive changes of the carrier frequency over the whole span of 250 MHz.

The full beam width in both horizontal and vertical direction is 11°, with a side lobe suppression of 15 dB like shown in Figure 6-2. This leads to a quite narrow "FOV" of the sensor.

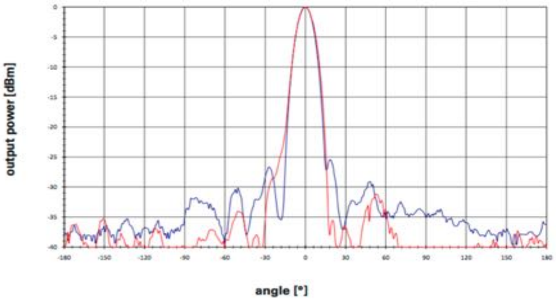


Figure 6-2. IVS-167 Antenna vertical (blue) and horizontal (red) pattern

6.1.2 LidarLite LiDAR

The chosen sensor is LidarLite (PulsedLight3D, 2015), a low-cost laser range finder characterized by low weight and good ranging capabilities up to 30-40m depending on the obstacle reflectivity.

Figure 6-3 LidarLite range-finder from PulsedLight

LidarLite is a device with an integrated FPGA on-board, that uses a patented algorithm to estimate the correct distance of a target.

This Laser Product is designated Class 1 during all procedures of operation.

Parameters	**Laser Value**
Wavelength	905nm (nominal)
Total Laser Power - Peak	1.3Watts
Mode of operation	Pulsed (max pulse train 256 pulses)
Pulse Width	0.5µSec (50% duty Cycle)
Pulse Repetition Frequency	10-20KHz nominal
Energy per Pulse	<280nJ
Beam Diameter at laser aperture	12mm x 2mm
Divergence	4mRadian x 2mRadian (Approx)

Table 9 LidarLite Laser Specifications

It is equipped with two kinds of interface:

- PWM
- I^2C

For my experimental system, I chose to interface the LiDAR through the I^2C interface.

In Table 10 the LidarLite pinout is shown.

Pin	Description
1	POWER_IN – 4.75-5.5V DC Nominal, Maximum 6V DC. Peak current draw from this input (which occurs during acquisition period) is typically < 100 mA over a duration from 4 to 20ms depending on received signal strength. Unless you use power management, the unit will draw 80 mA between acquisition times.
2	POWER_EN - Active high, enables operation of the 3.3V micro-controller regulator. Low puts board to sleep, draws <40 µA. (Internal 100K pull-up)
3	Mode Select – Provides trigger (high-low edge) PWM out (high)
4	I2C Data (SDA)
5	I2C Clock (SCL)
6	Signal/power ground.

Table 10 LidarLite Pinout

6.1.3 Maxbotix SONAR

The SONAR used in this experiment is the MB1242 shown in Figure 3-3. It is designed for robot/ UAV applications and can detect obstacles up to 7 meters far, depending on the material and shape of the obstacle. It is provided with an I^2C interface that allows it to be easily interfaced with a microcontroller.

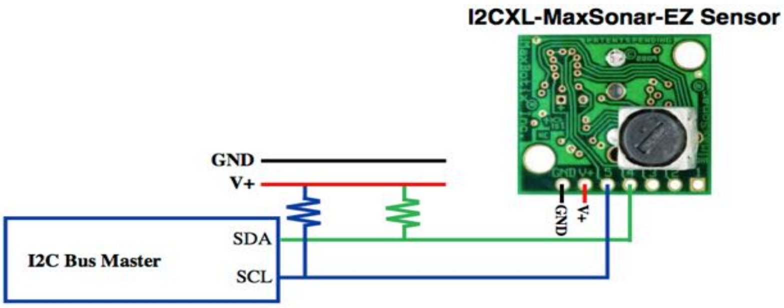


Figure 6-4 MB1242 connection

Sensor reading rate changes depending on the distance from the obstacle, for close obstacles the rate is up to 40 Hz, for far obstacles the rate must be slowed down to 15 Hz. The ultrasonic signal frequency is 42 KHz. When supplied at 5 V its average current consumption is of 4.4 mA, with peaks of 100 mA.

6.2 Rule Sieve

Before putting raw data into a Kalman filter, the information must first be manipulated and suitably “sieved”. A rule filter has been implemented which does a coarse selection of the data to be passed to the Kalman filter. Rules applied to data are the following:

1. Maximum and minimum Range constraints
2. Measurements sanity check

6.2.1 Maximum and minimum Range constraints:

Every sensor has its own operating range, therefore data coming e.g. from the ultrasound sensor near its full-scale range get dropped by the filter, in order to avoid invalid (saturated) readings that would impair the estimation. In Table 11 the chosen range constraints are shown.

	Min Range (m)	Max Range (m)
RADAR	2	40
LiDAR	0.1	30
SONAR	0.1	7

Table 11 Range constraints

6.2.2 Measurements sanity check:

Even if a measurement appears to be correct, it can be distorted by the limits of the technology of a sensor. For instance, LiDAR sensor measurements are usually very precise if compared to the ones of FMCW RADAR and SONAR, however they can be a lot less accurate than the other two because LiDAR sensors have very tight FOV, so they can miss obstacles that have narrow form factors like nets or wires.

The rule filter compares the measurements coming from the three sensors and if the other two sensors have taken a valid measurement and measure a distance shorter than that one of the LiDAR, the latter gets dropped. Dropping a measurement means to invalidate it by passing a 0 value into the H matrix of the Kalman filter for that measurement.

6.3 Data Fusion – Kalman Filter

Kalman filters are often used in data fusion because of their ease of implementation, and optimality in a mean-squared error sense (Khalegi, 2013). To implement the Kalman filter we have used the well-known equations for the two-step discrete Kalman filter cycle (Welch, 2006)

Time update (prediction)

$$\hat{x}_k^- = A\hat{x}_{k-1}^- + Bu_{k-1}^-$$

$$P_k^- = AP_{k-1}^- A^T + Q$$

Measurement update (correction)

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$

$$\hat{x}_k \ = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

$$P_k \ = (I - K_k H)P_k^-$$

Here the desired physical quantity in output from the filter is the distance of a target. Distance has been chosen as the state of the filter ($\hat{x}_k$ is the distance estimation) and the three measurements (z_k), coming from sensors, are fed into the filter in the measurement update phase, where the *a posteriori* state estimate is computed. K_k is the Kalman filter gain. The remaining Kalman filter parameters are defined as follows:

$$A = 1 \qquad R = \begin{pmatrix} r_R & 0 & 0 \\ 0 & r_L & 0 \\ 0 & 0 & r_{RS} \end{pmatrix}$$

$$B = 0$$

$$Q = 10^{-5} \qquad H = \begin{pmatrix} h_{Rk} \\ h_{Lk} \\ h_{Sk} \end{pmatrix}$$

- A is the state transition matrix, because it represents the evolution of the single state of the system (the distance).

- B, the control matrix is zero, because we assume of not having any external input (u_k) to the system.

- H matrix indicating the observation model, is a 3×1 matrix containing the three values (h_{Rk}, h_{Lk}, h_{Sk}) where the R, L and S subscripts refer to RADAR, LiDAR and SONAR respectively. These values can be ones or zeros depending on whether the measure of each of the three sensors has passed or not the previous rule filter.

- R is the measurement noise covariance and is populated with the noise of the three sensors (*r_R=0.36, r_L=0.0025, r_S=0.01*) as it is obtained from technical specs of each sensor.

- Q is the process noise covariance, its value is not so obvious and often must be guessed as a first assumption and then tuned by experimentation. Here I have chosen a low value of Q assuming a low process noise.

The last parameter to define is P_0, the initial estimate error covariance. It is not critical, it just needs to be different from zero, so we set $P_0 = 1$.

6.4 First experimental setup

The first experimental setup was realized by interfacing the three sensors with a PC running Matlab.

6.4.1 LiDAR and SONAR interfaces

For what regards LiDAR and SONAR the choice has been to acquire the data through Arduino, and then reading it into Matlab through the USB Virtual COM port of the PC. The actual connection is shown in Figure 6-5.

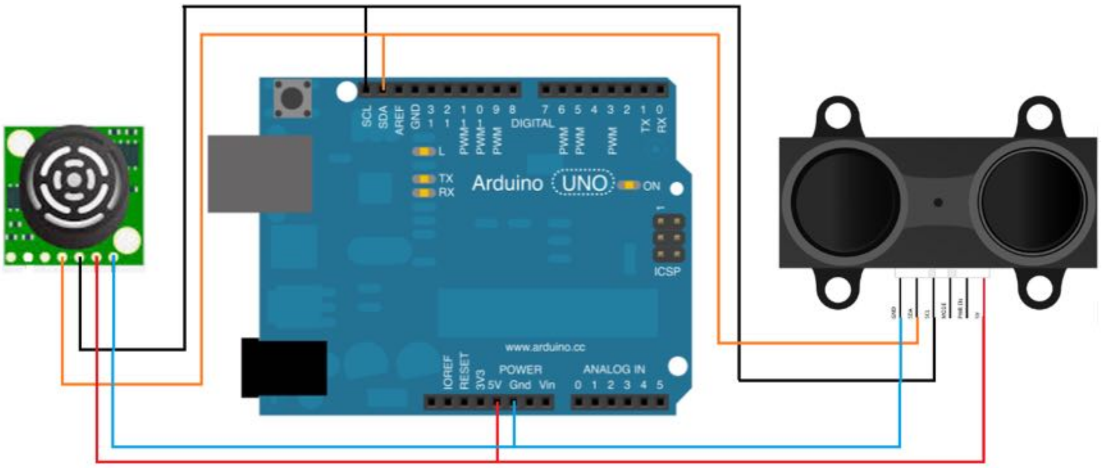


Figure 6-5 LiDAR and SONAR connection to Arduino

The code for reading these two sensors on Arduino is quite simple:

In the declaration part are defined the address of registers that will be used:

```
#include <I2C.h>

#define      UltraSound_ADDRESS      0x70
#define      LIDARLite_ADDRESS       0x62
#define      RegisterMeasure         0x00
#define      MeasureValue            0x04
#define      RegisterHighLowB        0x8f
```

In the loop phase the software continuously prompts each sensor for an updated measure.

In the snippet below the code is shown for the ultrasound sensor, the LiDAR one is much similar:

```
nackack = 100;

while (nackack != 0)

{

      nackack = I2c.write(UltraSound_ADDRESS,0x00,0x51);

      delay(1);

}

delay(100);
```

```
nackack = 100

while (nackack != 0)

{

nackack = I2c.read(UltraSound_ADDRESS,0x02,2,distanceArray_U);

delay(1);

}


  distance_u = (distanceArray_U[0] << 8) + distanceArray_U[1];

  Serial.println(distance_u);
```

On Matlab side it is necessary to open a serial interface, and wait for data to come:

```
delete(instrfindall);
s = serial('serialportname');
fopen(s);
flushinput(s);
fgetl(s);
reading = fgetl(s);
misure = textscan(reading,'%d,%d');
laserdist = double(misure{1})/100;
ultradist = double(misure{2})/100;
fclose(s);
```

6.4.2 RADAR interface

For the RADAR the interface has been somewhat more complicated since the signal coming out from the device is analog. Therefore, it was necessary to develop a signal conditioning circuit and then to convert the signal to digital through an ADC.

The analog part has been made with a prototyping board and wire-wrapped with a soldering iron.

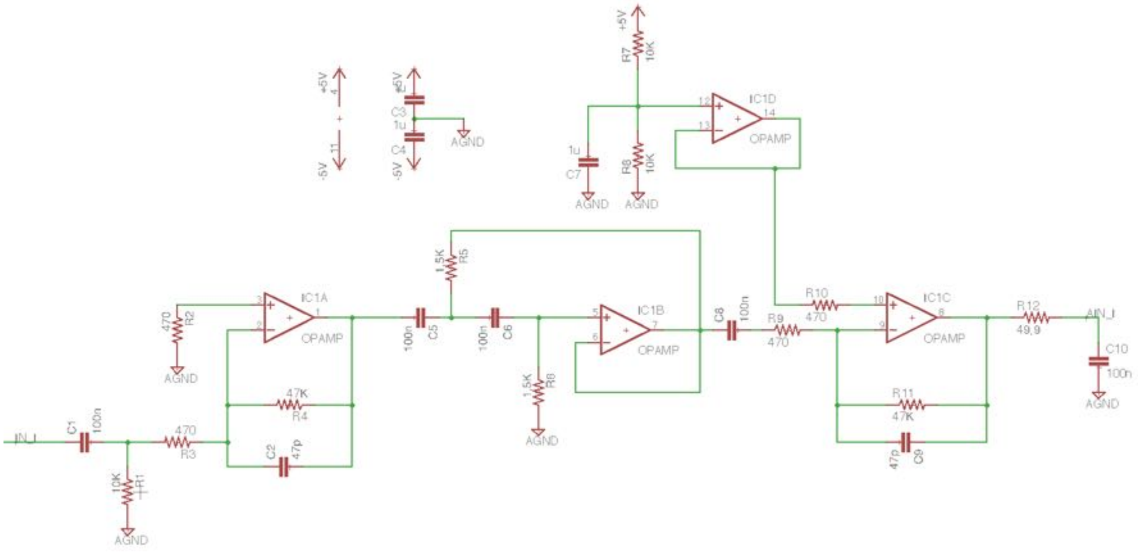


Figure 6-6 Shematic of Analog conditioning circuit

I have taken RADAR output at the IF pin in base-band frequencies. The output signal at this pin is in the order of few mV, and furthermore it presents spurious frequencies that feed into the output signal from the input modulating signal. To this purpose, I designed a filter-amplifier chain like shown in Figure 6-6.

In the first stage, there is a passive high-pass RC network made by R1 and C1 with a high-pass frequency of roughly 159 Hz, this is to block the high swing signal leaking from the transmitter in the IF output.

The first op-amp, with its passive network R2, R3, R4, C2 constitute a gain block of 100 with a low-pass filter at approximately 72 KHz. The second op-amp is tied in a Sallen-Key high-pass filter configuration with high-pass frequency calculated to be around:

$$f_{cut} = \frac{1}{2\pi\sqrt{R5\;R6\;C5\;C6}} \approx 1061\;Hz$$

This filter stage is designed to cut out the 100 Hz noise coming from the transmitter and to attenuate the low frequencies in which the RADAR would have a too high dynamic to be correctly read.

Next, after a bypass capacitor C8, there is another amplification stage of 100 with a low pass filter constituted by: R9, R10, C9, R11.

The fourth op-amp (IC1D) is used to fix the common-mode at the center of the dynamic of the ADC.

Lastly there is a passive low-pass filter R12, C10, used as an anti-aliasing filter for the ADC.

The difficulties that I encountered here were in limiting the noise of the circuit to get a good signal-to-noise ratio at the output, so we chose a low noise op-amp (MAX44252) and operated in dual supply to minimize the common-mode noise. In Figure 6-7 the response of the filter (simulated in PSpice) in magnitude and phase is shown.

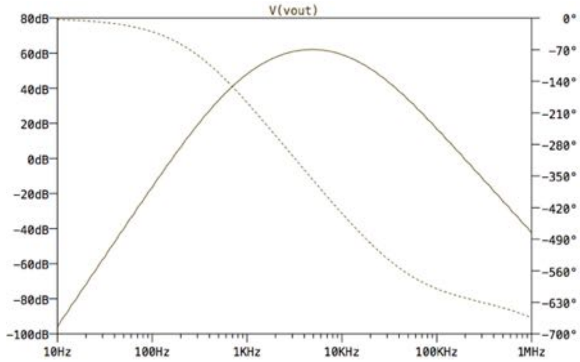


Figure 6-7. Response of the FMCW Radar Filter – magnitude (solid), phase (dotted)

As it can be seen, it behaves like a band-pass filter starting from 2 kHz up to 20 kHz.

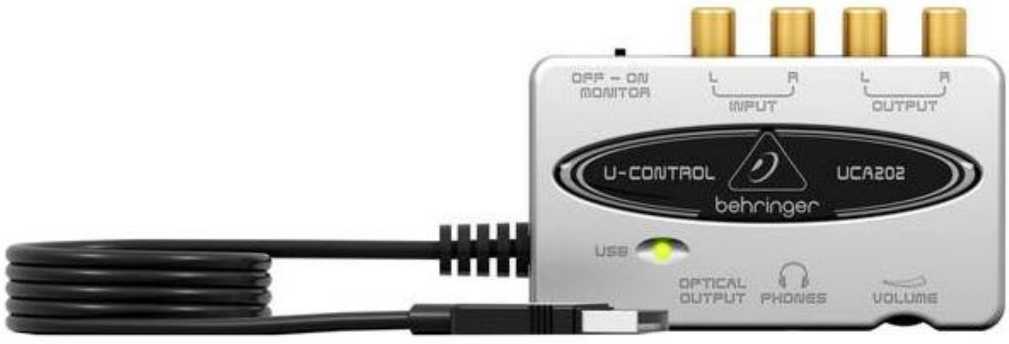


Figure 6-8 Behringer UCA202 Audio A/D interface

A UCA202 from Behringer has been used as an analog to digital converter interface, it is an audio device with good characteristics that met our needs. Its specifications are shown in Table 12:

A/D Resolution	16 bit
Sample Rate	32.0 kHz, 44.1kHz, 48.0 kHz
THD	0.05% typ. @-10dBV, 1kHz
Crosstalk	-77dB @0 dBV, 1kHz
Signal-to-noise ratio	A/D 89 dB typ. @ 1kHz, A-weighted D/A 96 dB typ. @ 1kHz, A-weighted
Frequency response	10Hz to 20kHz, +/- 1dB @44.1 kHz sample rate 10 Hz to 22kHz,+/- 1dB @48.0 kHz sample rate

Table 12 UCA-202 specifications

This device has a USB interface and gets detected by the PC as a sound card.

The actual wiring between RADAR and PC is shown in Figure 6-9

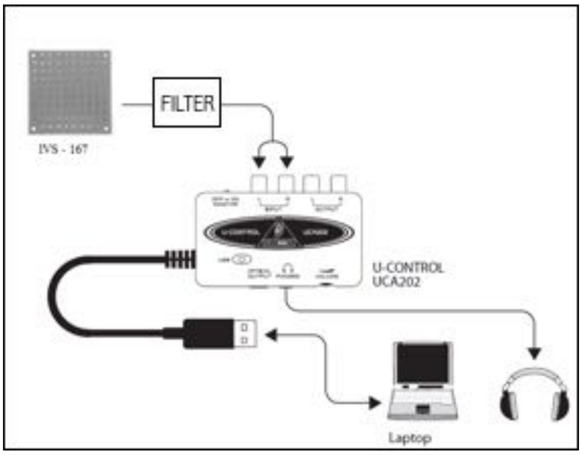


Figure 6-9 RADAR interface wiring diagram

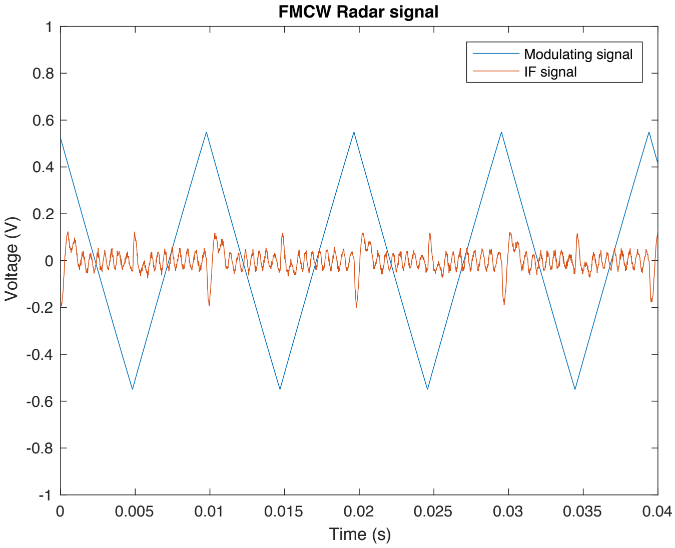


Figure 6-10. FMCW Radar Waveform

The converted signal looks like in Figure 6-10 where the modulating signal applied to the RADAR is shown in blue, while the output amplified and filtered IF signal is shown in red.

The frequency of the modulating signal has been chosen such that the relationship between the detected distance and the relative peak frequency is well inside the ADC sampling frequency capabilities.

As discussed before, the distance and velocity information is contained in the IF signal as frequencies following these equations.

$$Distance = \frac{(fup\ + fdown) \cdot c \cdot T\_triang}{4\Delta F},$$

$$Velocity = \frac{|fup - fdown| \cdot c \cdot T\_triang}{4\Delta F},$$

Where f_{UP} and f_{DOWN} are the peak frequencies contained in the IF signal corresponding to the UP/DOWN ramp of the transmitted signal.

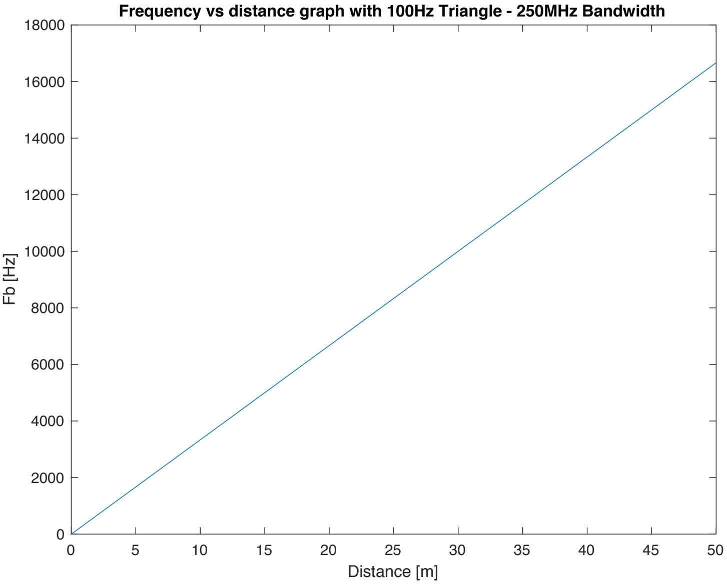


Figure 6-11 Frequency vs Distance graph at 100Hz

Figure 6-11 shows how target distance relates to the peak frequency in the IF signal. In our case for 100 Hz modulating signal and 48 kHz ADC sampling rate we should be reading good signals up to 40 m.

So, once the signal from RADAR arrives to Matlab it is necessary to elaborate it to extract the frequency components in order to find the peaks corresponding to a possible target.

This elaboration is made by means of a Discrete Fourier Transform.

$$X_q = \mathcal{F}_d(x_n) = \sum_{k=0}^{N-1} x_k e^{-i\frac{2\pi}{N}kq} \qquad q = 0, 1, \ldots, N-1$$

In Matlab the signal is elaborated using the FFT function.

To avoid noise and non-idealities FFT must be done on the portion of signal that contains most of the information, this signal is just a chunk of the IF signal as shown in Figure 6-12:

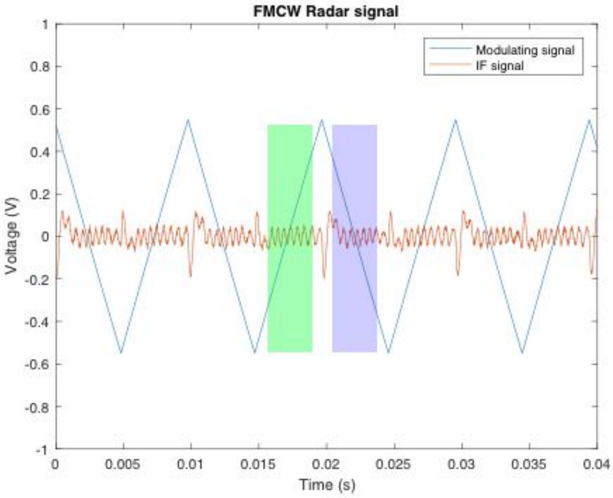


Figure 6-12 Up and Down chunk selection for FFT

The green part is the UP chunk and the violet part is the DOWN chunk.

Since doing FFTs on chunks of signal is the same as multiplying the signal with a rectangular window, the effect in the FFT is the leak of energy out from the mainlobe to the sidelobes. In Figure 6-13 a good graphical representation is reported, taken from (Cheung, 2011), that helps to understand the problem.

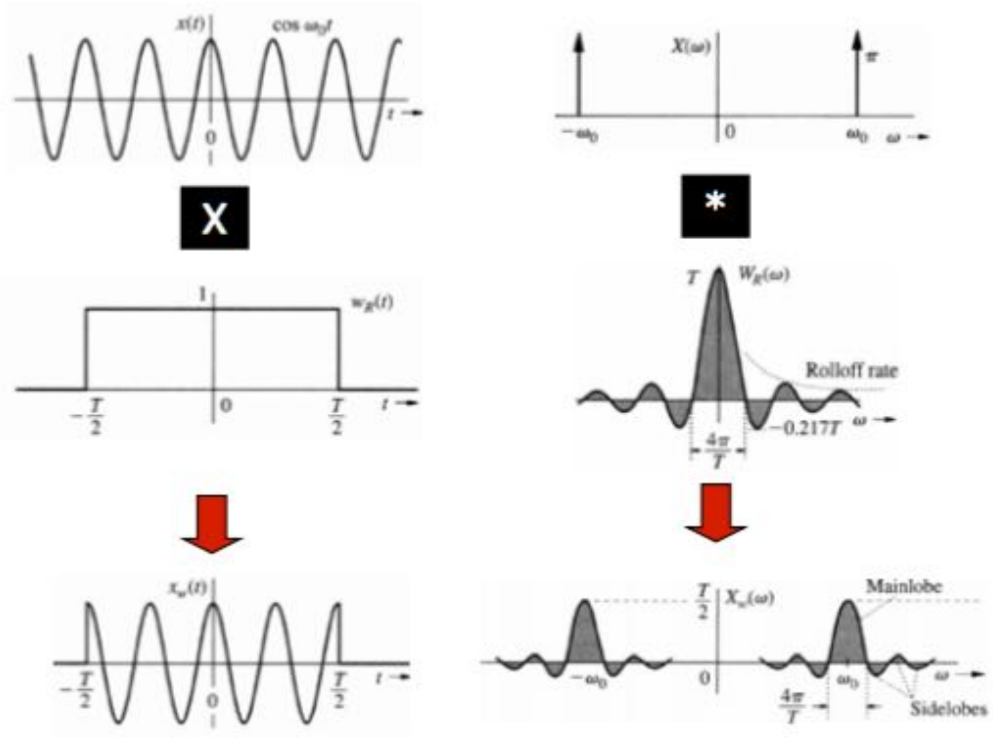


Figure 6-13 Windowing effect

While a sinusoid transforms to a Dirac impulse in the frequency domain, a rect transforms into a sinc, the results of the multiplication in time domain produces the distorted result in frequency domain.

The effects are even more clear by looking at Figure 6-14, in which the mainlobe to sidelobes ratio is evident.

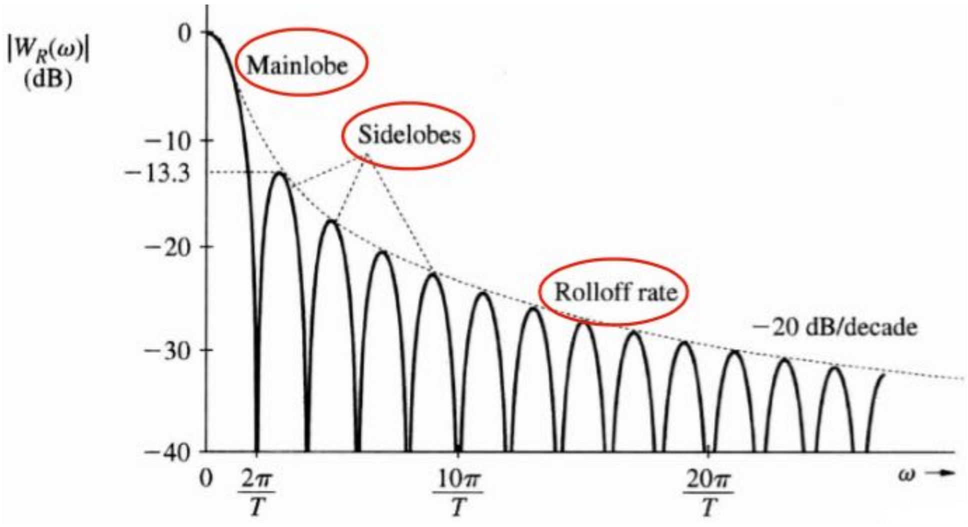


Figure 6-14 Details of Windowing effect

The remedy to such a problem is to avoid discontinuity in the windowing function, this will reduce leakage and consequently the side lobes. The effect of windowing in time is shown in Figure 6-15:

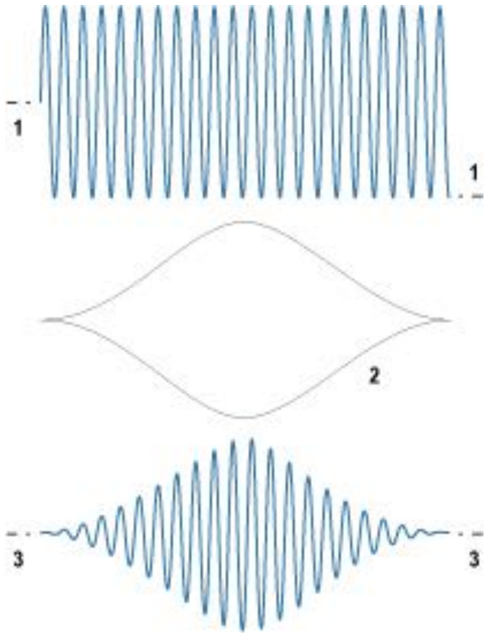


Figure 6-15 Applying windowing function to signal

The effect in frequency is shown in Figure 6-16, where different windowing functions are compared each other. In my application, at first, I used a Hanning window:

$$H(t) = \frac{1}{2}\left[1 + \cos\left(\frac{2\pi t}{T}\right)\right]$$

which is characterized by very low aliasing. Further experimentation can be done to find which windowing function fits best this application.

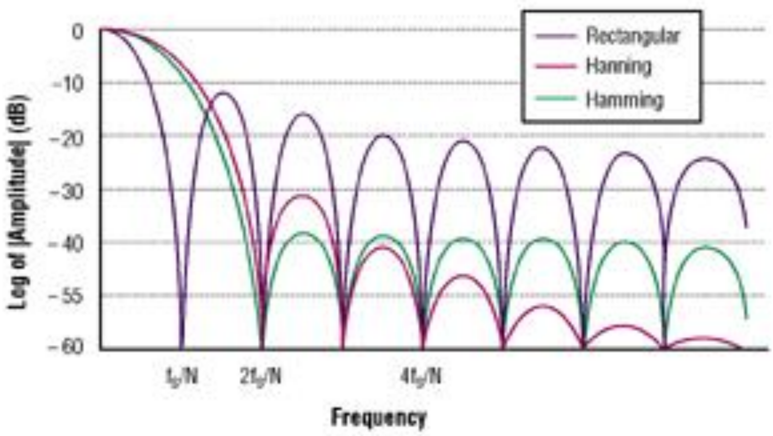


Figure 6-16 Effects of windowing in frequency domain

The snippet of code below reports the Matlab loop that analyzes every sample coming from the ADC and selects for every window the highest peak.

```
while ( (k+winSize-1) <= n_samples )

    FrameSignal_up = rec_up(k:k+winSize-1);
    FrameSignal_down = rec_down(k:k+winSize-1);

    %%%%%%%%%% Start process FFT %%%%%%%%%%%%%%%%
    w = hanning(winSize);

    wdata_up = [FrameSignal_up(:).*w; zeros(NFFT-SAM,1)];
    wdata_down = [FrameSignal_down(:).*w;zeros(NFFT-SAM,1)];

    P_up = fft(wdata_up)/SAM;
    P_down = fft(wdata_down)/SAM;

    M_up = abs(P_up(1:NFFT/2+1)).^2;
    M_down =abs(P_down(1:NFFT/2+1)).^2;

    M_up = (M_up + M_old_up)/2;
    M_down = (M_down + M_old_down)/2;
    M_old_up = M_up;
    M_old_down = M_down;


    f = Fs/2*linspace(0,1,NFFT/2+1);

    [M_max_up(i), id_up] = max(M_up(1+jump:end));

    if(M_max_up(i)>1e-7)
        f_pulse_up(i) = f(id_up + jump);
        [M_max_down(i), id_down] = max(M_down(1+jump:end));
            f_pulse_down(i) = f(id_down + jump);
    else
        f_pulse_up(i)=Fs/2;
        f_pulse_down(i)=Fs/2;
    end
    Marks(i,1)=M_max_up(i);
    Marks(i,2)=f_pulse_up(i);
    a= 0.5;
    f_pulse_up_filt(i+1) = a*f_pulse_up_filt(i) + (1-a)*f_pulse_up(i);
    f_pulse_down_filt(i+1) = a*f_pulse_down_filt(i) + (1-
a)*f_pulse_down(i);
    t(i+1) = t(i)+ 2*5e-3;
    i = i + 1;

   k = k + SAM;
end
```

As it can be seen in the code, a zero-padding technique has been used. This is because the FFT has been done with a power-of-two window, and the samples contained in a single slope are too many for 128 samples and too few for 256 samples window.

The constraint of using a power-of-two window in FFT comes from the fact that the final implementation will be made on hardware, and in hardware the FFT implementation is constrained to be radix-2 or radix-4 perhaps.

The actual RADAR distance calculation is done with these commands:

```
c = 3e8;
T_triang = (10e-3)./2;    %semiperiod
delta_f = 250e6;
dist = (f_pulse_up_filt + f_pulse_down_filt)*c*T_triang/(4*delta_f);
```

```
dist_unf = (f_pulse_up  + f_pulse_down)*c*T_triang/(4*delta_f);
speed = (abs(f_pulse_up_filt -
f_pulse_down_filt))*c*T_triang/(4*delta_f);
```

6.4.3 *Rule sieve and Kalman filter*

After getting the distance on every sensor, it is now time to pass the data through the rule sieve that will clean the samples as defined in section 6.2.

The Matlab code below shows the rule sieve implementation, `Readings` is the vector containing the distance samples for every sensor, the `z` matrix will contain the measurements in input to the Kalman filter, while the `H` matrix is the observation matrix and its values are put to zero whenever the measurement doesn't pass the rule sieve filter.

```
%1 RADAR
%2 LIDAR
%3 SONAR

for i=1:N
    if(Readings(i,3) < 7)
        z(3,i)=Readings(i,3);
        H(3,i)=1;
    else
        z(3,i)=0;
        H(3,i)=0;
    end
    if Readings(i,2)==0
        z(2,i)=0;
        H(2,i)=0;
    else
        z(2,i)=Readings(i,2);
        H(2,i)=1;
    end
    if Readings(i,1) < 2
        z(1,i)=0;
        H(1,i)=0;
    else
        z(1,i)=Readings(i,1);
        H(1,i)=1;
    end
    if(z(2,i)>z(1,i)+2 && H(1,i)==1)
        H(2,i)=0;
    end
    if(z(2,i)>z(3,i)+2 && H(3,i)==1)
        H(2,i)=0;
    end
    if(z(1,i)>z(3,1)+2 && H(3,i)==1)
        H(1,i)=0;
    end

end
```

After passing the rule sieve filter, data is passed to the Kalman filter:

```
Q=1e-5;
R=[0.36 0 0;0 0.0025 0;0 0 0.01];
x=zeros(1,N);
p=zeros(1,N);
p(1)=1;
A=1;
B=0;
for i=1:N
[x(i+1),p(i+1),K(i,:)]=kalmanC(z(:,i),x(i),p(i),0,A,B,H(:,i),Q,R);
end
```

the kalmanC function is my Matlab implementation of the Kalman filter. Even if Matlab offered a Kalman filter implementation, I decided to do it on my own because such approach helped in the next phase when I had to write the Kalman filter for the DSP:

```
function [x1,P1,K]=kalmanC(z,x0,P0,u0,A,B,H,Q,R)
%A state evolution matrix
%B input matrix
%H observation matrix
%Q process covariance matrix
%R measurements covariance matrix
x = A * x0 + B * u0;
P = A * P0 * A' + Q;
y = z - H * x;
S = H * P * H' + R;
K = P * H' * inv(S);
x1 = x + K * y;
P1 = (eye(size(P)) - K*H)*P;
end
```

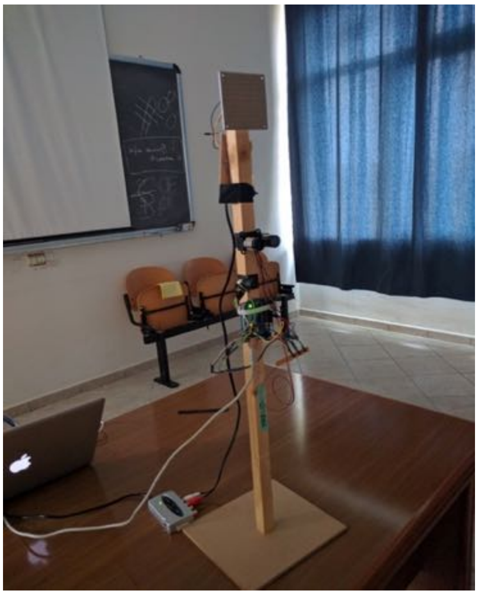


Figure 6-17. First Experimental setup

6.5 Second experimental setup (DSP)

The first experimental setup has proven good, giving promising results, anyway it is far from being a viable solution because it is not integrated. So, the next step has been to design an embedded system for this integration. Truly the Leonardo autopilot could have been used, but unfortunately it lacks the analog circuitry necessary to the purpose, so I decided to build an integrated interface that could also be connected to Leonardo but that also has a DSP on board (of the same class of Leonardo one) in order, in the future, to compare DSP performance with FPGA performance.

The system architecture of the designed system is reported in the block diagram below (Figure 6-18)

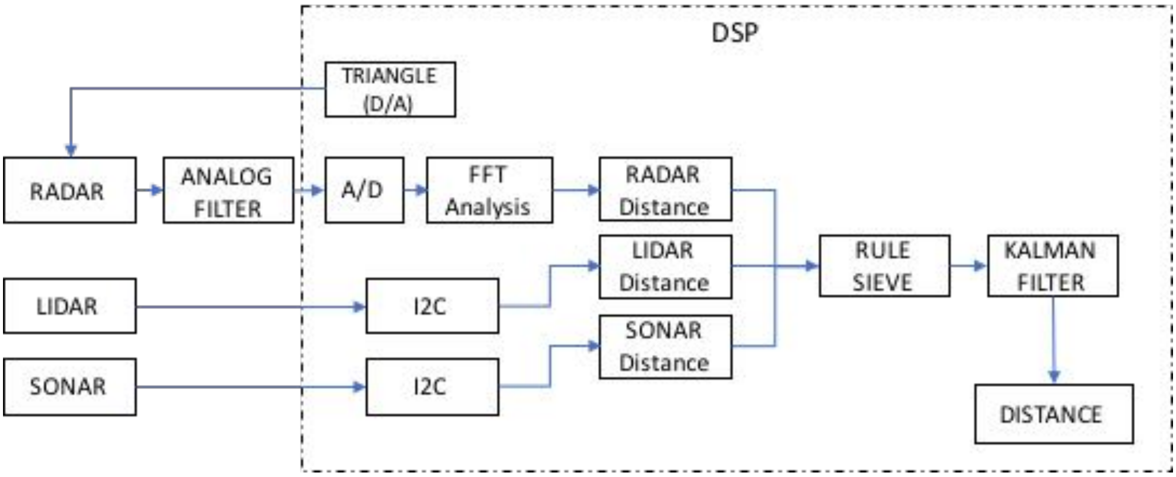


Figure 6-18 System Architecture

In this configuration, the DSP does all the job except the analog filtering. The schematic of the digital section of the system is reported in appendix at 10.2.1.

As it can be seen from the schematic, the DSP interfaces directly with LidarLite and Maxbotix SONAR through the I^{2}C interface, the system has an UART serial interface to transmit distance and debug data.

The analog-to-digital conversion circuitry is redundant, I've implemented several possibilities, the system can alternatively use:

- 12 bit internal ADC of the STM32
- 14 bit external ADC LTC2312-14
- 24 bit external ADC PCM1808

I've provided those three choices for several reasons, the first solution to use the internal ADC is the most effective in costs and easiness of use together with a still good resolution of 12 bit.

The 14 bit external ADC is a bit more powerful than the first choice and would be a good choice to be used with an FPGA for its easy serial hardware interface.

The last ADC is a 24 bit audio converter and I've provided it for testing should the other two configurations not be good enough in results.

The core of the multi-sensor system is a DSP from ST Microelectronics, STM32F446RC. This unit offers a high performance-to-cost ratio. In Table 13 some key data from its datasheet are reported (ST, 2015):

STM32F446RC	
Architecture	ARM Cortex M4 32 bit +
Performance	180 MHz - 225 DMIPS
Flash	256 kB
RAM	128 kB

Table 13. STM32F446RC Specifications

Figure 6-20 shows the system integrated on a PCB.

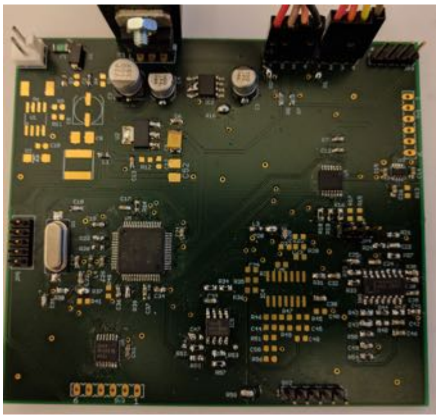


Figure 6-19 DSP Multi-sensor Board

The PCB shown here has got every part of the system that is needed for the multi-sensor to work, moreover it has additional circuitry that will be useful to interface the system to Leonardo.

6.5.1 Firmware

The firmware for this system was written in C using the OpenSTM32 interface from ST, that is available free of charge being a platform based on Eclipse and GCC.

The backbone of the firmware is generated by STM32 Cube MX: it is a configurator in which the user can choose the interfaces of the DSP that will be used in the project and can do basic configurations like clock frequency, ADC conversion frequencies, which interrupt must be enabled.

Let's analyze now the C source code for the multi-sensor system, it can be divided in several main sections:

- Radar driver
- Matrix library
- Sonar and LiDAR driver
- Kalman filter
- FFT

6.5.1.1 Sonar and LiDAR driver

Sonar and LiDAR are operated through the I^{2}C interface. Due to some limitation in LiDAR firmware it can happen sometimes that the LiDAR hangs and does not respond to I^{2}C commands anymore. This issue is documented on PulsedLight datasheet and has been resolved in a new version of the LiDAR. To overcome this issue, I've written a state-machine that regularly checks if the LiDAR is still up and responding, otherwise it will be restarted through the I^{2}C interface.

The state machine is made of three states: IDLE, BUSY, WAIT_DATA. The states of the machine are scanned at the RADAR triangle rate which is 100 Hz. A timeout variable accounts to end the current request and start a new request if necessary.

```
if(Lidar_State == IDLE)
{
      Lidar_State = BUSY;
      I2C_WriteBuf = 0x04;
      HAL_I2C_Mem_Write_IT(&hi2c1, LIDAR_ADDRESS, 0, 1, &I2C_WriteBuf,
1);
}
else if(Lidar_State == BUSY)
{
      if(Lidar_Cnt++ == 40)
      {
            HAL_I2C_Mem_Read_IT(&hi2c1, LIDAR_ADDRESS, 0x8F, 1,
&I2C_ReadBuf[2], 2);
            Lidar_State = WAIT_DATA;
            Lidar_Cnt = 0;
            Lidar_Timeout = 0;
      }
}
else if(Lidar_State == WAIT_DATA)
{
      Lidar_Timeout++;
      if(Lidar_Timeout > 80)
      {
            Lidar_Timeout = 0;
            HAL_I2C_DeInit(&hi2c1);
            HAL_I2C_Init(&hi2c1);
            Lidar_State = IDLE;
      }
}
```

As it can be seen by the code the I^{2}C request is interrupt based, so when there is activity on I^{2}C a callback function is called:

```
  if(hi2c->Devaddress == SONAR_ADDRESS)
  {
        Sonar_State = IDLE;
        Sonar_Range = I2C_ReadBuf[1] + 256 * I2C_ReadBuf[0];
        Sonar_Timeout = 0;
  }

  if(hi2c->Devaddress == LIDAR_ADDRESS)
  {
        Lidar_State = IDLE;
```

```
            Lidar_Range = I2C_ReadBuf[3] + 256 * I2C_ReadBuf[2];
            Lidar_Timeout = 0;
  }
```

6.5.1.2 RADAR driver

The RADAR management is more complicated if compared with the other two technologies because the sensor is not provided with a digital interface.

The DAC converter inside the DSP is used to drive the Vtune pin (together with an analog amplifier/filter) and it is configured to generate a triangle wave and to synchronize it to Timer 6, this timer is also used to synchronize all the other events of the system:

```
    /**DAC channel OUT1 config
    */
  sConfig.DAC_Trigger = DAC_TRIGGER_T6_TRGO;
  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
  if (HAL_DAC_ConfigChannel(&hdac, &sConfig, DAC_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Triangle wave generation on DAC OUT1
    */
  if (HAL_DACEx_TriangleWaveGenerate(&hdac, DAC_CHANNEL_1,
DAC_TRIANGLEAMPLITUDE_4095) != HAL_OK)
  {
    Error_Handler();
  }
```

The RADAR state machine is made of six states: RADAR_IDLE, ACQ_UP, WAIT_DOWN, ACQ_DOWN, WAIT_ELAB, ELAB.

When the SM is in its RADAR_IDLE state and a front of the triangle is detected the ADC is started by the following command

```
if(Rising_front == 1)
{
if(Radar_State == RADAR_IDLE)
{
```

```
            Radar_State = ACQ_UP;
            HAL_ADC_Start_DMA(&hadc1, ADCBuffer, 2*ADC_BUFFER_LEN);
}
…
```

The ADC is used with DMA. DMA stands for Direct Memory Access and is a feature that allows ADC to acquire samples and storing them directly in memory without interrupting the normal operation of the DSP, then when the acquisition is ended, an interrupt is issued and the DSP can elaborate ADC data. In this case, this is done with the code shown below:

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef * AdcHandle)
{
      if(Radar_State == ACQ_UP)
      {
            Radar_State = WAIT_DOWN;
      }
      else if(Radar_State == ACQ_DOWN)
      {
            Radar_State = WAIT_ELAB;
      }
}
```

When the RADAR enters in the ELAB state the FFT is done on the acquired ADC data

```
FFTRealPtr = (int16_t *)ADCBuffer;
//FFT calculation
   myfft(FFTRealPtr, FFTImg, 8);
```

the function myfft does the job, it implements a Fixed-Point Fast FFT at 16 bit, the algorithm is optimized for low resource usage, today is one of the most used and it was written by Tom Roberts in 1989.

The algorithm uses a precomputed table of 1024 elements to implement sine multiplication thus saving a lot of computing time.

However the execution time of this function on the DSP is of about 10 ms.

After the FFT the code will look for peak values:

```
//Max absolute value search
MaxIdx = 0;
MaxValue = 0;
for(i=0;i<FFTBINS/2;i++)
{
      Modulo32ptr[i]=(int32_t)FFTRealPtr[i]*(int32_t)FFTRealPtr[i]+(in
t32_t)FFTImg[i]*(int32_t)FFTImg[i];
      if(Modulo32ptr[i] > MaxValue)
      {
            MaxValue = Modulo32ptr[i];
            MaxIdx = i;
      }
}

fpeak_d_up = (Fs/2)*MaxIdx/(FFTBINS/2);
```

The same is done for the down slope, after the down slope peak search the range and speed measured by RADAR for the main peak is calculated as follows:

```
freq_r = (fpeak_d_up + fpeak_d_down)/2;
freq_d = fabs(fpeak_d_up - fpeak_d_down)/2.0;
if(MaxValue < MIN_THRESHOLD)
{
      Radar_Range = 0;
      Radar_Speed = 0;
}
else
{
      Radar_Range = freq_r * 3.0e8 * (1.0/(100.0*2.0))/(2.0*230.0e6);
      Radar_Speed = 3.0e8 * freq_d / (2 * 24e9);
}
Radar_Range_filt = Radar_Range_filt * 0.9 + Radar_Range * 0.1;
Radar_Speed_filt = Radar_Speed_filt * 0.9 + Radar_Speed * 0.1;
```

The speed and range is updated only if the peak value is above a noise threshold (MIN_THRESHOLD).

6.5.1.3 Filtering sensor data

Once the data measured by the three sensors is ready it can be sent to the Sieve Filter and then passed to the Kalman Filter:

```
SieveFilter(Radar_Range_filt,(double)Lidar_Range/100.0,(double)Sonar_R
ange/100.0, &z, &H);
kalmanC(KO.x1,KO.P1,&z,&H);
```

The SieveFilter function is the C implementation of the Matlab code:

```
void SieveFilter(double dRadar,double dLidar,double dSonar, MATRIX *
z, MATRIX *H)
{
      // 0 RADAR
      // 1 LIDAR
      // 2 SONAR
      if((dSonar < 7) && (dSonar > 0.1))
      {
            FillMatrix(z,2,0,dSonar);
            FillMatrix(H,2,0,1);
      }
      else
      {
            FillMatrix(z,2,0,0);
            FillMatrix(H,2,0,0);
      }

      if(dLidar == 0)
      {
            FillMatrix(z,1,0,0);
            FillMatrix(H,1,0,0);
      }
      else
      {
            FillMatrix(z,1,0,dLidar);
            FillMatrix(H,1,0,1);
      }

      if(dRadar < 2)
      {
            FillMatrix(z,0,0,0);
            FillMatrix(H,0,0,0);
      }
      else
      {
            FillMatrix(z,0,0,dRadar);
            FillMatrix(H,0,0,1);
      }
      if((dLidar > dRadar + 2) && (elem(H,RADAR,0) == 1))
      {
            FillMatrix(H,LIDAR,0,0);
      }
      else if((dLidar > dSonar + 2) && (elem(H,SONAR,0) == 1))
      {
            FillMatrix(H,LIDAR,0,0);
      }
      if(dRadar > dSonar + 2 && (elem(H,SONAR,0) == 1))
      {
            FillMatrix(H,RADAR,0,0);
      }
}
```

This function is quite lightweight, it takes approximately 300 µs to execute on the DSP. After passing the SieveFilter data is sent to the Kalman filter through the function kalmanC shown below:

```
void kalmanC(double x0,double P0,MATRIX * z,MATRIX * H)
{

      double x;
      double P;
/*function [x1,P1,K]=kalmanC(z,x0,P0,u0,A,B,H,Q,R)
x = A * x0 + B * u0;
P = A * P0 * A' + Q;
y = z - H * x;
S = H * P * H' + R;
K = P * H' * inv(S);
x1 = x + K * y;
P1 = (eye(size(P)) - K*H)*P;
end*/

      x = x0;
      P = P0 + Q;
      Matrix_SCALMPY(H,&H0,x);
      Matrix_SUB(z,&H0,&y);
      Matrix_TRANSP(H,&HT);
      Matrix_SCALMPY(&HT,&PHT,P);
      Matrix_MPY(H,&PHT,&HPHT);
      Matrix_SUM(&HPHT,&R,&S);
      Matrix_INV3(&S,&INVS);
      Matrix_MPY(&HT,&INVS,&HTINVS);
      Matrix_SCALMPY(&HTINVS,&K,P);
      Matrix_MPY(&K,H,&KH);
      Matrix_MPY(&K,&y,&Ky);
      KO.x1 = x + elem(&Ky,0,0);
      KO.P1 = (1 - elem(&KH,0,0))*P;
}
```

This function takes about 2.1ms to execute on the DSP @ 180 MHz

One of the difficulties that I found in translating the Matlab code into C is that C lacks a native matrix elaboration library, so I’ve written it on my own, the functions shown above in the Kalman Filter function are part of this library that the interested reader can find in the Appendix section.

If we take together all the code involved in the elaboration phase the average time of execution is 23 ms, thus taking to a frequency of roughly 43 Hz.

The main amount of time is taken by the FFT function that must be computed for the two slopes.

Figure 6-20 shows the system mounted on a plastic box.

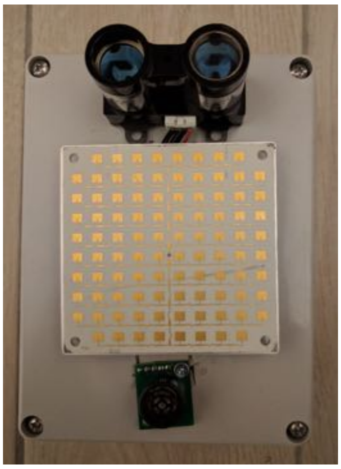


Figure 6-20. Second Experimental setup

6.6 Third experimental setup (Leonardo Autopilot)

Looking at the second experimental setup you can see that most of the computation time is spent calculating the FFT, this kind of computation, very heavy for microprocessors, can be done in hardware with an FPGA thus reducing a lot the computation time.

The third phase of the experiment is to test the multi-sensor system on the Leonardo autopilot, one could obviously write all the code inside the STM32 DSP available on

Leonardo but this would waste a good part of the DSP resources in doing FFT calculations for the RADAR section, leaving little room for the other tasks and, as shown in before, slowing down the whole autopilot.

So I decided to implement the RADAR part on the FPGA while leaving the other sensors connected to the STM32 being the I^{2}C interface almost effortless for the DSP.

The Kalman filter too can be implemented on FPGA, thus reducing the time for calculation. To this purpose, I've written the code for a micro CPU that executes the instructions of the Kalman filter on FPGA.

The whole project has been written in VHDL, the FPGA clock rate has been set at 100 MHz. The development platform is Altera Quartus II web edition 14.1.

As stated before, Leonardo lacks the analog section to read the RADAR, moreover it lacks the ADC connections to FPGA. Anyway, the DSP system designed before has the required interfaces to connect the RADAR directly to the FPGA.

In this case, the ADC chosen to interface with Leonardo is the LTC2312-14, a 14 bit ADC from Linear Technologies.

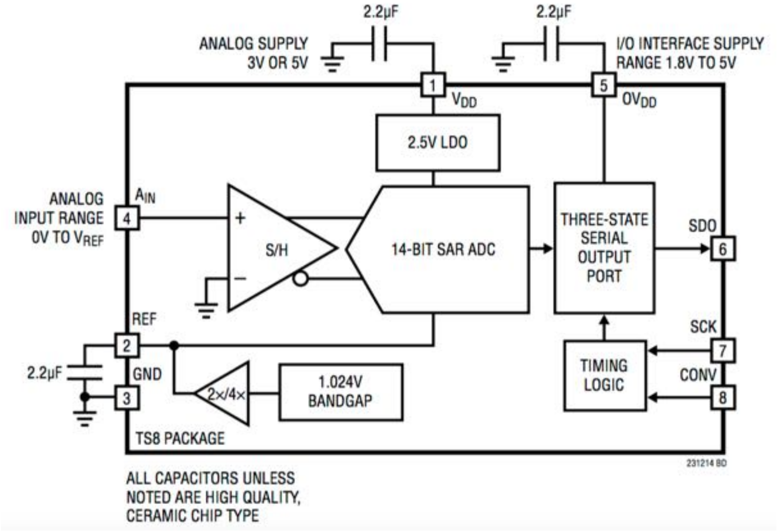

2.2µF
2.2µF
ANALOG SUPPLY 3V OR 5V
I/O INTERFACE SUPPLY RANGE 1.8V TO 5V
1
5
VDD
OVDD
2.5V LDO
ANALOG INPUT RANGE 0V TO VREF
AIN
4
S/H
14-BIT SAR ADC
THREE-STATE SERIAL OUTPUT PORT
SDO
6
REF
2
SCK
7
2.2µF
GND
3
2×/4×
1.024V BANDGAP
TIMING LOGIC
CONV
8
TS8 PACKAGE
231214 BD
ALL CAPACITORS UNLESS NOTED ARE HIGH QUALITY, CERAMIC CHIP TYPE

Figure 6-21 Block diagram of LTC2312-14

The pinout of the ADC is of just 8 pins with a very small package TSOT23, so it could be embedded in the Leonardo autopilot in a future development of the platform. In Table 14 ADC characteristics are shown:

Resolution	14 bit
Sampling frequency	500 ksps
SINAD	77dB @ 5V, 72.6 dB @ 3V
SFDR	88dB

Table 14 LTC2312-14 characteristics

On the FPGA side, I had to implement the ADC interface, the information of the timings can be found on the datasheet of the part and it is shown in Figure 6-22. Just 3 pins are necessary, a rising front on the CONV pin starts the conversion which is timed via an internal oscillator. After a time $t_{CONV-MIN}$ = 1300 ns the device enters a so-called NAP mode, which is a low-power idle mode in which the device sits waiting for commands to come.

Now a negative transition on the CONV pin will make the ADC exit from NAP mode and prepare the MSB bit of the sample on the SDO pin, further transitions on the SCK pin will make the ADC shift out the other bits to the SDO pin till it is arrived to LSB.

This section lasts at least $t_{ACQ\text{-}MIN} = 13.5 * t_{SCK} + t_2 + t_9 = 700$ ns.

This takes to a total $t_{THROUGHPUT} = t_{CONV\text{-}MIN} + t_{ACQ\text{-}MIN} = 1300 + 700 = 2000$ ns.

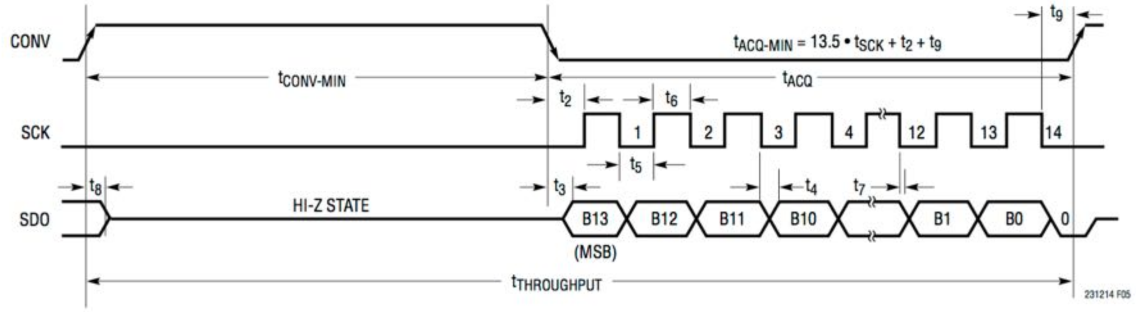


Figure 6-22 LTC2312-14 Timing

On the FPGA ADC entity interface is:

```
entity adc_if is
port(
      clk   :     in std_logic;
      rstn  :     in std_logic;

      --Serial interface
      sclk  :     out std_logic;
      sdi   :     in std_logic;
      conv  :     out std_logic;

      channel :   in std_logic;
      req         :     in std_logic;
      ready :     out std_logic;
      data  :     out std_logic_vector(13 downto 0)
      );
end adc_if;
```

The interface will be driven by a control entity with a mechanism of request-ack. The `channel` input let the controller to choose which ADC to read, while a pulse on the `req` pin will start a conversion on the ADC, when the data is ready the `ready` pin will rise and the data will be available at `data` pins.

The internal code of the ADC interface is made of a state machine which handles the ADC waiting times. In the first tests, I found that some sample of the ADC occasionally came with 0 value, thus creating noise peaks in the FFT. To smooth the incoming data, I decided to apply a median filter to the ADC, this filter has the advantage of totally filtering out single outliers, so it seemed the correct remedy for this problem. I also opened a ticket to Linear Technologies, but I'm still waiting for a satisfying response.

The median filter entity is shown here:

```
entity median is
port(
      clk   :     in std_logic;
      rstn  :     in std_logic;

      Ain : in signed(13 downto 0);
      Bin : in signed(13 downto 0);
      Cin : in signed(13 downto 0);
      Med : out signed(13 downto 0);
      Req : in std_logic;
      Ready : out std_logic
      );
end median;
```

It basically takes three individual samples `Ain`, `Bin`, `Cin`, and outputs the median value of the three on the `Med` pin, this is always managed on a request-acknowledge base, on the `Req` and `Ready` pins.

After the ADC interface there is the windowing block that takes the samples coming from ADC and sends the correct chunks to the FFT block.

The entity of the winblock is shown here:

```
entity winblock is
```

```
port(
            clk         : in std_logic;
            rstn        : in std_logic;

            --to adc_if
            adcready          : in std_logic;
            adcdata     : in std_logic_vector(13 downto 0);
            adcchan     : out std_logic;
            adcreq      : out std_logic;

            --to fft256
            fft_sop     : out std_logic;
            fft_eop     : out std_logic;
            fft_real    : out std_logic_vector(13 downto 0);
            fft_ready:  in std_logic;
            fft_valid: out std_logic;

            --from ext
            start       : in std_logic
            );
end winblock;
```

In the entity code there are the two interfaces, the former towards the ADC and the latter towards the FFT block, there is also a `start` pin that is used by the windowing block to synchronize the start of the window with the correct position in the modulating waveform.

While the ADC samples arrive they are stored in a FIFO memory, this is to give enough time to the logic sitting past the ADC to multiply the data by the Hanning function. In VHDL this is performed by using a hardware multiplier and a ROM memory, in this memory are saved pre-calculated values for the Hanning function. When data is ready, the state machine will p send everything to the FFT block. In the appendix the source code of the winblock entity is shown.

The modulating waveform itself is generated by another block, the triangle generator, that generates the triangle in the form of a PWM signal at the output pin `triangle_out`. The `sync` pin generates a pulse at every change in slope of the triangle wave, the entity of the triangle generator is shown below:

```
entity triangle_gen is
port
(
      clk : IN std_logic;
      rstn: IN std_logic;
      triangle_out : OUT std_logic;
      sync : OUT std_logic
);
end triangle_gen;
```

The FFT block is proprietary from Altera. I could have implemented it on my own, however the Altera version is optimized for the Cyclone IV FPGA and is very efficient.

The setting up of the FFT block is easy and is done using Altera QSYS configurator, the chosen configuration is:

- 256 bins
- Burst type

  The Altera FFT IP core can operate in four different ways: Variable Streaming, Streaming, Buffered Burst, Burst. The first two are for pure Floating Point mathematics, but waste a lot of FPGA area. Buffered Burst can operate in Fixed Point but still trade area for speed, Burst is the method that operates in Fixed Point and consumes less area on FPGA thus paying a bit on the throughput.
- *Block Floating Point Output*

  The output format of the FFT is Block Floating Point, it is a particular format that is a trade-off between fixed-point and full floating-point.

  In fixed-point FFT the data precision must be large enough to represent all the values of the transform computation. For large FFT transform sizes the data width can become excessive or you can have a loss of precision.

On the other hand, floating-point FFTs, representing every value with an independent mantissa, allow to manage huge numbers, but floating-point logic is more complicated and to keep precision high it requires a lot of resources.

- Natural Order Input-Output

  The output order of samples can be natural, bit-reversed or digit-reversed. These output modes depend on the internal structure of the FFT, and normally to have a reduced latency one must accept to have outputs bit-reversed or digit-reversed. In this case, because we can accept an additional latency I chose natural order for both input and output.

The code for the fft256 entity is:

```
entity fft256 is
      port (
            clk         : in std_logic := '0';
            reset_n     : in std_logic := '0';
            sink_valid  : in  std_logic := '0';
            sink_ready  : out std_logic;
            sink_error  : in std_logic_vector(1 downto 0) := (others =>
'0');
            sink_sop    : in std_logic := '0';
            sink_eop    : in std_logic := '0';
            sink_real   : in std_logic_vector(11 downto 0) := (others
=> '0');
            sink_imag   : in std_logic_vector(11 downto 0) := (others
=> '0');
            inverse     : in std_logic_vector(0 downto 0) := (others =>
'0');
            source_valid : out std_logic;
            source_ready : in  std_logic := '0';
            source_error : out std_logic_vector(1 downto 0);
            source_sop  : out std_logic;
            source_eop  : out std_logic;
            source_real : out std_logic_vector(11 downto 0);
            source_imag  : out std_logic_vector(11 downto 0);
            source_exp   : out std_logic_vector(5 downto 0)
      );
end fft256;
```

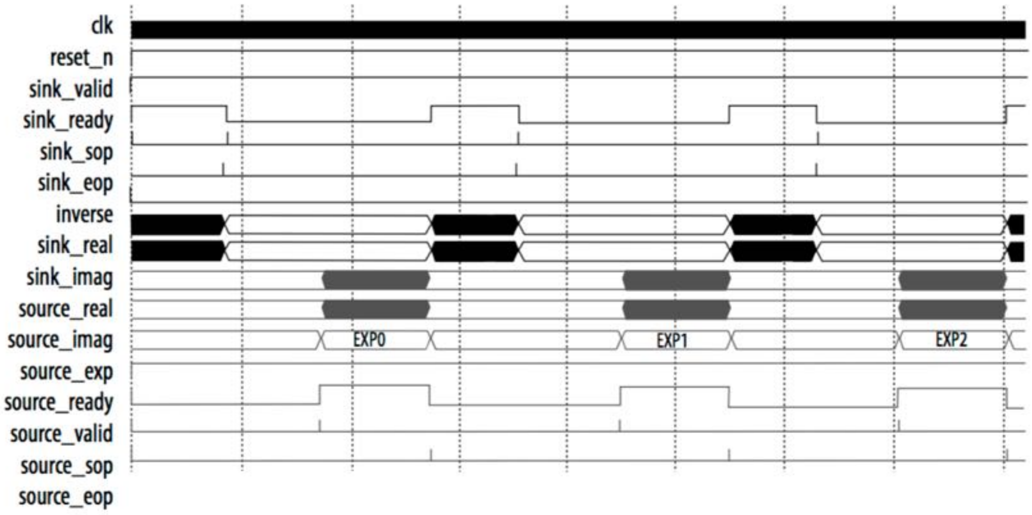


Figure 6-23 FFT block timing

Figure 6-23 shows the timing for the FFT block, when the data is ready the windowing block puts it in `sink_real` register, raises the `sink_ready` bit and issues a pulse on the `sink_sop` bit, at every clock cycle the windowing block shifts a new sample in the FFT block, till it arrives at 256 samples, on the last sample the windowing block issues a pulse on the `sink_eop` bit and it lowers the `sink_ready` bit.

After a certain latency time the FFT block outputs the transformed data in this way:

Puts the data in the `source_real`, `source_imag`, `source_exp` signals, where `source_exp` is the exponent of the block floating point notation, it raises the `source_ready` signal and issues a pulse on the `source_valid` signal. The FFT block continues to shift out all the FFT samples till it arrives at 256 samples, on the last sample the FFT block issues a pulse on the `source_eop` bit and it lowers the `source_ready` bit.

This interface is very convenient, since it easily connects with a FIFO memory.

The output of the FFT block is then analyzed by the fft_analyzer block, which seeks the peaks in the FFT and outputs it on UART. The code for the fft_analyzer entity is written below:

```
entity fft_analyzer is
PORT
(
      clk           : in std_logic;
      rstn          : in std_logic;

      fft_ready : out std_logic;
      fft_sop       : in std_logic;
      fft_eop       : in std_logic;
      fft_valid : in std_logic;
      fft_real  : in std_logic_vector(11 downto 0);
      fft_imag  : in std_logic_vector(11 downto 0);
      fft_exp       : in signed(5 downto 0);

      UART_TX : out std_logic;
      UART_RX : in std_logic
);

END fft_analyzer;
```

The timing diagram above shows the RADAR processor for a single slope of the FMCW triangle signal, the time elapsed is about 8 µs

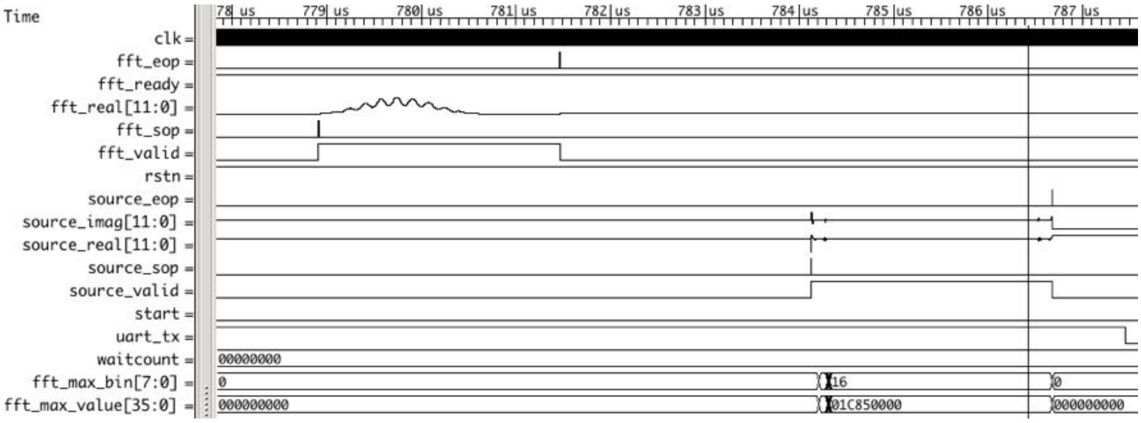


Figure 6-24 Timing of the RADAR processor system

The results of the synthesis of the RADAR signal processor is shown above, the occupation of the FPGA resources is about the 20%:

Timing Models	Final
Total logic elements	4,661
Total combinational functions	3,664
Dedicated logic registers	3,605
Total registers	3605
Total pins	9
Total virtual pins	0
Total memory bits	17,088
Embedded Multiplier 9-bit elements	38
Total PLLs	1

6.6.1 Implementation of the Kalman Filter on FPGA

As shown before in 6.3 the operations involved in a Kalman filter are mainly matrix addition and multiplications plus a matrix inversion. These kind of matrices operations can have high computational costs even for a relative small Kalman filter like the one used in our multi-sensor system that only counts 3x3 matrices. As stated before, the execution of a single cycle of the Kalman filter on the STM32F4 DSP lasted about 2.1ms, for our application can be still enough, however exploiting the pipelining inside the FPGA more interesting results can be achieved.

In my implementation, to limit the resources on the FPGA I built a simple processor architecture inspired to the Harvard model, like shown in Figure 6-25. The Program Memory block contains all the instructions necessary to execute a Kalman filter cycle, the ALU block contains the basic floating point arithmetic units to perform additions, subtractions, multiplications and divisions, the Data Memory block will keep the variables used during the operations. The Processing Unit is the brain of the Kalman Filter processor, it is capable of doing matrix operations like inversion or matrix multiplications.

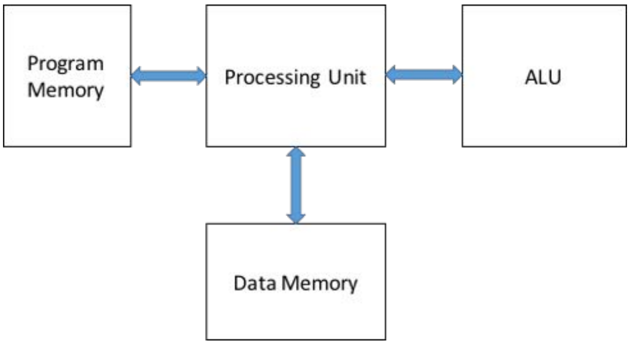


Figure 6-25 Block scheme of the Kalman processor

6.6.1.1 ALU Block

The ALU block entity is formed by two 32-bits-wide inputs for the operands IN1, IN2, these inputs are single precision floating point IEEE-754 format.

The OPER input, 2 bits wide for the operator. The EN input to enable the ALU.

The VALID output that is pulled HIGH by the ALU when at least one operation is finished and a valid data is available at the RESULT output.

The entity declaration of the ALU block is shown below:

```
entity alu is
port(
            clk         : in std_logic;
            rstn        : in std_logic;

            IN1               : in std_logic_vector(31 downto 0);
            IN2               : in std_logic_vector(31 downto 0);
            RESULT            : out std_logic_vector(31 downto 0);

            OPER        : in std_logic_vector(1 downto 0);
            EN          : in std_logic;
            VALID       : out std_logic
            );
end alu;
```

The behavior of the ALU block is shown below:

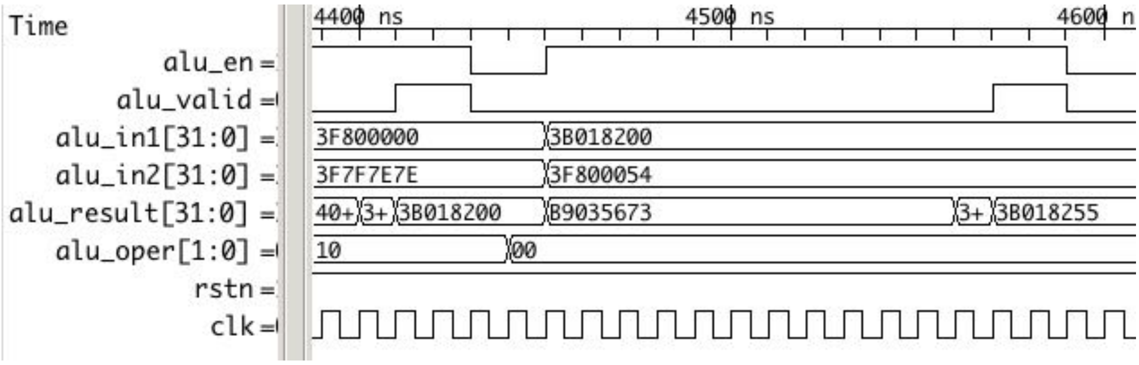


Figure 6-26 Timing of the ALU block

When the ALU block is enabled through the alu_en signal the ALU block takes the inputs IN1, IN2 and OPER and starts counting the time according to the expected duration of the selected operation, the expected times are:

- Addition/Subtraction component: 10 cycles
- Multiplication component: 10 cycles
- Division component: 14 cycles

For the interested reader in the source code of the ALU block is available in the Appendix.

6.6.1.2 Processing Unit Block

The Processing Unit executes the program stored in a ROM memory hardcoded inside the FPGA.

The entity of the Processing Unit is called CPU and is shown below:

```
entity cpu is
generic(
      PROGRAM_LEN : integer := 13

      );
port(
            clk         : in std_logic;
            rstn        : in std_logic;

            START_in    : in std_logic;

            x0                : in std_logic_vector(31 downto 0);
            P0                : in std_logic_vector(31 downto 0);

            H                 : in std_logic_vector(2 downto 0);

            z1                : in std_logic_vector(31 downto 0);
            z2                : in std_logic_vector(31 downto 0);
            z3                : in std_logic_vector(31 downto 0);

            x1                : out std_logic_vector(31 downto 0);
            P1                : out std_logic_vector(31 downto 0);

            DONE        : out std_logic
);
end cpu;
```

The entity accepts as input the Kalman variables x0 (that is the actual state of the system), P0 that is the actual process noise of the system, H that is the observation matrix and z1, z2, z3 that are the measurements coming from the three sensors.

The program is started with by raising the signal START_in, at the end of the program the Processing Unit put the output on the x1 and P1 outputs and raises high the DONE signal.

The Processing Unit is made by a single finite states machine made by 25 states, the main states are:

IDLE: in this state the machine waits indefinitely for a high signal on the START input, when the start input is detected the program counter signal is reset and the machine advances to the START state.

START: in this state the inputs of the CPU block are loaded in the data memory and the machine starts loading the first instruction.

LOAD: in this state the instruction is loaded and the machine goes to the state of execution requested by the program.

The instruction is 28 bits wide and is composed by 3 operands of 8 bits each, 1 operator of 4 bits.

EXECx: these are the states in which the processing unit performs the matrix operations. The code is written to maximize the pipelining of the operations in order to minimize the time for each instruction.

PC_FWD: in this state the program counter is advanced, if the program has reached the end the DONE output goes high and the signals x1 and P1 are available at the output.

One of the toughest operation to perform in a Kalman Filter is the matrix inversion, this operation is necessary to resolve the following equation:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

To calculate the inverse of a 3x3 matrix the first step is to calculate the minors, this step is necessary both for calculating the determinant and the adjugate (the transpose of the cofactor matrix):

$$A^{-1} = \frac{1}{\det(A)} \operatorname{adj}(A)$$

In order to calculate the minors in pipeline I prepared a table with the succession of elements of the matrix that must be multiplied and then subtracted in sequence. The elements will build the adjugate matrix:

$$\mathbf{A}^{-1} = \frac{1}{\det \mathbf{A}} \begin{pmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{pmatrix}$$

1) a_{22},a_{33}

2) a_{23},a_{32}

3) a_{13},a_{32}

…

18) a_{12},a_{21}

In this way the calculation of the inverse matrix is very efficient, and it takes about 126 clock cycles, that is 4.6 μs at 100 MHz clock.

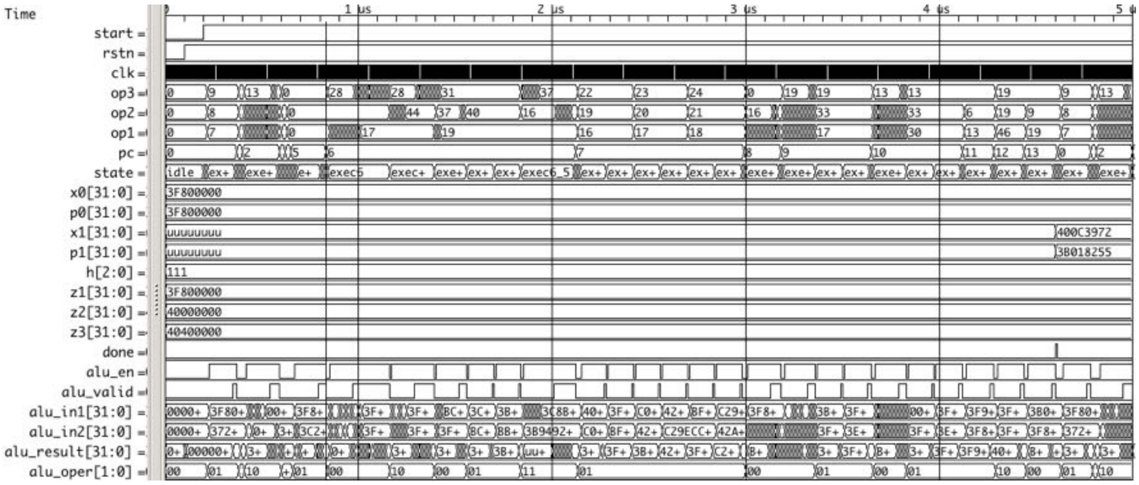


Figure 6-27 Timing of the Kalman Processor

I synthesized the above block in the FPGA, the VHDL of the cpu entity contains also the alu block, so the synthesis gives the occupation of the whole Kalman processor on the FPGA, the results of the synthesis are shown below:

Top-level Entity Name	cpu
Family	Cyclone IV E
Device	EP4CE22E22C8
Timing Models	Final
Total logic elements	13,693
Total combinational functions	13,023
Dedicated logic registers	3,216
Total registers	3216
Total pins	231
Total virtual pins	0
Total memory bits	5,137
Embedded Multiplier 9-bit elements	23
Total PLLs	0

The occupation is roughly the 50% of the FPGA space in terms of logic elements.

The next step is to put all the system together, both the fmcw block and the kalman processor block and synthesize it on the FPGA, after the correct synthesis of the system, the design is fitted in the physical FPGA through the Fitter (Place & Route) step, the result is shown above:

Top-level Entity Name	fmcw_top
Family	Cyclone IV E
Device	EP4CE22E22C8

Timing Models	Final
Total logic elements	18,024 / 22,320 (81 %)
Total combinational functions	16,789 / 22,320 (75 %)
Dedicated logic registers	6,728 / 22,320 (30 %)
Total registers	6728
Total pins	12 / 80 (15 %)
Total virtual pins	0
Total memory bits	22,225 / 608,256 (4 %)
Embedded Multiplier 9-bit elements	61 / 132 (46 %)
Total PLLs	1 / 4 (25 %)

The FPGA is almost full, but the design fits correctly inside it.

The difference in performance between the STM32 solo implementation and the FPGA-STM32 implementation is huge:

On the RADAR signal elaboration side: 20 ms roughly for two FFT calculations on STM32 vs roughly 16 μs on FPGA.

On the Kalman processing side: 2.1 ms roughly for a single Kalman cycle on STM32 vs 4.7 μs roughly on FPGA.

In Figure 6-28 the test setup is shown:

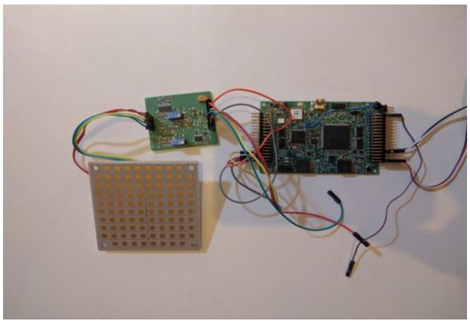


Figure 6-28 Leonardo Autopilot test setup

6.7 Case studies

In order to test the assumptions made so far, several tests have been carried out. The experiments have been realized in our laboratories and in an outdoor environment and we analyzed various kinds of obstacles known to pose problems in detection with current technologies, so we focused on poles, nets, vegetation, windows and persons.

In the following subsections, the various case studies addressed in this work are shown. Measurements are made by pointing the device towards the target for 5 seconds with an output rate of 2 Hz.

6.7.1 Wall

As a first case study, here we propose a brick wall, clearly it's a simple obstacle, but we'll take this case as reference for the other tests. Figure 6-29 shows the output of the three sensors and in the line with circle markers shows the output from our Kalman filter.

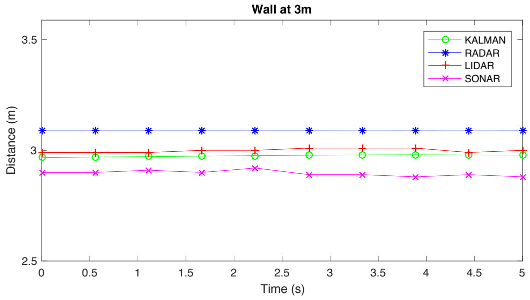


Figure 6-29. Wall at 3m distance

Figure 6-30 shows the results for the same wall at 10m distance, this example shows how the Kalman output ties to the LiDAR output, since it is more confident of its measurements, while the rule sieve has discarded the SONAR measurement altogether.

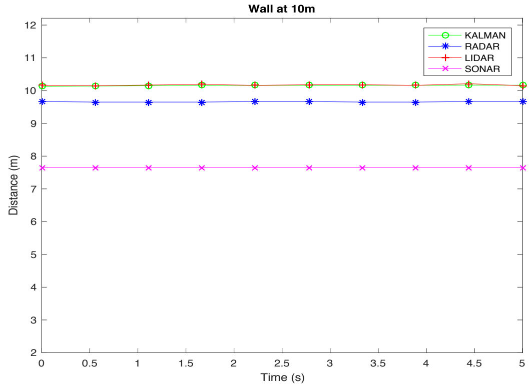


Figure 6-30. Wall at 10m distance

6.7.2 Window

The second case study is a window (Figure 6-31). This kind of obstacles stress the difficulties of the LiDAR technology that is deceived by the glass.

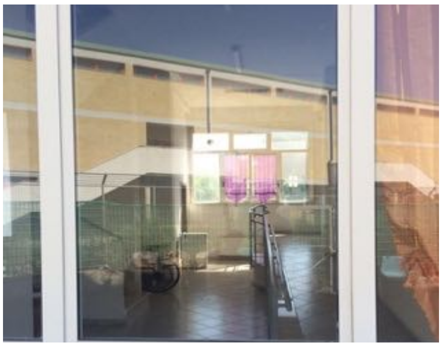


Figure 6-31. Window used for the experiment

This is evident in Figure 6-32: in fact, the LiDAR data is totally wrong most of the time. Nevertheless, the Kalman filter output is still good, because the rule filter discarded the measurements of the LiDAR.

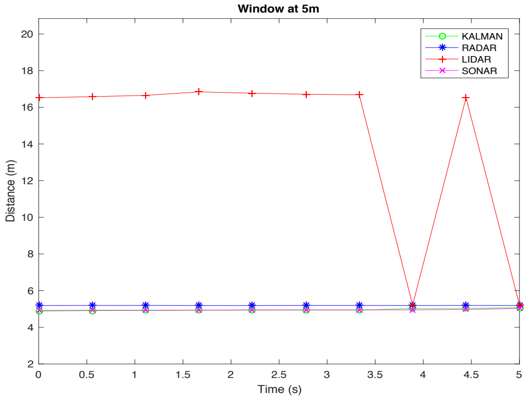


Figure 6-32. Window at 5m distance

6.7.3 Pole

A difficult obstacle is a pole (Figure 6-33), due to its long and thin form factor it is hard to detect at distance with sensors characterized by a narrow FOV.

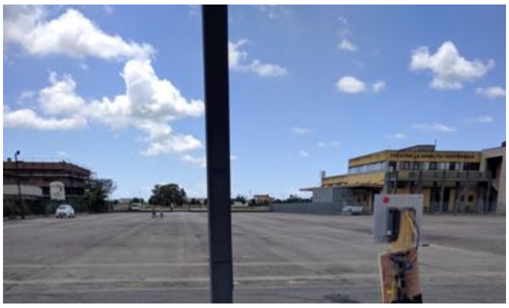


Figure 6-33. Experimental setup, Pole

As it is shown in Figure 6-34, at 5m distance, pointing the sensors straight to the pole, it is correctly detected by all sensors, so the Kalman filter gives more trust to the LiDAR which has the smaller measurement noise.

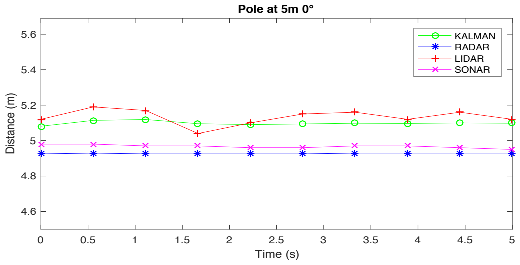


Figure 6-34. Pole at 5m distance, 0° angle

Figure 6-35 is taken again at 5m, but slightly tilting the sensors by few degrees. The LiDAR is no more pointing correctly to the pole thus measuring the background scene, while SONAR and RADAR correctly detect the pole. The Kalman filter output follows the latter two sensors. Tilting the sensors more, like in Figure 6-36, causes the RADAR to lose the target that gets detected just by the SONAR. Another test has been done by approximately pointing straight to the pole at 15m distance (Figure 6-37). The obstacle is detected just by the RADAR sensor and the Kalman filter converges slightly more slowly.

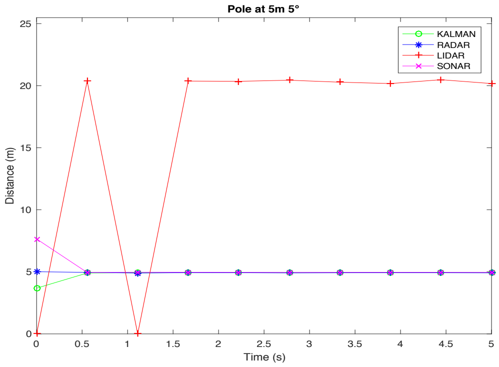


Figure 6-35. Pole at 5m distance, 5° Angle

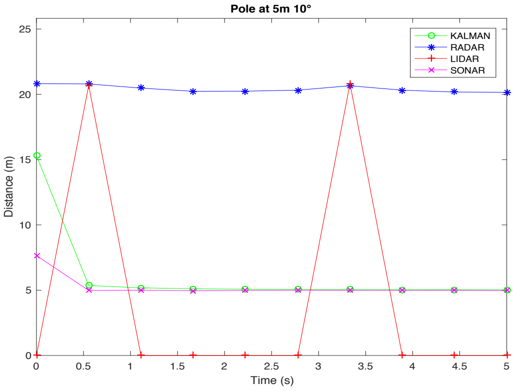


Figure 6-36. Pole at 5m distance, 10° Angle

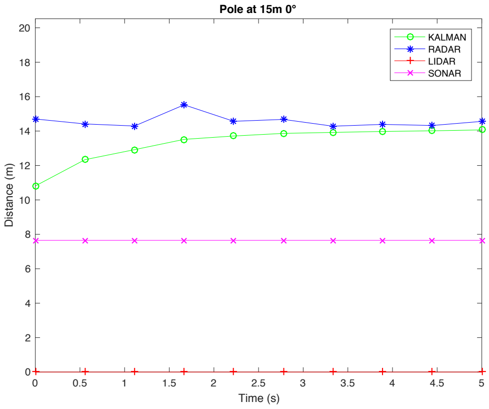


Figure 6-37 Pole at 15m distance, 0° Angle

6.7.4 Metal Net

A challenging obstacle is a Metal Net (Figure 6-38), this kind of target is difficult to be detected using a LiDAR or a SONAR due to its form factor while the RADAR easily detects it.

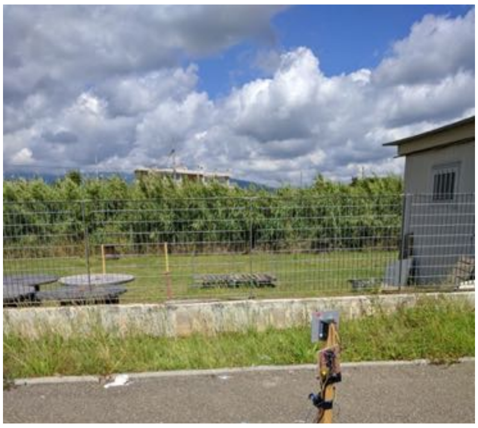


Figure 6-38. Metal Net, experimental setup

Figure 18 is taken at 3m distance, where RADAR and SONAR correctly detect the target, LiDAR is yet failing to give correct measurements.

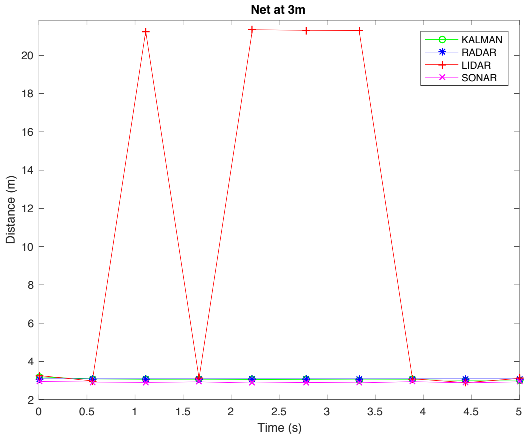


Figure 6-39 Metal Net at 3m distance

At 5m even the SONAR fails to correctly detect the distance, as shown in Figure 6-40.

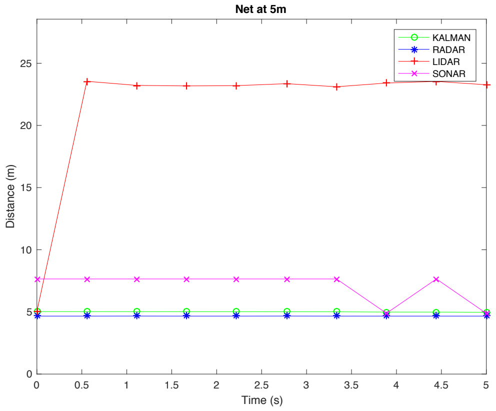


Figure 6-40 Metal Net at 5m distance

The behavior is like the previous one even increasing the distance from the target as shown in Figure 6-41.

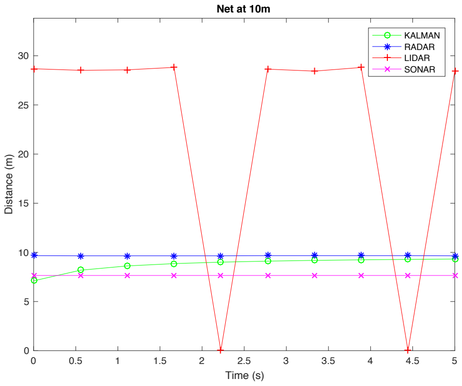


Figure 6-41. Metal Net at 10m distance

A dynamic test has been performed with the multi-sensor system pointing towards the net starting from 5m and then closing up to the net like shown in Figure 6-42:

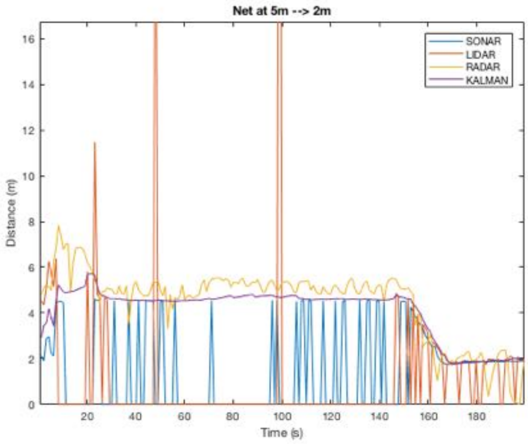


Figure 6-42 Net obstacle approaching from 5m to 2m

The results show that even in this case the RADAR is very useful to the detection of the obstacle, because the LiDAR often fails to see it, the SONAR behaves a little better but only sees well the obstacle when it is at a very close distance.

6.7.5 Person

Another interesting case study is the detection of persons. Figure 6-43 shows the detection of a person at 5m distance, here the worst sensor is the SONAR that fails to detect the human as an obstacle. However, LiDAR and RADAR behave correctly and the system detects the person successfully.

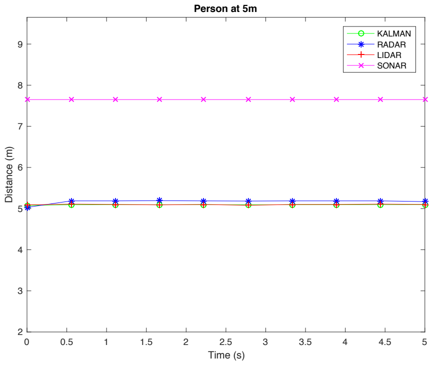


Figure 6-43. Person at 5m distance

Going further at 10m distance as shown in Figure 6-44, the RADAR too sometimes doesn’t see the obstacle but the overall system continues to correctly detect it.

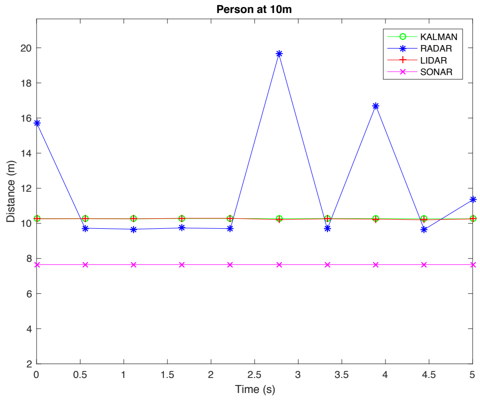


Figure 6-44. Person at 10m distance

6.7.6 Vegetation

Vegetation (Figure 6-45) can be a tough obstacle to detect due to the heterogeneity of its form factor.

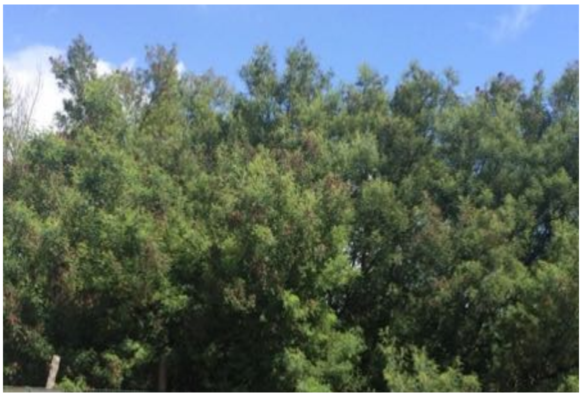


Figure 6-45. Vegetation obstacle

As can be seen by the evidence of the experiment in Figure 6-46 Ultrasonic sensor is easily deceived by this obstacle, while the best behavior here is the one of LiDAR, Figure 6-47 and Figure 6-48.

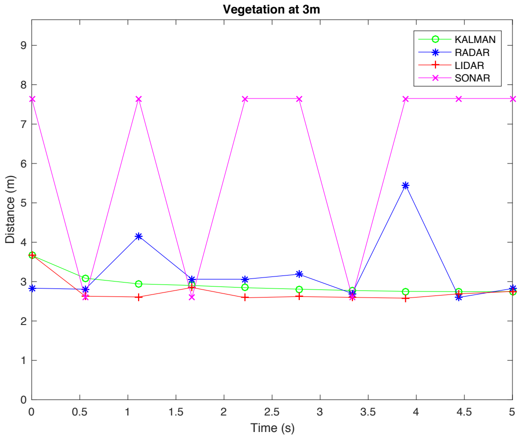


Figure 6-46. Vegetation at 3m

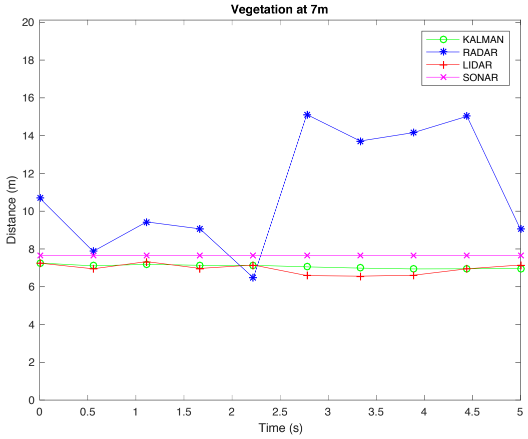


Figure 6-47. Vegetation at 7m

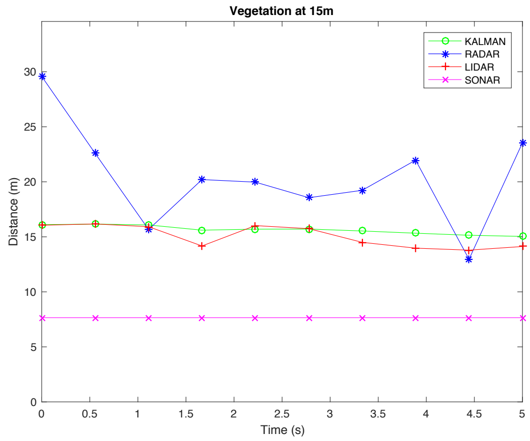


Figure 6-48. Vegetation at 15m

A dynamic test has been done with the sensors aimed at the vegetation, in this case a hedge, starting from a distance of 8m and then approaching progressively until reaching 2m.

As shown in Figure 6-49 the multi-sensor system correctly detects the hedge from the beginning, however, due to the distance the SONAR doesn't see the hedge until the sensor reaches a distance of roughly 5m, from this moment on, every sensor correctly detects the obstacle.

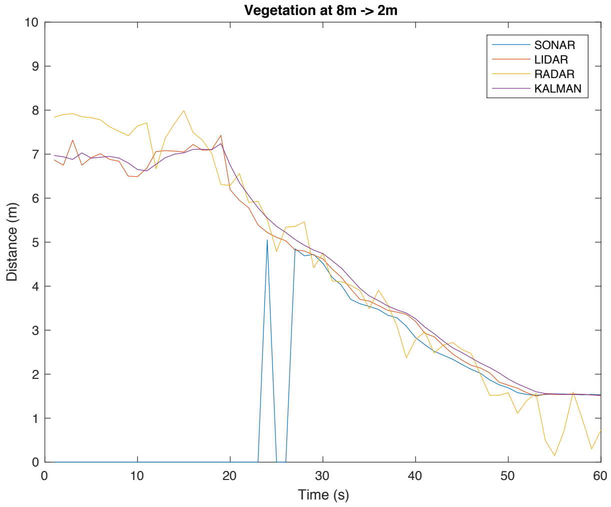


Figure 6-49 Vegetation at 8m approaching

6.7.7 *Test Flight with sensor mounted onboard a RPAS*

Once demonstrated that the system has a good behavior on several kinds of tough obstacles, the system, mounted in a plastic box, has been mounted on-board a lightweight RPAS (2 Kg), as shown in Figure 6-50.

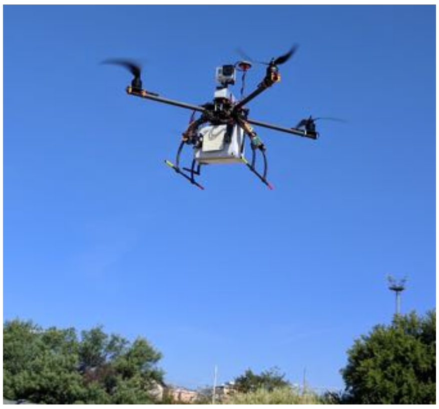


Figure 6-50 Multi-sensor system mounted on a RPAS

To better understand the operation of the multi-sensor system, a camera has been mounted on the RPAS aimed in the same direction of the sensor. The camera is a GoPro Hero 3+, it is an action camera with a fisheye optic that can acquire videos at 1920x1080 pixels at 60 fps.

The tested scenario is flying towards a metal net, like shown in Figure 6-51 the drone is pointing a metal net and it is at a distance of 5 meters from it.

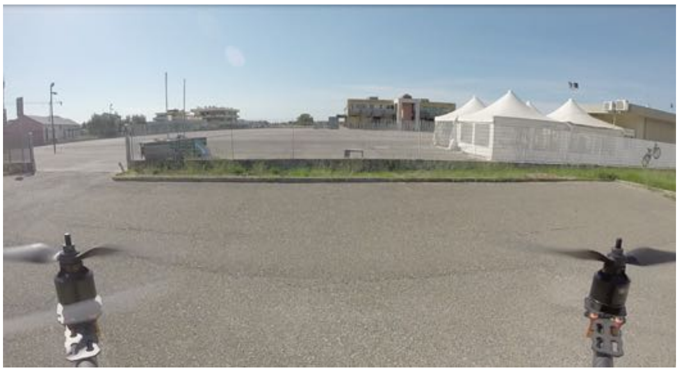


Figure 6-51 RPAS flying towards a metal net

The multi-sensor system detected the metal net correctly, with respect to the static case shown in section 6.7.4 the response of the radar is noisier, and the other sensors suffer from the noise onboard the RPAS also, however the Kalman filter behavior is correct and the net is correctly detected. The first seconds of acquisition are at different heights, this is normal, because the sensors here are slightly pointing the ground, because of the different FOVs of the three sensors the distances are detected differently on the ground.

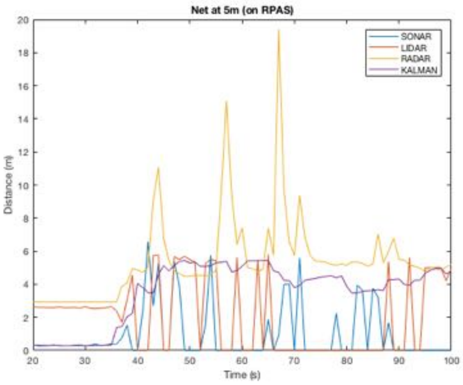


Figure 6-52 Metal net detected by multi-sensor system

Another test has been made pointing towards a building, like shown in Figure 6-53, the building is a relatively simple obstacle, however it presents parts like windows, that can deceive an obstacle detection system made by a single sensor.

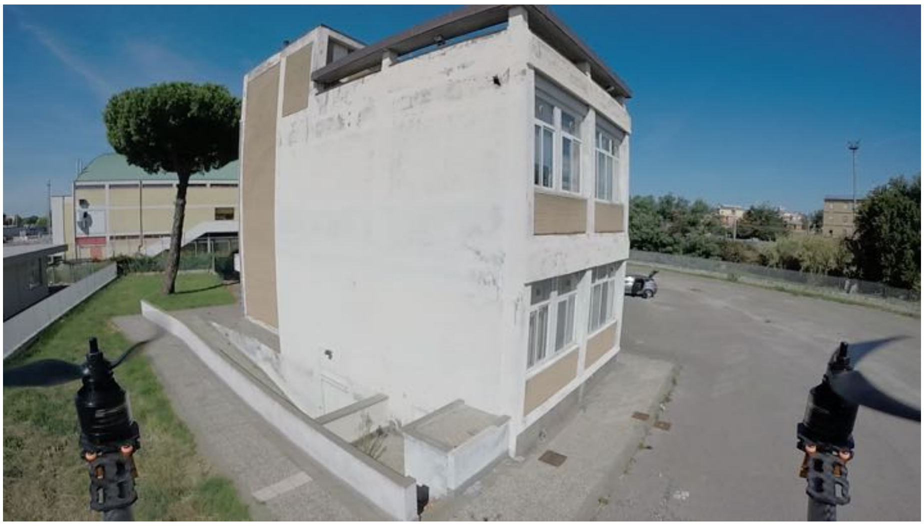


Figure 6-53 RPAS flying towards a building

Figure 6-54 shows the building being detected by the system, this time the best sensor is the LiDAR, while the RADAR is too noisy and the SONAR is out of range.

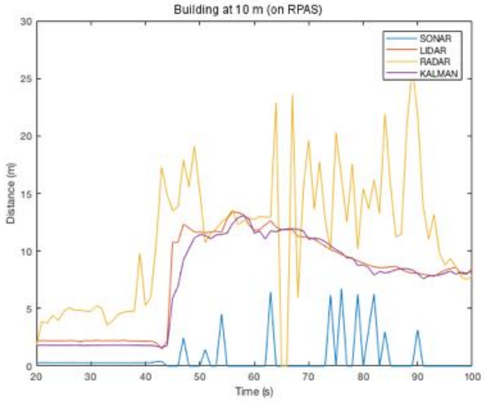


Figure 6-54 Building detected by multi-sensor system

The last case shown is the drone flying towards a tree as shown in Figure 6-55.

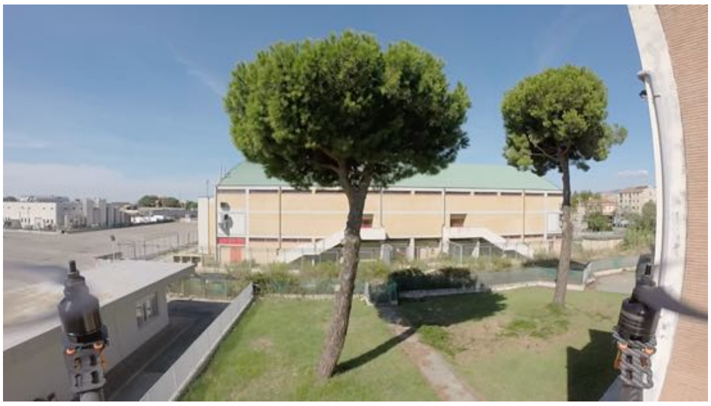


Figure 6-55 RPAS flying towards a tree

The drone flies at roughly 12 m from the tree and the obstacle is correctly detected by the system, as shown in Figure 6-56.

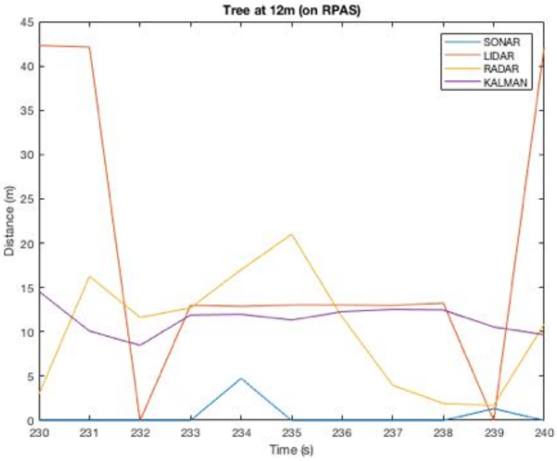


Figure 6-56 Tree detected by multi-sensor system

6.8 Considerations

The multi-sensor approach to obstacle detection using Kalman filter gives promising results: the three sensors complementary capabilities are exploited by the filter, giving an accurate estimation of obstacle distance.

Moreover, the data coming from FMCW RADAR itself are very interesting. In fact, such technology behaves particularly well in all those situations where optical systems fail. However, often practical implementations of obstacle detection systems for UAVs lack this kind of technology because RADAR systems are typically bulky and power thirsty. In this research work I also have demonstrated that miniaturization of this technology has made possible the integration even on UAVs. For instance, the setup shown in this work has a power consumption of approximately 2W and a weight of less than 0,2 Kg.

7 INTEGRATION IN SHARED AIR SPACE

As stated early in chapter 2.4 one of the most important task to be fulfilled to enable autonomous guidance of RPAS is the integration in a shared airspace. Today our skies are busy with millions of flights of traditional planes, in order for this to be possible even with RPAS it must be necessary for them to communicate with other aircrafts and with the air traffic control authority to identify, signal their own position, ask right of fly and so on.

As of today there's nothing of it and when a RPAS must do an operation in a zone that could be busy with other air traffic, the RPAS operator must signal it to the local authority that must emit a NOTAM (notification to air men). This system is somewhat sluggish because the bureaucracy times are in the order of days if not weeks.

So this is a bottleneck to the broad diffusion of RPAS operations.

The regulators are moving to overcome these obstacles and are trying to converge on a system for automatically trace the presence of RPAS and to allow or deny their flight.

As stated before, for RPAS weighing less than 150 kg regulations are different country by country. In Italy starting from July 1st 2016 every RPAS must have an electronic device onboard that guarantees its identification and that allows the transmission in real time of the information about the owner, the operator and the fundamental flight data, and should be able to register them. Characteristics of the system are fixed by ENAC, the Italian regulator (the practical application of this regulation is still in process of being defined).

7.1 Data link

The data link is one of the side problems that must be considered when thinking at the integration of RPAS in common air-space. Today drones are typically piloted using remote controls directly taken by the RC model world. In Europe it is normally used the 2.4 GHz band, a band that is full of devices from other technologies like WiFi and Bluetooth. RCs used to pilot drones transmit the data necessary to move the drone in the air and in the more advanced models they even include some telemetry information. Video signal is also often necessary in drones application, today another transmitter is present on drones commonly in the band of 5.8 GHz. If one has to convey more advanced telemetries the 433 MHz band is often used as a viable channel for this information.

As a result, for a single drone different channels are commonly used and the reliability of communications is often left to the operator that must verify himself the goodness of communication before to start a mission.

Moreover, the communication between drone and ground station, by coming from the amateur world, is often unencrypted leaving the side uncovered to eavesdropping.

Figure 7-1 RC remote used in RPAS applications

A common solution must be found that establish clearly the band occupation of a single RPAS and that together guarantees reliability of the link.

A promising solution seems to be MAVLINK, a Micro Air Vehicle Communication Protocol, it is open source and it was first released in 2009 by Lorenz Meier.

MAVLINK protocol is lightweight as it has got a small overhead, it is message based and it is well suited to be encapsulated in a UDP protocol.

MAVLINK is stateless, it is up to the ground-station to verify if the system is still alive using the heartbeat message, the heartbeat message rate is flexible and can be configured to be every 60, 30, 10 or 1 second, even if 1 second is strongly recommended. A system will be considered connected only if the heartbeat arrives.

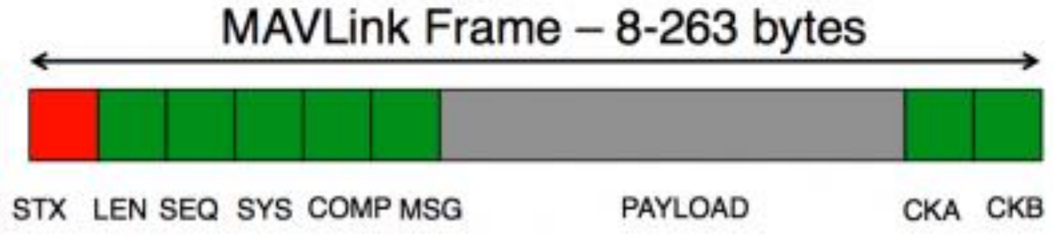


Figure 7-2 MAVLINK frame

Figure 7-2 shows the structure of the MAVLINK frame. It is interesting to see how the header part is limited to just 6 Bytes, leaving much room free for the payload, the checksum is relatively strong with two bytes at the end of the packet.

The SEQ byte indicates the sequence of the packet and help the system to reconstruct the correct sequence of messages especially in case of loss of packets, event very frequent in a wireless link.

Byte Index	Content	Value	Explanation
0	Packet start sign	V1.0: 0xFE	Indicates the start of a new packet.
1	Payload length	0-255	Indicates length of the following payload.
2	Packet sequence	0-255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1-255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0-255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0-255	ID of the message - the id defines what the payload “means” and how it should be correctly decoded.
6 to (n+6)	Data	(0 – 255) bytes	Data of the message, depends on the message id.
(n+7) to (n+8)	Checksum (low byte, high byte)	ITU X.25/SAE AS-4 hash, excluding packet start sign, so bytes 1..(n+6) Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).	

Table 15 MAVLINK frame content

In order to be a good candidate for drone communication MAVLINK needs to be secure, actually it is unencrypted, but these limits should be overcome with sMAVLINK, an encrypted extension of the protocol.

sMAVLINK comes with symmetric key cryptography for performance and power consumption limits, it uses the following cryptographic algorithms:

-AES-GCM (Galois Counter Mode), it will provide authenticated encryption

-SHA-256 for key derivation

sMAVLINK is still a work in progress but is in the right direction because it is thought just for drones applications, and is not adapted from other fields.

7.2 Participation to ENAC Call for Demonstration

In fall 2015, in view of integration of RPAS into shared airspace, ENAC issued a call for demonstration for a system of Electronic Identification that permit the transmission from the RPAS of flight information like:

- Altitude
- Speed
- Operator identification data
- Position

The requirements for the system were:

- Transmission of flight data from RPAS to a remote platform
- Recording and storage of flight data to a remote platform

- Data availability both Real-time and delayed

In response to the call of ENAC the research group in which I study, in collaboration with Oben s.r.l., a spin of the university of Sassari, we prepared a demonstrative system based on a small embedded board that I developed for Oben: TAG board.

7.2.1 TAG Board

TAG is a board born for the synchronization of the images taken by a camera on board a drone when used in photogrammetry applications.

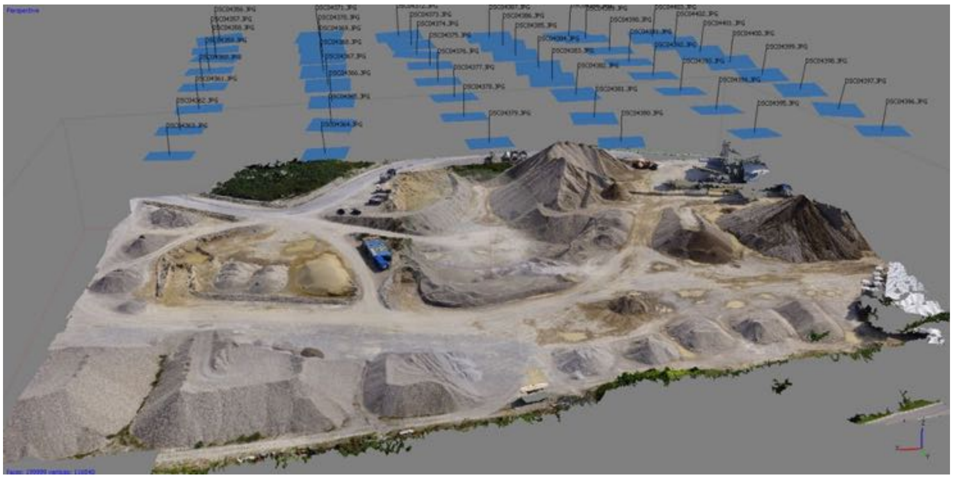


Figure 7-3 Photogrammetry example

Photogrammetry is a technique that allows to make measurements from photographs. Oben is a startup that does surveys and elaborations with RPAS, one of the main problems that Oben encounters when doing photogrammetry is the huge time spent in the pre-elaboration phase, where the acquired pictures must be sorted and georeferenced. This phase is often done by hand, pasting the GPS data with every single image, moreover, the software used for reconstruction takes a lot of time to reconstruct the angle and the direction at which the picture was taken. From this problem starts the need of Oben for a device that simplifies the pre-elaboration phase by joining together three information:

- Camera attitude
- GPS positioning
- picture shot timing

In order to do these tasks TAG is equipped with:

- GPS receiver with enhanced timing capabilities
- 32 bit 100 MHz microcontroller
- uSD slot to store the events recorded by the board
- IMU to record the actual attitude of the camera at the time of the shot
- Isolated inputs, outputs to drive the camera shutter
- Serial Expansion interface to add for example a rf telemetry interface
- can be positioned close to the camera (that can be a DSLR camera or a thermal-camera as well) and it is directly plugged to its remote shutter port.
- USB interface for configuration

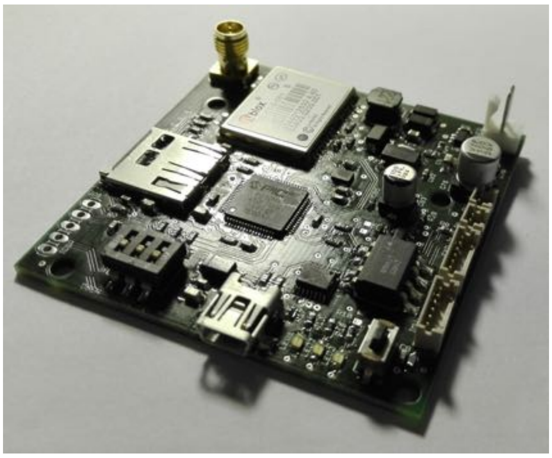


Figure 7-4 TAG Board

The IMU is on a remote board, the choice to put it separately from the main board has been made because the main board can be put fixed on the drone frame, while the IMU board can be put directly on the camera, in order to measure the attitude of a gimballed camera separately from the attitude of the RPAS.

For the IMU board I chose to use a BNO055 Intelligent 9-axis absolute orientation sensor from BOSCH. The key features of this IMU are that it can output directly the fused sensor data which is provided on the I^2C bus in the form of Euler angles or of quaternions. The IMU is connected to the main board through a 6 pins connector.

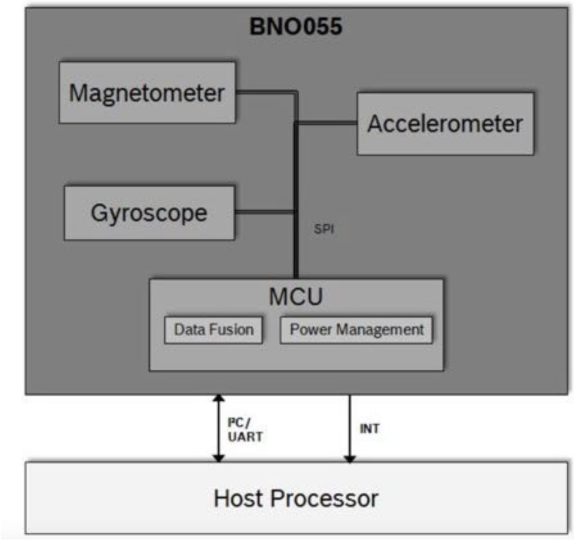


Figure 7-5 Block scheme of BNO055

The IMU board is very small, just 25x25mm, in Figure 7-6 it is compared with a 20 cents coin.

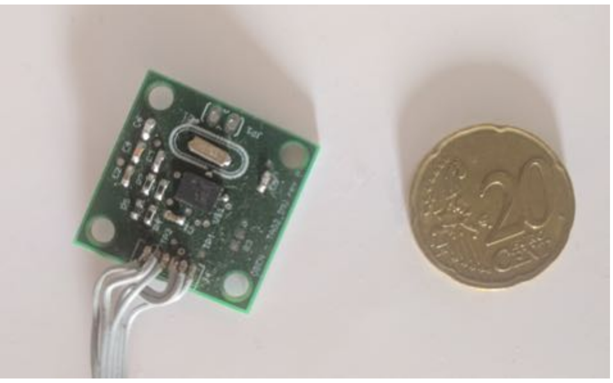


Figure 7-6 TAG IMU board

7.2.2 The demonstration system

Like shown in Figure 7-7 the system is composed by three entities that have to interact with each other:

- TAG board (onboard drone)
- Base Station (at ground)
- Remote Server

TAG board will acquire every second the information required. Through the RF link the information is sent to the base station that can be a PC on which it is running a software that captures data from the RF link and sends it through internet to a remote server.

The remote server reads the ID of the system and stores the data in database.

Always on the server side the operator can monitor in real-time the actual air traffic status through a web interface.

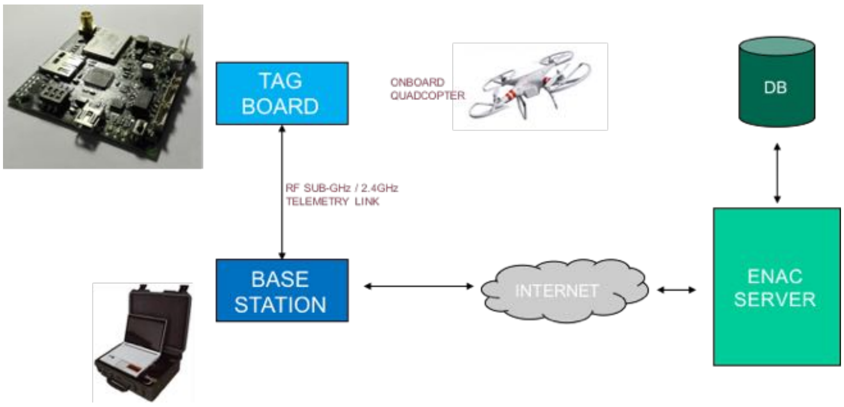


Figure 7-7 Working principle of the system

The database application is very simple, and based on MySQL and PHP, basically the BASE STATION once received the telemetry data from TAG board uploads it on ENAC SERVER by executing this PHP script:

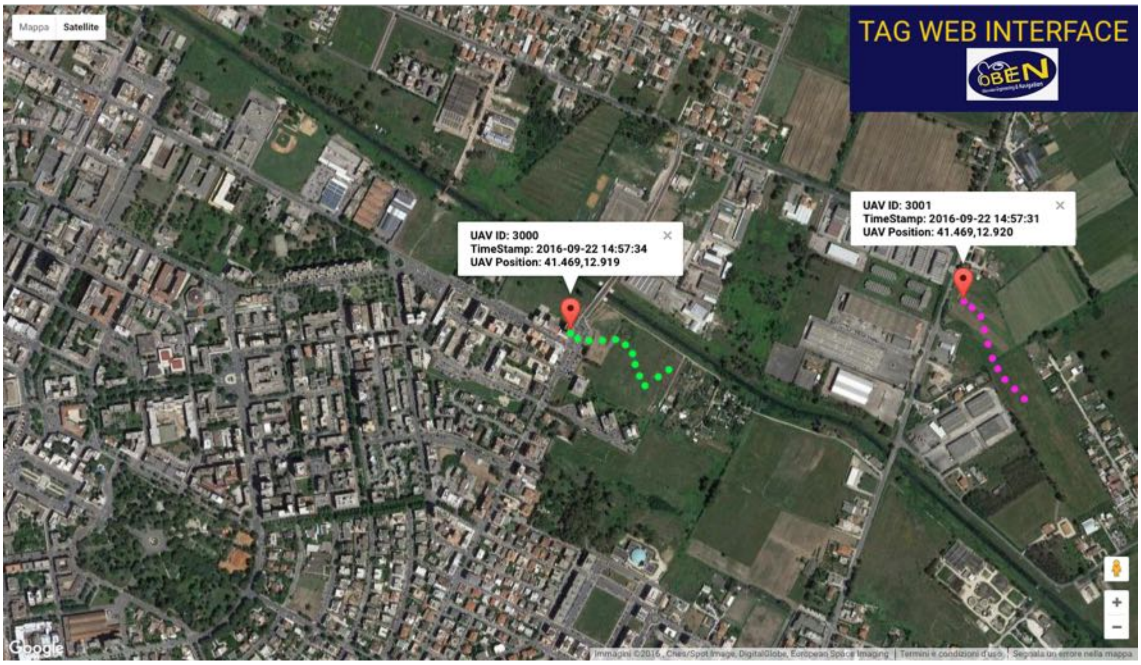
```
<?php
    // Connect to MySQL
    include("dbconnect.php");
    // Prepare the SQL statement
    $SQL = "INSERT INTO ENAC_SERVER_DB.drone (uavid, lat, lon, hMSL) VALUES
('".$_GET["uavid"]."',          '".$_GET["lat"]."',          '".$_GET["lon"]."',
'".$_GET["hMSL"]."')";
    // Execute SQL statement
    mysql_query($SQL);
    // Go to the review_data.php (optional)
    header("Location: review_data.php");
?>
```

Where the dbconnect.php is the connection/authentication part:

```
<?php
$MyUsername = "RPASuser1";  // enter your username for mysql
$MyPassword = "";  // enter your password for mysql
$MyHostname = "localhost";       // this is usually "localhost" unless your
database resides on a different server

$dbh = mysql_pconnect($MyHostname , $MyUsername, $MyPassword);
$selected = mysql_select_db("ENAC_SERVER_DB ",$dbh);
?>
```

The web interface is shown in Figure 7-8, the interface, based on a Google API, shows every RPAS acting in that area, shows its trail corresponding at every valid packet received by the transponder, the last position is shown by a bigger marker and a popup window contains the main information needed by the operator to track the RPAS.

In view of a broad diffusion of such application the communication part between every base station and the server could be implemented with protocols like MQTT that would allow for less overhead in the communication enabling lot more connections to the single server.


Mappa
Satellite
TAG WEB INTERFACE
Google
Immagini ©2016 , Cnes/Spot Image, DigitalGlobe, European Space Imaging | Termini e condizioni d'uso | Segnala un errore nella mappa

UAV ID: 3000
TimeStamp: 2016-09-22 14:57:34
UAV Position: 41.469,12.919

UAV ID: 3001
TimeStamp: 2016-09-22 14:57:31
UAV Position: 41.469,12.920

Figure 7-8 TAG Interface transponder demonstration

8 CONCLUSIONS

When I started this research project three years ago the autonomous guidance of drones was addressed by the research world just with huge devices occupying a lot of space and consuming a lot of energy. This was because every system was essentially based on SLAM and other resource-thirsty algorithms running on notebooks or bulky embedded PC. Today even COTS drones are equipped with Sense and Avoid features but are always far from being ready for autonomous guidance since they are typically slow in detection of sudden obstacles or have limitations in detecting tough obstacles like wires, nets or windows.

As of today the research is moving towards technologies based on the perception and very promising results are coming from Event Cameras, that join the information available from camera vision to the need of fast onboard computing. However, these approaches are at their very start and as of today they can’t represent a unique solution to the autonomous guidance problem. In this vision, the Multi-Sensor system proposed in this thesis is of interest because no sensor has the truth about the surrounding environment, only with fusion of the data coming from multiple sources it is thinkable to have a reliable vision of the surrounding environment that is as accurate as possible.

A further development of this doctorate project is to integrate the multi-sensor system developed in these years with an event camera visual odometry system and fuse the data coming from the two worlds in a single obstacle information, the system would be able to move accurately in the surrounding environment thanks to the camera system and then it would be able to avoid for example a window that a camera would not see at all.

Compared to when I started my PhD project in 2014, now in 2017 the computing power to do these tasks on a limited weight and size is starting to be available on large scale. The technology developed for modern smartphones provide that integration, moreover meanwhile the FPGA manufacturers moved towards integrated SoC solutions where an FPGA is always more often accompanied with an ARM high-end microprocessor. This kind of device could be the best candidate for an evolved Leonardo autopilot that includes both multi-sensor analysis of RADAR, LIDAR and SONAR and Visual Odometry.

However, for a safe deployment of an autonomous system this is not enough. It is also important to enforce communication security between RPAS and the ground station, at the moment of writing there is the need of a single protocol/channel that enables RPAS to share data with the ground station. Every manufacturer goes on its own path, but together with a shared airspace they share the RF spectrum too, so it is necessary to agree to a standard for RPAS communication, because in a RF spectrum that is day by day always busier with broadband communications it is fundamental that the vital link between a RPAS and its pilot must be reliable 100% of the time.

Finally, a full autonomous system must be able to enter a shared airspace and actively communicate its position and identification information, the approach proposed in this work goes in this direction, the limitation here is that it should be also provided with standard technologies of identification like ADS-B that it would enable other aircrafts (both manned and unmanned) to detect the RPAS.

Most likely the solution to this problem will come from projects like the SESAR "U-Space" that will be operational in the coming years and the whole community of RPAS manufacturers and operators will need to conform to the new standards.

9 ACKNOWLEDGEMENTS

I would like to thank all those who contributed to the realization of this research project, especially professor Balsi who mentored me, and he has always been a reliable guide during these three years.

I would like to thank the other members of the research team in which I was committed during these years: Eng. Salvatore Esposito and Eng. Paolo Fallavollita, always ready to help me with their experience.

Thanks also to Oben s.r.l., the spin-off of “Università di Sassari” with which I worked within the TAG project and the ENAC Call for Demonstration.

In the end thanks to my son and my wife who supported me on this journey, with patience and understanding.

10 APPENDIX

10.1 SOURCE CODES

10.1.1 Code for PPM signal generation

```
library ieee;

USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

entity ppm_encoder_us is
generic(NCHANS : integer :=8);
port(
clk: in std_logic;
clk1M : in std_logic;
rstn: in std_logic;
channel: in std_logic_vector((NCHANS-1) downto 0);
ppm_out : out std_logic
);
end ppm_encoder_us;


architecture behavior of ppm_encoder_us is
type fsm is (idle,S1,S2);
signal start_ppm_seq, channel0_q,clk1M_front,clk1M_q,clk1M_q1:
std_logic;
signal state : fsm;
signal wait_time, on_time : std_logic_vector (15 downto 0);
type Wordreg is array(0 to NCHANS - 1) of std_logic_vector(15 downto
0);
signal pwm_duty : Wordreg;
signal j : integer range 0 to 10;

component pwm_rx
port(
clk: in std_logic;
clk1M : in std_logic;
rstn: in std_logic;
pwm : in std_logic;
duty_out: out std_logic_vector(15 downto 0)
);
end component;
```

```
begin

clk1M_front <= clk1M_q1 and not clk1M_q;
start_ppm_seq <= channel0_q;
process(clk,rstn)
begin
      if rstn='0' then
            clk1M_q <= '0';
            clk1M_q1 <= '0';
            channel0_q <= '0';
            wait_time <= (others => '0');
            on_time <= (others => '0');
            state <= idle;
            ppm_out <= '1';
            j <= 0;

      elsif rising_edge(clk) then
            channel0_q <= channel(0);
            clk1M_q1 <= clk1M;
            clk1M_q <= clk1M_q1;
            case state is
            when idle =>
                  j <= 0;
                  if start_ppm_seq = '1' then
                        state <= S1;
                  else
                        state <= idle;
                        end if;
            when S1 =>
                  if j<NCHANS then
                  if pwm_duty(j) > conv_std_logic_vector(800,16) then
                        if pwm_duty(j) < conv_std_logic_vector(2000,16)
then
                                    wait_time <= pwm_duty(j);
                              else
                                    wait_time <=
conv_std_logic_vector(2000,16);
                                    end if;
                              else
                                    wait_time <=
conv_std_logic_vector(800,16);
                              end if;
                              ppm_out <= '0';
                              on_time <= (others => '0');
                              state <= S2;
                        else
                              ppm_out <= '1';
                              wait_time <= (others =>'0');
                              state <= idle;
                        end if;
                  when S2 =>
                        if clk1M_front = '1' then
                              wait_time <= wait_time - x"0001";
                              on_time <= on_time + x"0001";
                        end if;
                        if on_time > conv_std_logic_vector(400,16) then
-- 400uS
                              ppm_out <= '1';
                        else
                              ppm_out <= '0';
                        end if;
                        if wait_time = 0 then
                              j<= j + 1;
                              state <= S1;
                        else
                              state <= S2;
                        end if;
                  when others =>
                        null;
            end case;
      end if;
end process;

pwms_inst: for i in 0 to NCHANS - 1 generate

inst_pwm_rx: pwm_rx
port map(
      clk => clk,
      clk1M => clk1M,
      rstn => rstn,
      pwm => channel(i),
      duty_out => pwm_duty(i)
);
end generate;

end architecture;
```

10.1.2 Code for the FFT Windowing Block

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity winblock is
port(
            clk         : in std_logic;
            rstn        : in std_logic;

            --to adc_if
            adcready          : in std_logic;
            adcdata     : in std_logic_vector(11 downto 0);
            adcchan     : out std_logic;
            adcreq      : out std_logic;

            --to fft256
            fft_sop     : out std_logic;
            fft_eop     : out std_logic;
            fft_real    : out std_logic_vector(11 downto 0);
            fft_ready:  in std_logic;
            fft_valid: out std_logic;

            --from ext
            start       : in std_logic
            );
end winblock;

architecture behaviour of winblock is

component fifo256
      PORT
      (
            clock       : IN STD_LOGIC ;
            data        : IN STD_LOGIC_VECTOR (11 DOWNTO 0);
            rdreq       : IN STD_LOGIC ;
            sclr        : IN STD_LOGIC ;
            wrreq       : IN STD_LOGIC ;
            almost_full       : OUT STD_LOGIC ;
            empty       : OUT STD_LOGIC ;
            full        : OUT STD_LOGIC ;
            q           : OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
      );
END component;

component multiplier
      PORT
      (
            clock0            : IN STD_LOGIC  := '1';
            dataa_0           : IN STD_LOGIC_VECTOR (11 DOWNTO 0) :=
(OTHERS => '0');
            datab_0           : IN STD_LOGIC_VECTOR (11 DOWNTO 0) :=
(OTHERS => '0');
            result            : OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
      );
END component;

component hann_rom
      port(

            address           : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            clock       : IN STD_LOGIC  := '1';
            q           : OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
      );
END component;
signal start_int,rst: std_logic;
signal adcfifo_empty,adcfifo_rd,adcfifo_wr,adcfifo_almost_full :
std_logic;
signal adcfifo_q,rom_q: std_logic_vector (11 downto 0);
signal mult_res: std_logic_vector(23 downto 0);
signal rom_addr : unsigned(7 downto 0);
signal waitmult,fft_count : integer;
type fsm1 is (IDLE,REQ_ADC,WAIT_ADC);
type fsm2 is (IDLE,FFTSTART,FFTSEND,FFTEND);
signal state : fsm1;
signal state2 : fsm2;

      begin

      rst <= not rstn;
      start_int <= start;
      adcchan <= '0';

-- adc to fifo
process(clk,rstn)
begin
if rstn = '0' then
      adcfifo_wr <= '0';
      state <= IDLE;
      adcreq <= '0';
elsif rising_edge(clk) then
      case state is
      when IDLE =>
      adcreq <= '0';
      adcfifo_wr <= '0';
      if start_int = '1' then
            state <= REQ_ADC;
            adcreq <= '1';
      end if;
      when REQ_ADC =>
      adcreq <= '0';
      if adcfifo_almost_full = '1' then
            adcfifo_wr <= '0';
            state <= IDLE;
      else
            adcfifo_wr <= '0';
            state <= WAIT_ADC;
      end if;
      when WAIT_ADC =>
      if adcready = '1' then
            adcfifo_wr <= '1';
            state <= REQ_ADC;
            adcreq <= '1';
      end if;

      when others =>
            null;
      end case;
end if;
end process;

inst_fifo_adc :  fifo256
      PORT MAP
      (
            clock       => clk ,
            data        => adcdata,
            rdreq       => adcfifo_rd,
            sclr        => rst,
            wrreq       => adcfifo_wr,
            almost_full => adcfifo_almost_full ,
            empty       => adcfifo_empty,
            full        => open,
            q                 => adcfifo_q
      );


inst_hann_rom :  hann_rom
      port map(

            address           => std_logic_vector(rom_addr),
            clock       => clk,
            q           => rom_q
      );
inst_fixmult : multiplier
      PORT map
      (
            clock0            => clk,
            dataa_0           => rom_q,
            datab_0           => adcfifo_q,
            result      => mult_res
      );
```

```
fft_real <= mult_res(23 downto 12);

process(clk,rstn)
begin
if rstn = '0' then
      rom_addr <= (others => '0');
      state2 <= IDLE;
      adcfifo_rd <= '0';
      fft_valid <= '0';
      fft_eop <= '0';
      fft_sop <= '0';
      waitmult <= 0;
      fft_count <= 0;
elsif rising_edge(clk) then
      case state2 is
            when IDLE =>
                  fft_valid <= '0';
                  fft_eop <= '0';
                  fft_sop <= '0';
                  rom_addr <= (others =>'0');
                  if adcfifo_almost_full = '1' then
                        state2 <= FFTSTART;
                        waitmult <= 0;
                  end if;
            when FFTSTART =>
                  adcfifo_rd <= '1';
                  rom_addr <= rom_addr + 1;
                  if waitmult = 3 then
                        fft_sop <='1';
                        fft_valid <= '1';
                        fft_count <= 0;
                        state2 <= FFTSEND;
                        waitmult <= 0;
                  else
                        waitmult <= waitmult + 1;
                        fft_sop <= '0';
                        fft_valid <= '0';
                  end if;


            when FFTSEND =>

                  fft_sop <= '0';
                  if fft_count = 254 then
                        fft_count <= 0;
                        state2 <= FFTEND;
                        fft_eop <= '1';
                  else
                        fft_count <= fft_count + 1;
                        fft_eop <= '0';
                        rom_addr <= rom_addr + 1;
                  end if;

                  if adcfifo_empty = '0' then
                        adcfifo_rd <= '1';
                  else
                        adcfifo_rd <= '0';
                  end if;
            when FFTEND =>
                  state2 <= IDLE;
                  fft_eop <= '0';
                  fft_valid <= '0';
            when others =>
            null;
      end case;
end if;
end process;


end architecture;
```

10.1.3 Code for ALU block of the Kalman Processor

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity alu is
port(
            clk         : in std_logic;
            rstn        : in std_logic;

            IN1               : in std_logic_vector(31 downto 0);
            IN2               : in std_logic_vector(31 downto 0);
            RESULT            : out std_logic_vector(31 downto 0);

            OPER        : in std_logic_vector(1 downto 0);
            EN          : in std_logic;
            VALID       : out std_logic

            );
end alu;

architecture behaviour of alu is

signal add_sub_int,valid_int,en_int,en_q,en_front : std_logic;
signal t_oper,oper_cnt : integer;
signal oper_int : std_logic_vector (1 downto 0);
signal
a_mul,b_mul,a_sum,b_sum,a_div,b_div,a_sub,b_sub,res_mul,res_sum,res_di
v : std_logic_vector(31 downto 0);

component fp_sub
      PORT
      (
            add_sub           : IN STD_LOGIC ;
            clock       : IN STD_LOGIC ;
            dataa       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            datab       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            result            : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
      );
END component;

component fp_mul
      PORT
      (
            clock       : IN STD_LOGIC ;
            dataa       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            datab       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            result            : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
      );
END component;
```

```
component fp_div
      PORT
      (
            clock       : IN STD_LOGIC ;
            dataa       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            datab       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            result            : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
      );
END component;

begin

inst_fp_mul :  fp_mul
      PORT MAP
      (
            clock       => clk ,
            dataa       => a_mul,
            datab       => b_mul,
            result            => res_mul
      );

inst_fp_div :  fp_div
      PORT MAP
      (
            clock       => clk ,
            dataa       => a_div,
            datab       => b_div,
            result            => res_div

      );

inst_fp_sub :  fp_sub
      PORT MAP
      (
            add_sub           => add_sub_int,
            clock       => clk ,
            dataa       => a_sum,
            datab       => b_sum,
            result            => res_sum

      );

VALID <= valid_int and EN;
oper_int <= OPER;
en_front <= en_int and not en_q;
en_int <= EN;
process (clk,rstn) is
begin
      if rstn = '0' then
            a_mul <= (others => '0');
            b_mul <= (others => '0');
            a_sum <= (others => '0');
            b_sum <= (others => '0');
            a_sub <= (others => '0');
            b_sub <= (others => '0');
            a_mul <= (others => '0');
            a_div <= (others => '0');
            RESULT <= (others => '0');
            en_q <= '0';
            t_oper <= 0;
            valid_int <= '0';
            add_sub_int <= '0';
            oper_cnt <= 0;

      elsif rising_edge(clk) then

            en_q <= en_int;
             case oper_int is
                  when "00" =>--MUL
                        a_mul <= IN1;
                        b_mul <= IN2;
                        RESULT <= res_mul;
                        t_oper <= 10;
                  when "01" =>--SUM
                        a_sum <= IN1;
                        b_sum <= IN2;
                        RESULT <= res_sum;
                        add_sub_int <= '1';
                        t_oper <= 10;
                  when "10" =>--SUB
                        a_sum <= IN1;
                        b_sum <= IN2;
                        add_sub_int <= '0';
                        RESULT <= res_sum;
                        t_oper <= 10;
                  when "11" =>--DIV
                        a_div <= IN1;
                        b_div <= IN2;
                        RESULT <= res_div;
                        t_oper <= 14;
                   when others =>
                        null;
            end case;




            if en_int = '1' then
                  if en_front = '1' then
                        oper_cnt <= t_oper;
                  else
```

```
                        if(oper_cnt > 0) then
                              oper_cnt <= oper_cnt - 1;
                              valid_int <= '0';
                        else
                              valid_int <= '1';
                        end if;
                  end if;
            else
                  valid_int <= '0';
                  oper_cnt <= t_oper;
            end if;

      end if;
end process;



end architecture;
```

10.1.4 Code for Processing Unit block of the Kalman Processor

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity cpu is
generic(
      PROGRAM_LEN : integer := 13

      );
port(
            clk         : in std_logic;
            rstn        : in std_logic;

            START_in    : in std_logic;

            x0                : in std_logic_vector(31 downto 0);
            P0                : in std_logic_vector(31 downto 0);

            H                 : in std_logic_vector(2 downto 0);

            z1                : in std_logic_vector(31 downto 0);
            z2                : in std_logic_vector(31 downto 0);
            z3                : in std_logic_vector(31 downto 0);

            x1                : out std_logic_vector(31 downto 0);
            P1                : out std_logic_vector(31 downto 0);

            DONE        : out std_logic
);
end cpu;

architecture behaviour of cpu is

type ROM is array ( 0 to 13) of std_logic_vector(27 downto 0);

constant rom_mem : ROM :=(
  0 => x"0708090",--RAM(9) = RAM(7)+RAM(8)
  1 => x"0A06001",--H0 = H*x
  2 => x"030A0D2",--y = z - H0
  3 => x"0A09001",--PH' = H'*P
  4 => x"0000003",--HPH' = H * PH'
  5 => x"0000004",--S = HPH' + R
  6 => x"00001C5",--INVERSE
  7 => x"1013166", --MPY
  8 => x"1010007", --SCAL_MPY
  9 => x"0010138", --MPY KH
  10 => x"0D100D8",--MPY KY
  11 => x"0D060D0",--SUM X+KY
  12 => x"2E13139",--SUB 1 - KH
  13 => x"130913A"--MPY (1 - KH)*P
  );

type RAM is array (0 to 46) of std_logic_vector(31 downto 0);

signal ram_mem : RAM :=
(
      0     => x"00000000",--H1
      1     => x"00000000",--H2
      2     => x"00000000",--H3
      3     => x"00000000",--z1
      4     => x"00000000",--z2
      5     => x"00000000",--z3
      6     => x"00000000",--x
      7     => x"00000000",--P0
      8     => x"3727c5ac",--Q
      9     => x"00000000",--P
      10    => x"00000000",--H0_1  PH_1
      11  => x"00000000",--H0_2  PH_2
      12  => x"00000000",--H0_3  PH_3
      13  => x"00000000",--y_1   K*y
      14  => x"00000000",--y_2
      15  => x"00000000",--y_3
      16  => x"00000000",--HPH'_1_1  S_0_0     K_1
      17  => x"00000000",--HPH'_1_2  S_0_1     K_2
      18  => x"00000000",--HPH'_1_3  S_0_2     K_3
      19  => x"00000000",--HPH'_2_1  S_1_0     K*H
      20  => x"00000000",--HPH'_2_2  S_1_1
      21  => x"00000000",--HPH'_2_3  S_1_2
      22  => x"00000000",--HPH'_3_1  S_2_0
      23  => x"00000000",--HPH'_3_2  S_2_1
      24  => x"00000000",--HPH'_3_3  S_2_2
      25  => x"3eb851ec",--R_1 0.36
      26  => x"3b23d70a",--R_2 0.0025
      27  => x"3c23d70a",--R_3 0.01
      28  => x"00000000",--sminor_1
      29  => x"00000000",--sminor_2
      30  => x"00000000",--sminor_3
      31  => x"00000000",--sminor_4
      32  => x"00000000",--sminor_5
      33  => x"00000000",--sminor_6
      34  => x"00000000",--sminor_7
      35  => x"00000000",--sminor_8
      36  => x"00000000",--sminor_9
      37  => x"00000000",--sminor_10
      38  => x"00000000",--sminor_11
      39  => x"00000000",--sminor_12
      40  => x"00000000",--sminor_13
      41  => x"00000000",--sminor_14     det(S)
      42  => x"00000000",--sminor_15
      43  => x"00000000",--sminor_16
      44  => x"00000000",--sminor_17
      45  => x"00000000",--sminor_18
      46  => x"3f800000" --1
      );
type MINORS_t is array(0 to 17,0 to 1) of integer;
constant minors : MINORS_t :=(
(20,24),--S_1_1*S_2_2   +
(21,23),--S_1_2*S_2_1

(21,22),--S_1_2*S_2_0   -
(19,24),--S_1_0*S_2_2

(19,23),--S_1_0*S_2_1   +
(20,22),--S_1_1*S_2_0

(18,23),--S_0_2*S_2_1   -
(17,24),--S_0_1*S_2_2

(16,24),--S_0_0*S_2_2   +
(18,22),--S_0_2*S_2_0

(17,22),--S_0_1*S_2_0   -
(16,23),--S_0_0*S_2_1

(17,21),--S_0_1*S_1_2   +
(18,20),--S_0_2*S_1_1

(18,19),--S_0_2*S_1_0   -
(16,21),--S_0_0*S_1_2

(16,20),--S_0_0*S_1_1   +
(17,19)--S_0_1*S_1_2

);




type fsm is (IDLE,START,LOAD,EXEC1,EXEC2,
                  EXEC3,
                  EXEC4,
                  EXEC5,EXEC5_1,EXEC5_2,EXEC5_3,EXEC5_4,
                  EXEC6,EXEC6_1,EXEC6_2,EXEC6_3,EXEC6_4,EXEC6_5,
                  EXEC7,EXEC7_1,EXEC7_2,
                  EXEC8,
                  EXEC9,EXEC9_1,EXEC9_2,
                  PC_FWD);
signal state : fsm;
signal pc,op1,op2,op3,temp : integer;
```

```
component alu
port(
            clk         : in std_logic;
            rstn        : in std_logic;

            IN1               : in std_logic_vector(31 downto 0);
            IN2               : in std_logic_vector(31 downto 0);
            RESULT            : out std_logic_vector(31 downto 0);

            OPER        : in std_logic_vector(1 downto 0);
            EN          : in std_logic;
            VALID       : out std_logic

            );
end component;

signal alu_IN1,alu_IN2,alu_RESULT: std_logic_vector(31 downto 0);
signal alu_OPER: std_logic_vector(1 downto 0);
signal alu_EN, alu_VALID, DONE_int: std_logic;
begin

DONE <= DONE_int;

inst_alu: alu
port map(
      clk   =>    clk,
      rstn  =>    rstn,
      IN1               => alu_IN1,
      IN2               => alu_IN2,
      RESULT            => alu_RESULT,

      OPER        => alu_OPER,
      EN                => alu_EN,
      VALID       => alu_VALID

      );

process (clk,rstn)
begin
      if rstn = '0' then
            DONE_int <= '0';
            pc <= 0;
            op1 <= 0;
            op2 <= 0;
            op3 <= 0;
            alu_EN <= '0';
            alu_OPER <="00";
            alu_IN1 <= (others => '0');
            alu_IN2 <= (others => '0');
      elsif rising_edge(clk) then
```

```
            case state is
            when IDLE =>
                  if START_in = '1' then
                        DONE_int <= '0';
                        pc <= 0;
                        state <= START;
                  end if;
            when START =>
                  if H(0) = '0' then
                        ram_mem(0) <= x"00000000";
                  else
                        ram_mem(0) <= x"3f800000";
                  end if;

                  if H(1) = '0' then
                        ram_mem(1) <= (others => '0');
                  else
                        ram_mem(1) <= x"3f800000";
                  end if;

                  if H(2) = '0' then
                        ram_mem(2) <= (others => '0');
                  else
                        ram_mem(2) <= x"3f800000";
                  end if;

                  ram_mem(3) <= z1;
                  ram_mem(4) <= z2;
                  ram_mem(5) <= z3;
                  ram_mem(6) <= x0;
                  ram_mem(7) <= P0;
                  state <= LOAD;
            when LOAD =>
                  op1 <= to_integer(unsigned(rom_mem(pc)(27 downto
20)));
                  op2 <= to_integer(unsigned(rom_mem(pc)(19 downto
12)));
                  op3 <= to_integer(unsigned(rom_mem(pc)(11 downto
4)));

                  case rom_mem(pc)(3 downto 0) is
                  when "0000" => --element sum
                        alu_OPER <= "01";--sum
                        state <= EXEC1;
                  when "0001" => --scal mpy
                        alu_OPER <= "00";
                        state <= EXEC2;
                  when "0010" => --sub
                        alu_OPER <= "10";--sub
                        temp <= 0;
                        state <= EXEC3;
```

```
      when "0011" => --mpy 3x1* 1x3
            alu_OPER <= "00";
            state <= EXEC4;
      when "0100" => --sum 3x3 + 3x3
            alu_OPER <= "01";--sum
            state <= EXEC5;
      when "0101" => --matrix_inv
            alu_OPER <= "00";
            state <= EXEC6;
      when "0110" => --mpy 1x3 * 3x3
            state <= EXEC7;
            --op1 <= 16;
            --op2 <= 19;
            --op3 <= 22;
            alu_OPER <= "01";--sum
      when "0111" => --scal mpy
            state <= EXEC8;
            alu_OPER <= "00";--mpy
            --op1 <= 16;
            --op2 <= 16;
      when "1000" => -- mpy 1x3 3x1
            state <= EXEC9;
            alu_OPER <= "00";--mul
            temp <= 0;
      when "1001" =>
            alu_OPER <= "10";--sub
            state <= EXEC1;
      when "1010" =>
            alu_OPER <= "00";--mul
            state <= EXEC1;
      when others =>
            null;
      end case;

when EXEC1 =>--simple operation [1] + [1]
      alu_IN1 <= ram_mem(op1);
      alu_IN2 <= ram_mem(op2);

      alu_EN <= '1';
      if (alu_VALID = '1') then
            ram_mem(op3) <= alu_RESULT;
            state <= PC_FWD;
      end if;
when EXEC2 =>--scalar mpy [3x1]*[1]
      if (ram_mem(0)(29) = '1') then
            ram_mem(op1) <= ram_mem(op2);
      else
            ram_mem(op1) <= x"00000000";
      end if;

      if (ram_mem(1)(29) = '1') then
```

```
                ram_mem(op1 + 1) <= ram_mem(op2);
          else
                ram_mem(op1 + 1) <= x"00000000";
          end if;

          if (ram_mem(2)(29) = '1') then
                ram_mem(op1 + 2) <= ram_mem(op2);
          else
                ram_mem(op1 + 2) <= x"00000000";
          end if;
          state <= PC_FWD;

    when EXEC3 =>
          alu_IN1 <= ram_mem(op1);
          alu_IN2 <= ram_mem(op2);
          op1 <= op1 + 1;
          op2 <= op2 + 1;
          if alu_VALID = '1' then
                if temp < 3 then
                      ram_mem(op3) <= alu_RESULT;
                      op3 <= op3 + 1;
                      temp <= temp + 1;
                else
                      state <= PC_FWD;
                end if;
          else
                alu_EN <= '1';
          end if;

    when EXEC4 =>

          if (ram_mem(0)(29) = '1') then
                ram_mem(16) <= ram_mem(10);
                ram_mem(17) <= ram_mem(10);
                ram_mem(18) <= ram_mem(10);
          else
                ram_mem(16) <= x"00000000";
                ram_mem(17) <= x"00000000";
                ram_mem(18) <= x"00000000";
          end if;

          if (ram_mem(1)(29) = '1') then
                ram_mem(19) <= ram_mem(11);
                ram_mem(20) <= ram_mem(11);
                ram_mem(21) <= ram_mem(11);
          else
                ram_mem(19) <= x"00000000";
                ram_mem(20) <= x"00000000";
                ram_mem(21) <= x"00000000";
          end if;
```

```
      if (ram_mem(2)(29) = '1') then
            ram_mem(22) <= ram_mem(12);
            ram_mem(23) <= ram_mem(12);
            ram_mem(24) <= ram_mem(12);
      else
            ram_mem(22) <= x"00000000";
            ram_mem(23) <= x"00000000";
            ram_mem(24) <= x"00000000";
      end if;

      state <= PC_FWD;
when EXEC5 =>
      alu_IN1 <= ram_mem(16);
      alu_IN2 <= ram_mem(25);
      alu_EN <= '1';
      state <= EXEC5_1;
when EXEC5_1 =>
      alu_IN1 <= ram_mem(20);
      alu_IN2 <= ram_mem(26);
      state <= EXEC5_2;
when EXEC5_2 =>
      alu_IN1 <= ram_mem(24);
      alu_IN2 <= ram_mem(27);
      if (alu_VALID = '1') then
            ram_mem(16) <= alu_RESULT;
            state <= EXEC5_3;
      end if;
when EXEC5_3 =>
      ram_mem(20) <= alu_RESULT;
      state <= EXEC5_4;
when EXEC5_4 =>
      ram_mem(24) <= alu_RESULT;
      state <= PC_FWD;
when EXEC6 =>
      alu_IN1 <= ram_mem(minors(op1,0)); --S_2_2
      alu_IN2 <= ram_mem(minors(op1,1)); --S_3_3
      if(op1 < 17) then
            op1 <= op1 + 1;
      end if;
      alu_EN <= '1';
      if(alu_VALID = '1') then
            if(op3 < 46) then
                  ram_mem(op3) <= alu_RESULT;
                  op3 <= op3 + 1;
            else
                  state <= EXEC6_1;
                  alu_EN <= '0';
                  op2 <= 28;
                  op3 <= 28;
                  alu_OPER <= "10";--sub
            end if;
```

```
                  end if;
            when EXEC6_1 =>
                  alu_IN1 <= ram_mem(op2);
                  alu_IN2 <= ram_mem(op2 + 1);
                  alu_EN <= '1';
                  if(op2 < 44) then
                        op2 <= op2 + 2;
                  end if;
                  if(alu_VALID = '1') then
                        if(op3 < 37) then
                              ram_mem(op3) <= alu_RESULT;--store minors
from 28 to 37
                              op3 <= op3 + 1;
                        else
                              state <= EXEC6_2;
                              alu_OPER <= "00";--mul
                              alu_EN <= '0';
                              op3 <= 28;
                              op1 <= 16;
                        end if;
                  else
                  end if;
            when EXEC6_2 =>
                  alu_IN1 <= ram_mem(op3);
                  alu_IN2 <= ram_mem(op1);
                  alu_EN <= '1';
                  if(op3 < 31) then
                        op3 <= op3 + 1;
                        op1 <= op1 + 1;
                        op2 <= 37;
                  else
                        if alu_VALID = '1' then
                              if(op2 < 40) then
                                    ram_mem(op2) <= alu_RESULT;
                                    op2 <= op2 + 1;
                              else
                                    state <= EXEC6_3;
                                    alu_EN <= '0';
                                    alu_OPER <= "01";--sum
                              end if;
                        end if;
                  end if;
            when EXEC6_3 =>
                  alu_IN1 <= ram_mem(37);
                  alu_IN2 <= ram_mem(38);
                  alu_EN <= '1';
                  if(alu_VALID = '1') then
                        ram_mem(40) <= alu_RESULT;
                        state <= EXEC6_4;
                        alu_EN <= '0';
                  end if;
```

```
when EXEC6_4 =>
      alu_EN <= '1';
      alu_IN1 <= ram_mem(40);
      alu_IN2 <= ram_mem(39);
      if alu_VALID = '1' then
            op3 <= 28;
            op2 <= 16;
            ram_mem(41) <= alu_RESULT;
            state <= EXEC6_5;
            alu_EN <= '0';
            alu_OPER <= "11";--div
      end if;
when EXEC6_5 =>
      alu_EN <= '1';
      alu_IN1 <= ram_mem(op3);
      alu_IN2 <= ram_mem(41);
      if op3 < 37 then
            op3 <= op3 + 1;
      end if;
      if alu_VALID = '1' then
            if op2 < 25 then
                  op2 <= op2 + 1;
                  ram_mem(op2) <= alu_RESULT;
            else
                  state <= PC_FWD;
            end if;
      end if;
when EXEC7 =>
      if ram_mem(0)(29) = '0' then
            ram_mem(op1) <= x"00000000";--16
            ram_mem(op1+1) <= x"00000000";
            ram_mem(op1+2) <= x"00000000";
      end if;
      if ram_mem(1)(29) = '0' then
            ram_mem(op2) <= x"00000000";--19
            ram_mem(op2+1) <= x"00000000";
            ram_mem(op2+2) <= x"00000000";
      end if;
      if ram_mem(2)(29) = '0' then
            ram_mem(op3) <= x"00000000";--22
            ram_mem(op3+1) <= x"00000000";
            ram_mem(op3+2) <= x"00000000";
      end if;
      state <= EXEC7_1;
when EXEC7_1=>
      alu_IN1 <= ram_mem(op1);
      alu_IN2 <= ram_mem(op2);
      if alu_VALID = '1' then
            ram_mem(op1) <= alu_RESULT;
            state <= EXEC7_2;
            alu_EN <= '0';
      else
            alu_EN <= '1';
      end if;
when EXEC7_2=>
      alu_IN1 <= ram_mem(op1);
      alu_IN2 <= ram_mem(op3);
      if alu_VALID = '1' then
            alu_EN <= '0';
            ram_mem(op1) <= alu_RESULT;
            if(op1 < 18) then
                  op1 <= op1 + 1;
                  op2 <= op2 + 1;
                  op3 <= op3 + 1;
                  state <= EXEC7_1;
            else
                  state <= PC_FWD;
            end if;
      else
            alu_EN <= '1';
      end if;
when EXEC8 =>--calcolo K = P * H'INVS
      alu_EN <= '1';
      alu_IN1 <= ram_mem(9);--P
      alu_IN2 <= ram_mem(op1);
      op1 <= op1 + 1;
      if alu_VALID = '1' then
            if op2 < 19 then
                  ram_mem(op2) <= alu_RESULT;
                  op2 <=op2 + 1;
            else
                  state <= PC_FWD;
            end if;
      end if;
when EXEC9 =>
      alu_IN1 <= ram_mem(op1);
      alu_IN2 <= ram_mem(op2);
      op1 <= op1 + 1;
      op2 <= op2 + 1;
      if alu_VALID = '1' then
            if temp < 3 then
                  ram_mem(op3) <= alu_RESULT;
                  op3 <= op3 + 1;
                  temp <= temp + 1;
            else
                  op3 <= op3 - 3;
                  alu_EN <= '0';
                  alu_OPER <= "01";--sum
                  state <= EXEC9_1;
            end if;
      else
            alu_EN <= '1';
```

```
                  end if;

            when EXEC9_1 =>
                  alu_IN1 <= ram_mem(op3);
                  alu_IN2 <= ram_mem(op3 + 1);
                  if alu_VALID = '1' then
                        ram_mem(op3) <= alu_RESULT;
                        state <= EXEC9_2;
                        alu_EN <= '0';
                  else
                        alu_EN <= '1';
                  end if;
            when EXEC9_2 =>
                  alu_EN <= '1';
                  alu_IN1 <= ram_mem(op3);
                  alu_IN2 <= ram_mem(op3+2);
                  if alu_VALID = '1' then
                        ram_mem(op3) <= alu_RESULT;
                        state <= PC_FWD;
                  end if;
            when PC_FWD =>
                  if(pc < PROGRAM_LEN) then
                        pc <= pc + 1;
                        state <= LOAD;
                  else
                        state <= IDLE;
                        x1 <= ram_mem(13);
                        P1 <= ram_mem(19);
                        DONE_int <= '1';
                  end if;
                  alu_EN <= '0';
            when others =>
                  null;
            end case;

      end if;
end process;


end architecture;
```

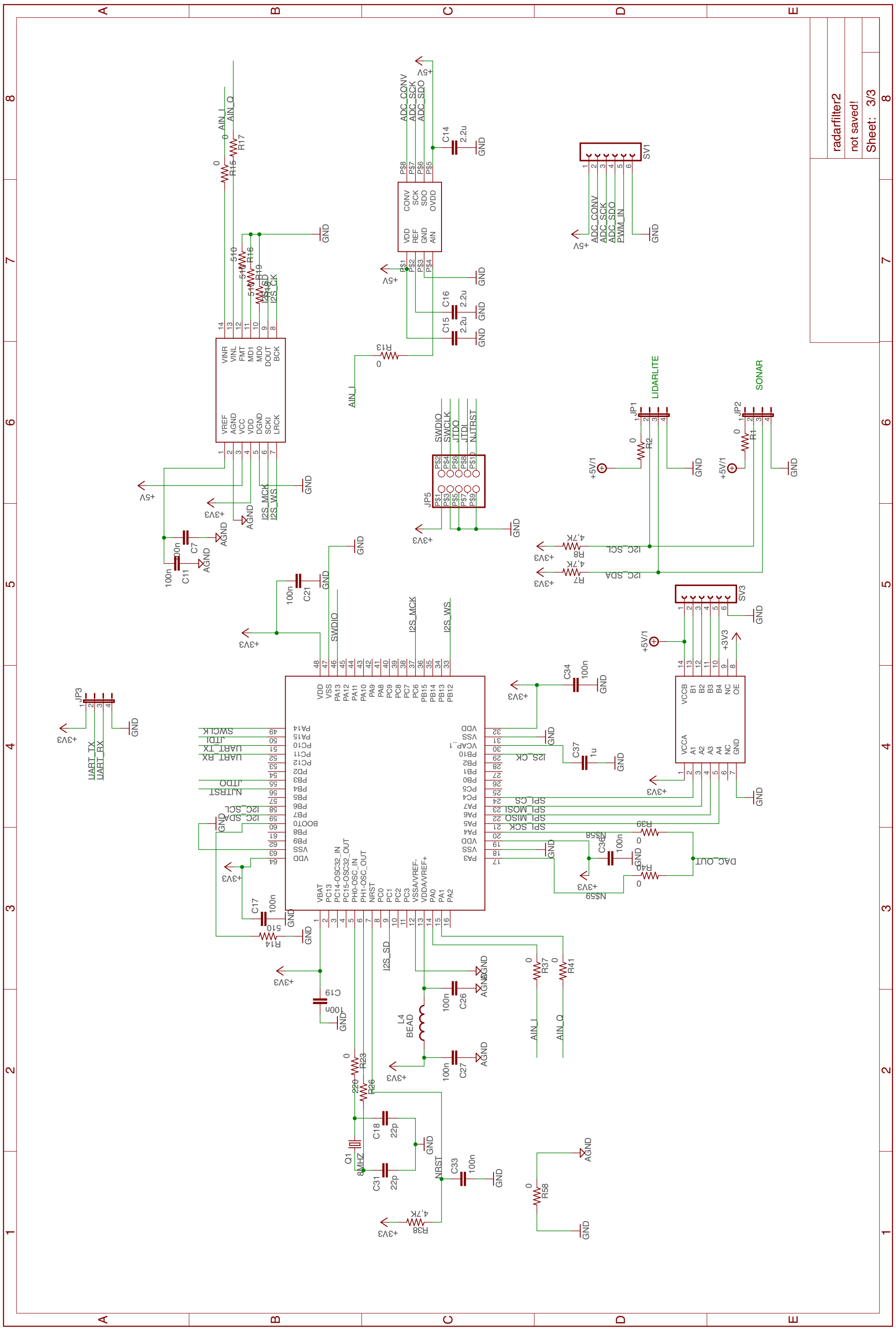
10.1.5 Matrix calculation C library

```
#define idx(i,j,cols) (i * cols + j)

void initMatrix(MATRIX * A,int r,int c)
{
      A->M = (double *)calloc(r * c, sizeof(double));
      A->r = r;
      A->c = c;
}

void Matrix_MPY(MATRIX * A,MATRIX * B,MATRIX * C)
{
      int i,j,k;
      memset(C->M,0,C->r * C->c * sizeof(double));
      for(i = 0; i < A->r;i++)
            for(j=0;j< B->c;j++)
                  for(k=0;k<B->r;k++)
                  {
                        C->M[i*B->c + j]+=A->M[i*A->c + k] * B->M[k*B-
>c + j];
                  }
}
void Matrix_SCALMPY(MATRIX * A, MATRIX *B,double scal)
{
      int i,j;
      for(i = 0; i < A->r;i++)
                  for(j=0;j< A->c;j++)
                        FillMatrix(B,i,j,elem(A,i,j) * scal);
}
void Matrix_TRANSP(MATRIX * A, MATRIX *B)
{
int i,j;
            for(i = 0; i < A->r;i++)
                  for(j=0;j< A->c;j++)
                        FillMatrix(B,j,i,elem(A,i,j));
}
void Matrix_SUM(MATRIX * A,MATRIX * B,MATRIX * C)
{
      int i,j;
            for(i = 0; i < A->r;i++)
                  for(j=0;j< A->c;j++)
                  {
                        C->M[i*A->c + j]=A->M[i*A->c + j] + B->M[i*B->c
+ j];
                  }
}

void Matrix_SUB(MATRIX * A,MATRIX * B,MATRIX * C)
{
      int i,j;
            for(i = 0; i < A->r;i++)
                  for(j=0;j< A->c;j++)
                  {
                        C->M[i*A->c + j]=A->M[i*A->c + j] - B->M[i*B->c
+ j];
                  }
}

double elem(MATRIX * A,int r,int c)
{
      return (A->M[r * A->c + c]);
}

void FillMatrix(MATRIX * A,int r, int c,double value)
{
      A->M[r*A->c + c] = value;
}

double Matrix_DET(MATRIX *A,int ord)
{
      double det = 0;

      if(ord == 1) det = elem(A,0,0);
      else if (ord == 2)
      {
            det = elem(A,0,0)*elem(A,1,1)-elem(A,0,1)*elem(A,1,0);
            /* code */
      }
      else if(ord == 3)
            {
                  det =
                  elem(A,0,0)*elem(A,1,1)*elem(A,2,2) +
               elem(A,0,1)*elem(A,1,2)*elem(A,2,0) +
               elem(A,0,2)*elem(A,1,0)*elem(A,2,1) -
               elem(A,2,0)*elem(A,1,1)*elem(A,0,2) -
               elem(A,2,1)*elem(A,1,2)*elem(A,0,0) -
               elem(A,1,0)*elem(A,0,1)*elem(A,2,2);
            }
      return det;
}
void Matrix_Print(MATRIX *A)
{
      int i,j;
      printf("\r\n");
      for(i=0;i<A->r;i++)
      {
            printf("|");
            for(j=0;j<A->c;j++)
            {
                  printf("%3.3f ",elem(A,i,j));
```

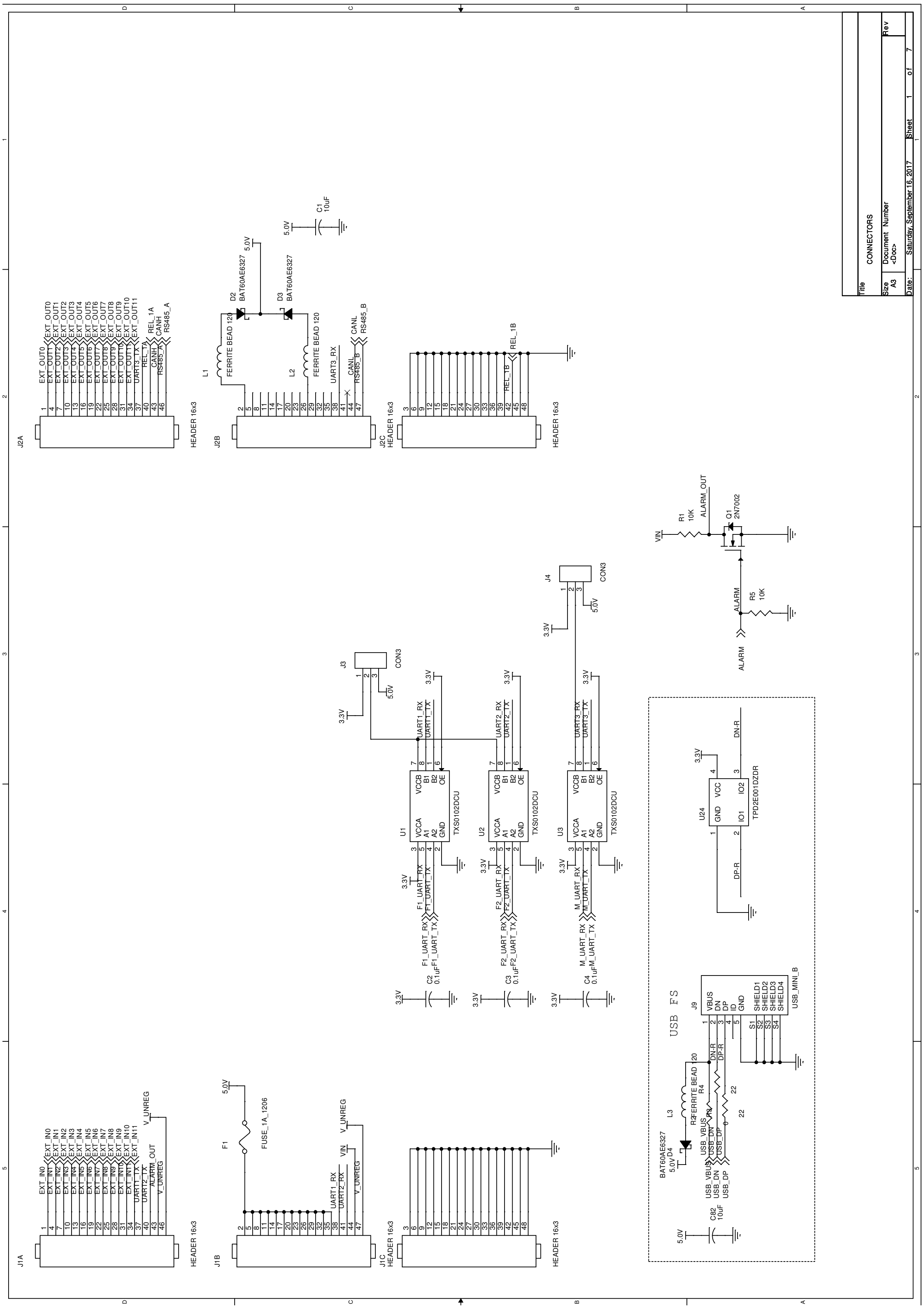
```
                }
                printf("|\r\n");
        }
}
void Matrix_INV3(MATRIX * A,MATRIX * INVA)
{
        double det = Matrix_DET(A,3);
        /*
        0,0 0,1 0,2
        1,0 1,1 1,2
        2,0 2,1 2,2*/
        FillMatrix(INVA,0,0,+(elem(A,1,1)*elem(A,2,2)-
elem(A,1,2)*elem(A,2,1))/det);//0,0
        FillMatrix(INVA,1,0,-(elem(A,1,0)*elem(A,2,2)-
elem(A,1,2)*elem(A,2,0))/det);//0,1
        FillMatrix(INVA,2,0,+(elem(A,1,0)*elem(A,2,1)-
elem(A,1,1)*elem(A,2,0))/det);//0,2
        FillMatrix(INVA,0,1,-(elem(A,0,1)*elem(A,2,2)-
elem(A,0,2)*elem(A,2,1))/det);//1,0
        FillMatrix(INVA,1,1,+(elem(A,0,0)*elem(A,2,2)-
elem(A,0,2)*elem(A,2,0))/det);//1,1
        FillMatrix(INVA,2,1,-(elem(A,0,0)*elem(A,2,1)-
elem(A,0,1)*elem(A,2,0))/det);//1,2
        FillMatrix(INVA,0,2,+(elem(A,0,1)*elem(A,1,2)-
elem(A,0,2)*elem(A,1,1))/det);//2,0
        FillMatrix(INVA,1,2,-(elem(A,0,0)*elem(A,1,2)-
elem(A,0,2)*elem(A,1,0))/det);//2,1
        FillMatrix(INVA,2,2,+(elem(A,0,0)*elem(A,1,1)-
elem(A,0,1)*elem(A,1,0))/det);//2,2

}

void deleteMatrix(MATRIX * A)
{
        free(A->M);
}
```

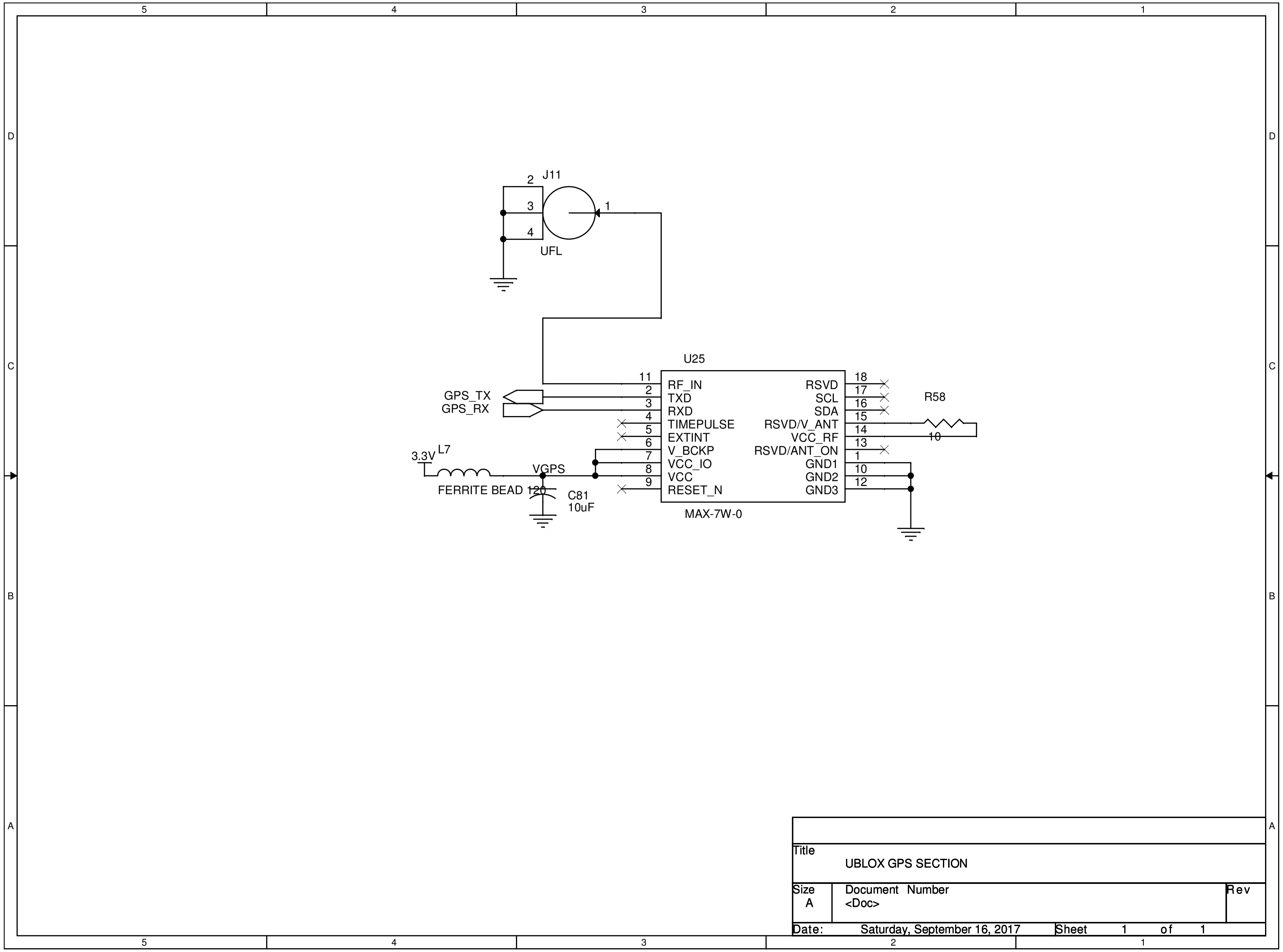
10.2 SCHEMATICS

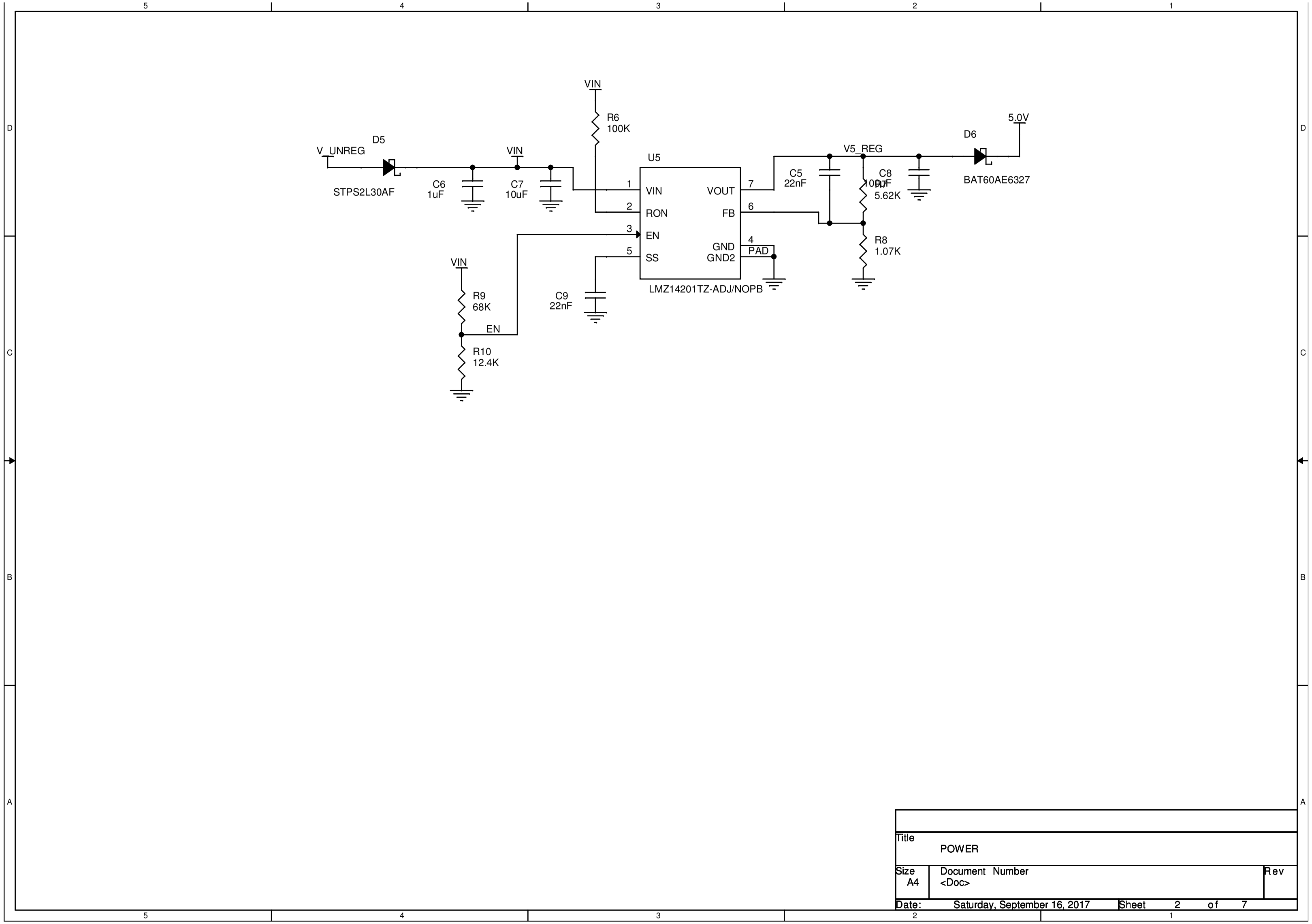
10.2.1 Digital section of DSP Multi-sensor system

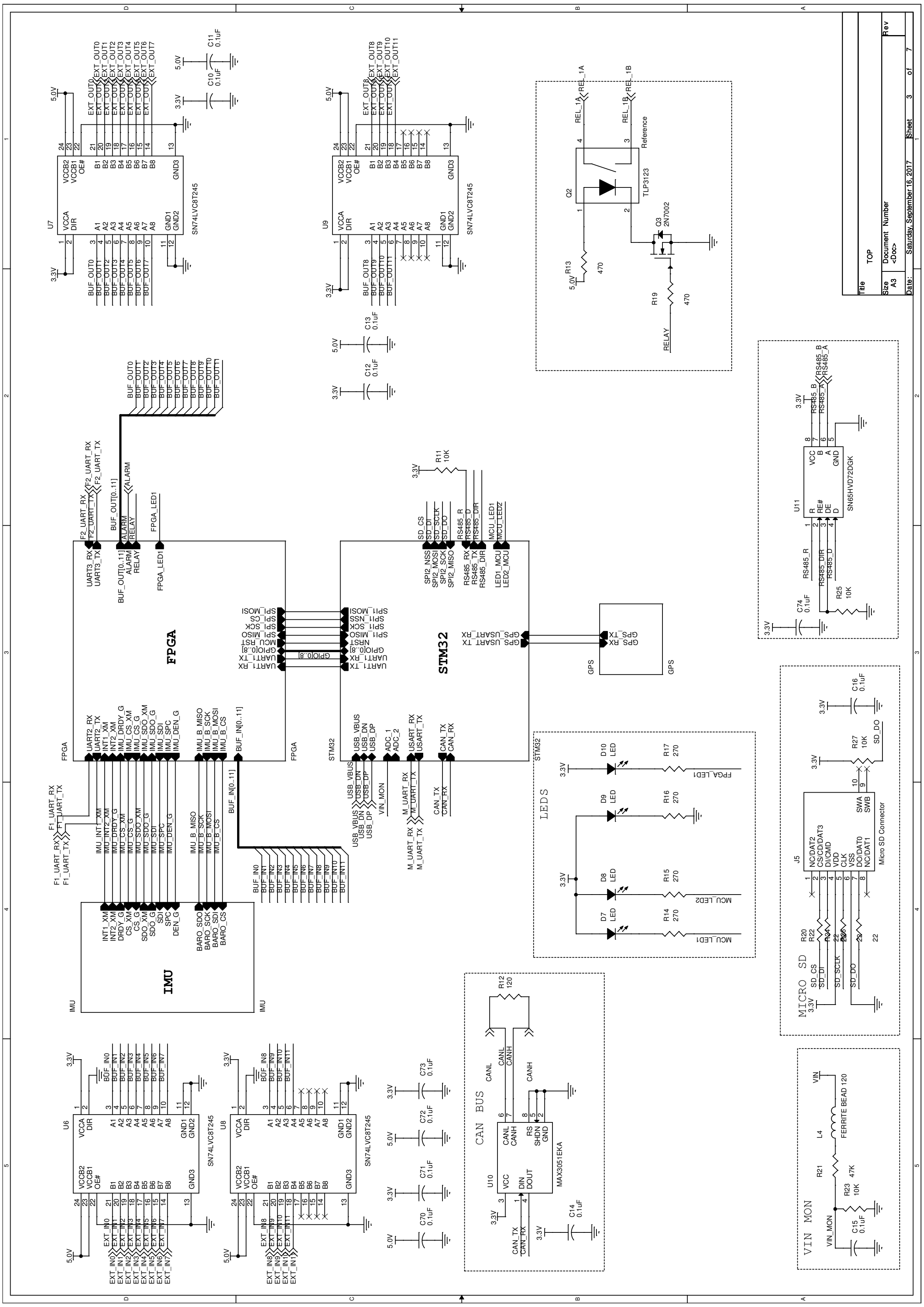


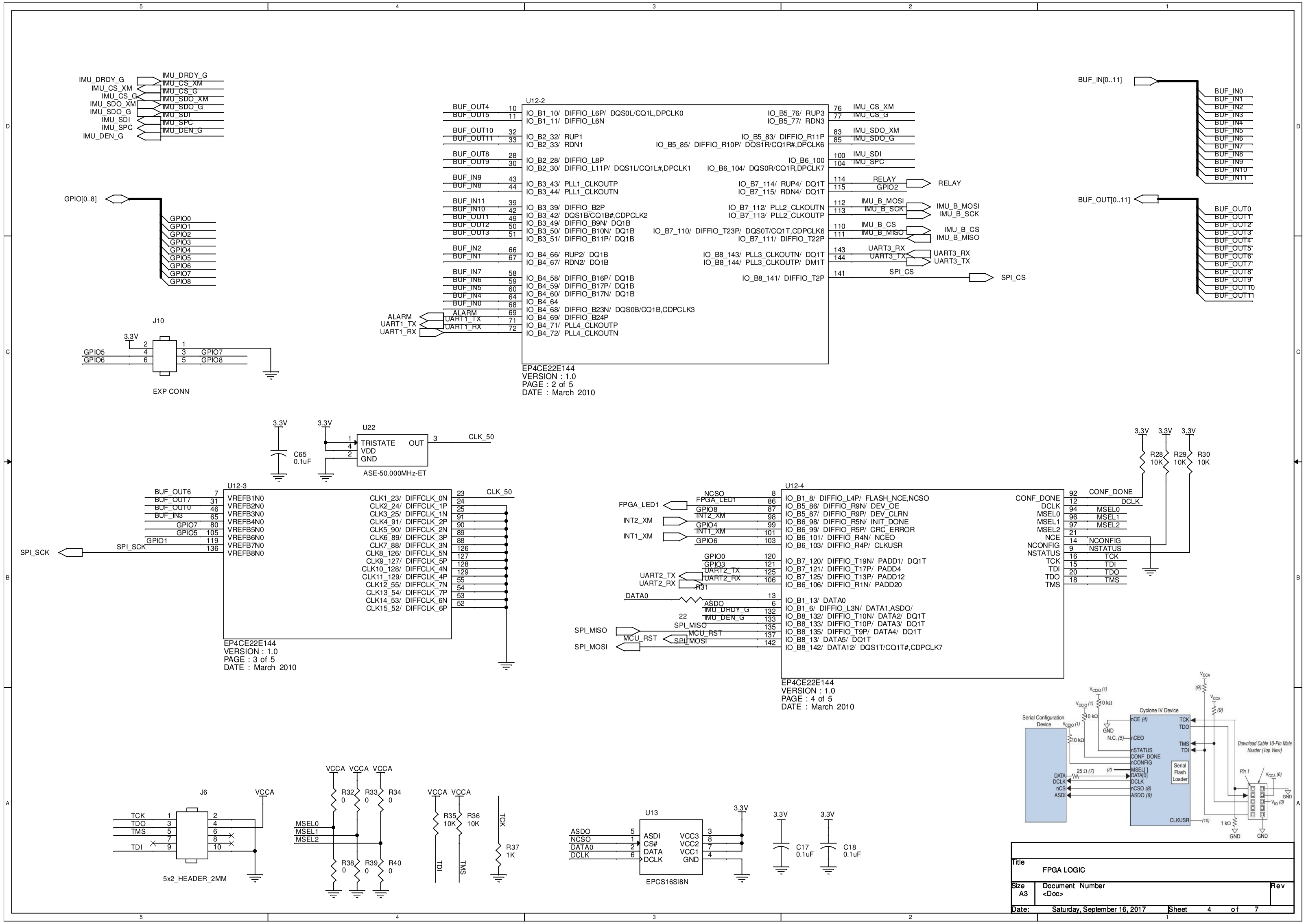
10.2.2 Leonardo autopilot schematic

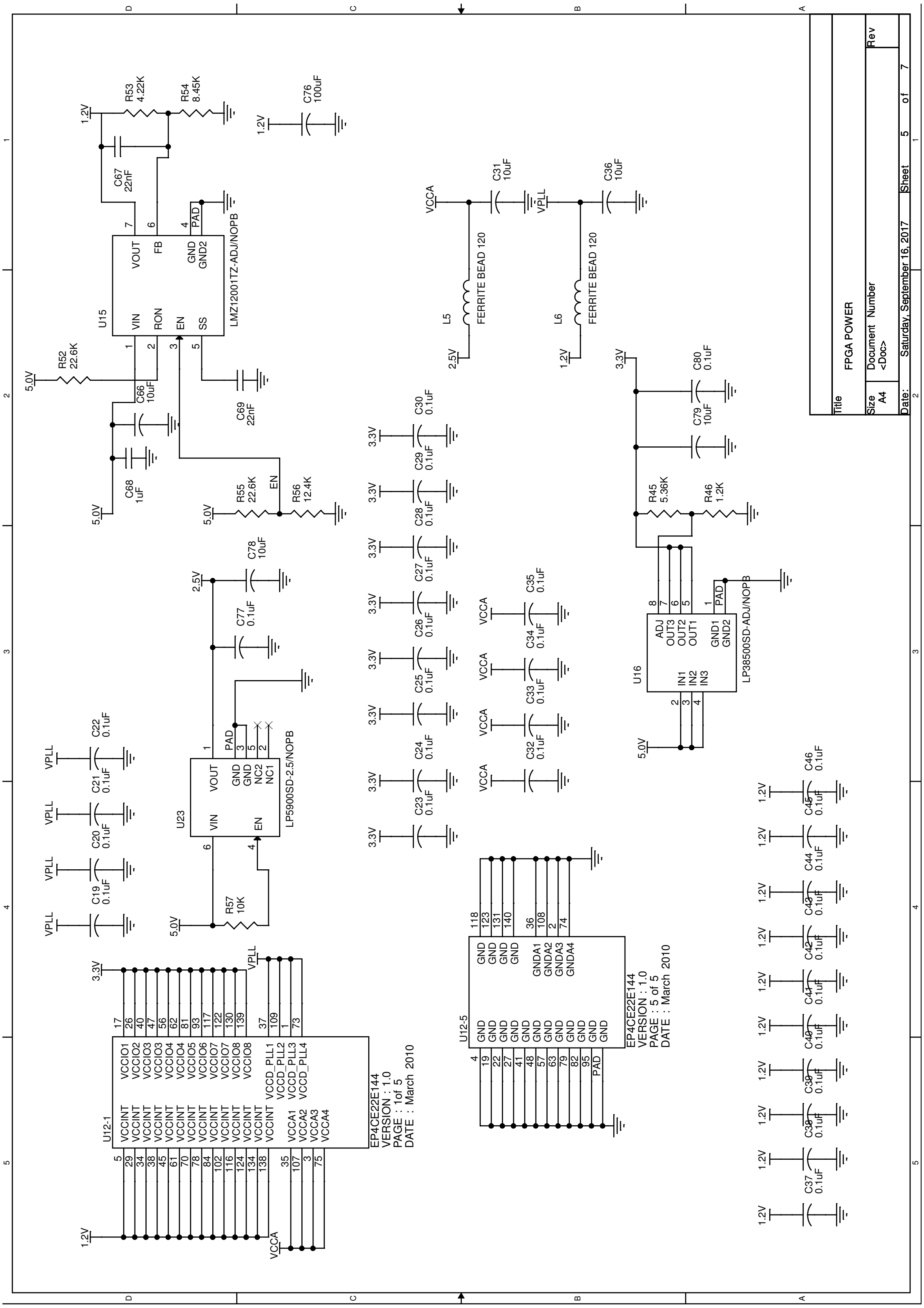


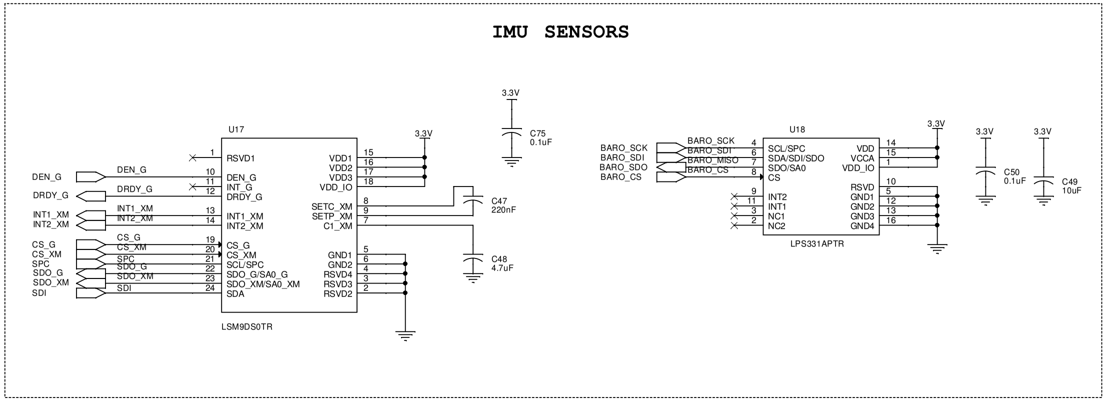

J11
2
3
4
1
UFL
U25
11
2
3
4
5
6
7
8
9
RF_IN
TXD
RXD
TIMEPULSE
EXTINT
V_BCKP
VCC_IO
VCC
RESET_N
RSVD
SCL
SDA
RSVD/V_ANT
VCC_RF
RSVD/ANT_ON
GND1
GND2
GND3
18
17
16
15
14
13
1
10
12
MAX-7W-0
R58
10
GPS_TX
GPS_RX
3.3V
L7
FERRITE BEAD 120
VGPS
C81
10uF
5
4
3
2
1
D
C
B
A
Title
UBLOX GPS SECTION
Size
A
Document Number
<Doc>
Rev
Date: Saturday, September 16, 2017
Sheet 1 of 1


V_UNREG
D5
STPS2L30AF
C6
1uF
VIN
C7
10uF
VIN
R6
100K
U5
1
VIN
2
RON
3
EN
5
SS
VOUT
7
FB
6
GND
GND2
4
PAD
LMZ14201TZ-ADJ/NOPB
C9
22nF
VIN
R9
68K
EN
R10
12.4K
C5
22nF
V5_REG
C8
10uF
5.62K
R8
1.07K
D6
BAT60AE6327
5.0V
Title
POWER
Size
A4
Document Number
<Doc>
Rev
Date:
Saturday, September 16, 2017
Sheet
2
of
7
5
4
3
2
1
D
C
B
A

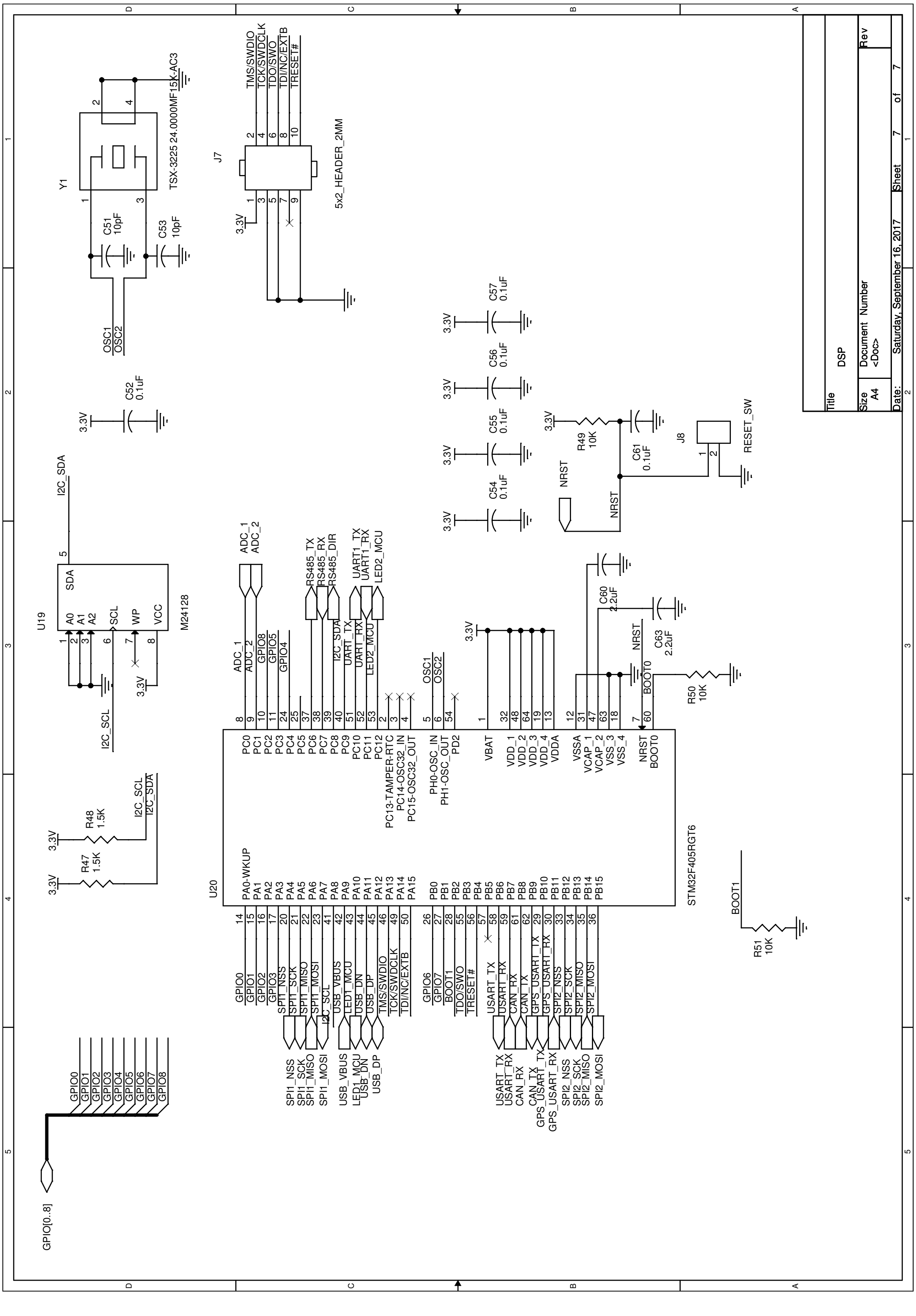

Title TOP
Size A3
Document Number <Doc>
Rev
Date: Saturday, September 16, 2017
Sheet 3 of 7
FPGA
STM32
IMU
GPS
LEDS
CAN BUS
MICRO SD
VIN MON
U6 SN74LVC8T245
U7 SN74LVC8T245
U8 SN74LVC8T245
U9 SN74LVC8T245
U10 MAX3051EKA
U11 SN65HVD72DGK
Q2 TLP3123
Q3 2N7002
J5 Micro SD Connector
L4 FERRITE BEAD 120
R12 120
R13 470
R19 470
R11 10K
R25 10K
R27 10K
R14 270
R15 270
R16 270
R17 270
R21 47K
R23 10K
D7 LED
D8 LED
D9 LED
D10 LED
Reference
RELAY
REL_1A
REL_1B
BUF_OUT[0..11]
BUF_IN[0..11]
ALARM
F1_UART_RX
F1_UART_TX
F2_UART_RX
F2_UART_TX
CAN_TX
CAN_RX
CANL
CANH
VIN_MON
SD_CS
SD_DI
SD_SCLK
SD_DO
RS485_A
RS485_B
MCU_LED1
MCU_LED2
FPGA_LED1
GPS_TX
GPS_RX
3.3V
5.0V


U12-2
EP4CE22E144
VERSION : 1.0
PAGE : 2 of 5
DATE : March 2010
BUF_OUT4 10 IO_B1_10/ DIFFIO_L6P/ DQS0L/CQ1L,DPCLK0
BUF_OUT5 11 IO_B1_11/ DIFFIO_L6N
BUF_OUT10 32 IO_B2_32/ RUP1
BUF_OUT11 33 IO_B2_33/ RDN1
BUF_OUT8 28 IO_B2_28/ DIFFIO_L8P
BUF_OUT9 30 IO_B2_30/ DIFFIO_L11P/ DQS1L/CQ1L#,DPCLK1
BUF_IN9 43 IO_B3_43/ PLL1_CLKOUTP
BUF_IN8 44 IO_B3_44/ PLL1_CLKOUTN
BUF_IN11 39 IO_B3_39/ DIFFIO_B2P
BUF_IN10 42 IO_B3_42/ DQS1B/CQ1B#,CDPCLK2
BUF_OUT1 49 IO_B3_49/ DIFFIO_B9N/ DQ1B
BUF_OUT2 50 IO_B3_50/ DIFFIO_B10N/ DQ1B
BUF_OUT3 51 IO_B3_51/ DIFFIO_B11P/ DQ1B
BUF_IN2 66 IO_B4_66/ RUP2/ DQ1B
BUF_IN1 67 IO_B4_67/ RDN2/ DQ1B
BUF_IN7 58 IO_B4_58/ DIFFIO_B16P/ DQ1B
BUF_IN6 59 IO_B4_59/ DIFFIO_B17P/ DQ1B
BUF_IN5 60 IO_B4_60/ DIFFIO_B17N/ DQ1B
BUF_IN4 64 IO_B4_64
BUF_IN0 68 IO_B4_68/ DIFFIO_B23N/ DQS0B/CQ1B,CDPCLK3
ALARM 69 IO_B4_69/ DIFFIO_B24P
UART1_TX 71 IO_B4_71/ PLL4_CLKOUTP
UART1_RX 72 IO_B4_72/ PLL4_CLKOUTN
IO_B5_76/ RUP3 76 IMU_CS_XM
IO_B5_77/ RDN3 77 IMU_CS_G
IO_B5_83/ DIFFIO_R11P 83 IMU_SDO_XM
IO_B5_85/ DIFFIO_R10P/ DQS1R/CQ1R#,DPCLK6 85 IMU_SDO_G
IO_B6_100 100 IMU_SDI
IO_B6_104/ DQS0R/CQ1R,DPCLK7 104 IMU_SPC
IO_B7_114/ RUP4/ DQ1T 114 RELAY
IO_B7_115/ RDN4/ DQ1T 115 GPIO2
IO_B7_112/ PLL2_CLKOUTN 112 IMU_B_MOSI
IO_B7_113/ PLL2_CLKOUTP 113 IMU_B_SCK
IO_B7_110/ DIFFIO_T23P/ DQS0T/CQ1T,CDPCLK6 110 IMU_B_CS
IO_B7_111/ DIFFIO_T22P 111 IMU_B_MISO
IO_B8_143/ PLL3_CLKOUTN/ DQ1T 143 UART3_RX
IO_B8_144/ PLL3_CLKOUTP/ DM1T 144 UART3_TX
IO_B8_141/ DIFFIO_T2P 141 SPI_CS
RELAY
IMU_B_MOSI
IMU_B_SCK
IMU_B_CS
IMU_B_MISO
UART3_RX
UART3_TX
SPI_CS
ALARM
UART1_TX
UART1_RX
IMU_DRDY_G
IMU_CS_XM
IMU_CS_G
IMU_SDO_XM
IMU_SDO_G
IMU_SDI
IMU_SPC
IMU_DEN_G
GPIO[0..8]
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
GPIO8
BUF_IN[0..11]
BUF_IN0
BUF_IN1
BUF_IN2
BUF_IN3
BUF_IN4
BUF_IN5
BUF_IN6
BUF_IN7
BUF_IN8
BUF_IN9
BUF_IN10
BUF_IN11
BUF_OUT[0..11]
BUF_OUT0
BUF_OUT1
BUF_OUT2
BUF_OUT3
BUF_OUT4
BUF_OUT5
BUF_OUT6
BUF_OUT7
BUF_OUT8
BUF_OUT9
BUF_OUT10
BUF_OUT11
J10
3.3V
GPIO5
GPIO6
GPIO7
GPIO8
EXP CONN
U22
TRISTATE
VDD
GND
OUT
CLK_50
ASE-50.000MHz-ET
C65
0.1uF
U12-3
EP4CE22E144
VERSION : 1.0
PAGE : 3 of 5
DATE : March 2010
BUF_OUT6 7 VREFB1N0
BUF_OUT7 31 VREFB2N0
BUF_OUT0 46 VREFB3N0
BUF_IN3 65 VREFB4N0
GPIO7 80 VREFB5N0
GPIO5 105 VREFB6N0
GPIO1 119 VREFB7N0
SPI_SCK 136 VREFB8N0
CLK1_23/ DIFFCLK_0N 23 CLK_50
CLK2_24/ DIFFCLK_1P 24
CLK3_25/ DIFFCLK_1N 25
CLK4_91/ DIFFCLK_2P 91
CLK5_90/ DIFFCLK_2N 90
CLK6_89/ DIFFCLK_3P 89
CLK7_88/ DIFFCLK_3N 88
CLK8_126/ DIFFCLK_5N 126
CLK9_127/ DIFFCLK_5P 127
CLK10_128/ DIFFCLK_4N 128
CLK11_129/ DIFFCLK_4P 129
CLK12_55/ DIFFCLK_7N 55
CLK13_54/ DIFFCLK_7P 54
CLK14_53/ DIFFCLK_6N 53
CLK15_52/ DIFFCLK_6P 52
SPI_SCK
U12-4
EP4CE22E144
VERSION : 1.0
PAGE : 4 of 5
DATE : March 2010
NCSO 8 IO_B1_8/ DIFFIO_L4P/ FLASH_NCE,NCSO
FPGA_LED1 86 IO_B5_86/ DIFFIO_R9N/ DEV_OE
GPIO8 87 IO_B5_87/ DIFFIO_R9P/ DEV_CLRN
INT2_XM 98 IO_B6_98/ DIFFIO_R5N/ INIT_DONE
GPIO4 99 IO_B6_99/ DIFFIO_R5P/ CRC_ERROR
INT1_XM 101 IO_B6_101/ DIFFIO_R4N/ NCEO
GPIO6 103 IO_B6_103/ DIFFIO_R4P/ CLKUSR
GPIO0 120 IO_B7_120/ DIFFIO_T19N/ PADD1/ DQ1T
GPIO3 121 IO_B7_121/ DIFFIO_T17P/ PADD4
UART2_TX 125 IO_B7_125/ DIFFIO_T13P/ PADD12
UART2_RX 106 IO_B6_106/ DIFFIO_R1N/ PADD20
DATA0 13 IO_B1_13/ DATA0
ASDO 6 IO_B1_6/ DIFFIO_L3N/ DATA1,ASDO/
IMU_DRDY_G 132 IO_B8_132/ DIFFIO_T10N/ DATA2/ DQ1T
IMU_DEN_G 133 IO_B8_133/ DIFFIO_T10P/ DATA3/ DQ1T
SPI_MISO 135 IO_B8_135/ DIFFIO_T9P/ DATA4/ DQ1T
MCU_RST 137 IO_B8_13/ DATA5/ DQ1T
SPI_MOSI 142 IO_B8_142/ DATA12/ DQS1T/CQ1T#,CDPCLK7
R31
22
FPGA_LED1
INT2_XM
INT1_XM
UART2_TX
UART2_RX
SPI_MISO
MCU_RST
SPI_MOSI
CONF_DONE 92 CONF_DONE
DCLK 12 DCLK
MSEL0 94 MSEL0
MSEL1 96 MSEL1
MSEL2 97 MSEL2
NCE 21
NCONFIG 14 NCONFIG
NSTATUS 9 NSTATUS
TCK 16 TCK
TDI 15 TDI
TDO 20 TDO
TMS 18 TMS
3.3V
R28 10K
R29 10K
R30 10K
J6
VCCA
TCK 1 2
TDO 3 4
TMS 5 6
7 8
TDI 9 10
5x2_HEADER_2MM
VCCA
R32 0
R33 0
R34 0
MSEL0
MSEL1
MSEL2
R38 0
R39 0
R40 0
R35 10K
R36 10K
TDI
TMS
TCK
R37 1K
U13
ASDO 5 ASDI
NCSO 1 CS#
DATA0 2 DATA
DCLK 6 DCLK
VCC3 3
VCC2 8
VCC1 7
GND 4
EPCS16SI8N
C17 0.1uF
C18 0.1uF
Serial Configuration Device
Cyclone IV Device
Download Cable 10-Pin Male Header (Top View)
nCE (4)
nCEO
N.C. (5)
nSTATUS
CONF_DONE
nCONFIG
MSEL[]
DATA[0]
DCLK
nCSO (8)
ASDO (8)
CLKUSR
TCK
TDO
TMS
TDI
Serial Flash Loader
DATA
DCLK
nCS
ASDI
10 kΩ
25 Ω (7)
1 kΩ
Pin 1
GND
Title
FPGA LOGIC
Size
A3
Document Number
<Doc>
Rev
Date: Saturday, September 16, 2017
Sheet 4 of 7


FPGA POWER
Title
Size A4
Document Number <Doc>
Rev
Date: Saturday, September 16, 2017
Sheet 5 of 7
U15 LMZ12001TZ-ADJ/NOPB
VIN RON EN SS VOUT FB GND GND2 PAD
R52 22.6K
R53 4.22K
R54 8.45K
R55 22.6K
R56 12.4K
C66 10uF
C67 22nF
C68 1uF
C69 22nF
C76 100uF
U23 LP5900SD-2.5/NOPB
VIN EN VOUT GND NC2 NC1 PAD
R57 10K
C77 0.1uF
C78 10uF
L5 FERRITE BEAD 120
L6 FERRITE BEAD 120
C31 10uF
C36 10uF
U16 LP38500SD-ADJ/NOPB
IN1 IN2 IN3 ADJ OUT3 OUT2 OUT1 GND1 GND2 PAD
R45 5.36K
R46 1.2K
C79 10uF
C80 0.1uF
C19 0.1uF
C20 0.1uF
C21 0.1uF
C22 0.1uF
C23 0.1uF
C24 0.1uF
C25 0.1uF
C26 0.1uF
C27 0.1uF
C28 0.1uF
C29 0.1uF
C30 0.1uF
C32 0.1uF
C33 0.1uF
C34 0.1uF
C35 0.1uF
C37 0.1uF
C38 0.1uF
C39 0.1uF
C40 0.1uF
C41 0.1uF
C42 0.1uF
C43 0.1uF
C44 0.1uF
C45 0.1uF
C46 0.1uF
U12-1 EP4CE22E144 VERSION : 1.0 PAGE : 1of 5 DATE : March 2010
VCCINT VCCIO1 VCCIO2 VCCIO3 VCCIO4 VCCIO5 VCCIO6 VCCIO7 VCCIO8
VCCD_PLL1 VCCD_PLL2 VCCD_PLL3 VCCD_PLL4
VCCA1 VCCA2 VCCA3 VCCA4
U12-5 EP4CE22E144 VERSION : 1.0 PAGE : 5 of 5 DATE : March 2010
GND GNDA1 GNDA2 GNDA3 GNDA4 PAD
1.2V
2.5V
3.3V
5.0V
VCCA
VPLL
EN
A B C D
1 2 3 4 5



Title	IMU SENSORS
Size A3	Document Number <Doc>	Rev
Date: Saturday, September 16, 2017	Sheet 6 of 7



Title
DSP
Size
A4
Document Number
<Doc>
Rev
Date:
Saturday, September 16, 2017
Sheet
7
of
7
U20
STM32F405RGT6
PA0-WKUP
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PA8
PA9
PA10
PA11
PA12
PA13
PA14
PA15
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PB8
PB9
PB10
PB11
PB12
PB13
PB14
PB15
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC9
PC10
PC11
PC12
PC13-TAMPER-RTC
PC14-OSC32_IN
PC15-OSC32_OUT
PH0-OSC_IN
PH1-OSC_OUT
PD2
VBAT
VDD_1
VDD_2
VDD_3
VDD_4
VDDA
VSSA
VCAP_1
VCAP_2
VSS_3
VSS_4
NRST
BOOT0
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
GPIO8
GPIO[0..8]
SPI1_NSS
SPI1_SCK
SPI1_MISO
SPI1_MOSI
I2C_SCL
I2C_SDA
USB_VBUS
LED1_MCU
USB_DN
USB_DP
TMS/SWDIO
TCK/SWDCLK
TDI/NC/EXTB
TDO/SWO
TRESET#
BOOT1
USART_TX
USART_RX
CAN_RX
CAN_TX
GPS_USART_TX
GPS_USART_RX
SPI2_NSS
SPI2_SCK
SPI2_MISO
SPI2_MOSI
ADC_1
ADC_2
RS485_TX
RS485_RX
RS485_DIR
UART1_TX
UART1_RX
LED2_MCU
OSC1
OSC2
3.3V
U19
M24128
A0
A1
A2
SCL
WP
VCC
SDA
R47
1.5K
R48
1.5K
C52
0.1uF
Y1
TSX-3225 24.0000MF15K-AC3
C51
10pF
C53
10pF
J7
5x2_HEADER_2MM
C54
0.1uF
C55
0.1uF
C56
0.1uF
C57
0.1uF
C60
2.2uF
C63
2.2uF
R49
10K
C61
0.1uF
J8
RESET_SW
R50
10K
R51
10K
A
B
C
D
1
2
3
4
5

REFERENCES

- Balsi M., Esposito S., Fallavollita P., Giannì C., "Airborne LiDAR scanning for forest biomass estimation", 2016 ENVRIPlus Workshop Unmanned Vehicles in Research

- Burri M., Oleynikova H., Achtelik M.W., Siegwart R. "Real-time visual-inertial mapping, re-localization and planning onboard MAVs in unknown environments", *2015 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS),* DOI 10.1109/IROS.2015.7353622

- Cheung P.Y.K, http://www.ee.ic.ac.uk/pcheung/teaching/ee2_signals, 2011, Lectures on Windowing effects in Discrete Fourier Transforms

- DJI Mavic Pro, http://www.dji.com/mavic/info#specs

- Ecalc, http://ecalc.ch, On line rc calculator

- ENAC Regolamento mezzi aerei a pilotaggio remoto (Italian regulation on RPAS)

- Fallavollita P., Esposito S., Balsi M., "Perception and decision systems for autonomous UAV flight", in Di Giamberardino P. *et al.*, "Computational Modelling of Objects Represented in Images III Fundamentals, Methods and Applications", CRC Press, 2012 - ISBN 9780415621342

- Fasano G., Accardo D., Tirri A.E., Moccia A. "Experimental Analysis of Onboard Non-Cooperative Sense and Avoid Solutions Based on Radar, Optical Sensors, and Data Fusion*", IEEE A&E Systems Magazine*, 2016, DOI 10.1109/MAES.2016.150164

- Forster, Christian, Matia Pizzoli, and Davide Scaramuzza. "SVO: Fast semi-direct monocular visual odometry." Robotics and Automation (ICRA), 2014 IEEE International Conference on. IEEE, 2014.

- Gageik N., Benz P., Montenegro S., “Obstacle Detection and Collision Avoidance for a UAV With Complementary Low-Cost Sensors”, *IEEE Access* 3, 599-609 (2015), DOI 10.1109/ACCESS.2015.2432455

- Groves P. “Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems” , Artech House , 2008

- Grzonka S., Grisetti G., Burgard W. “A fully autonomous indoor quadrotor”, IEEE Transactions on Robotics, vol. 28, pag. 90-100, 2012

- Gohl P. *et al.*, "Omnidirectional Visual Obstacle Detection using Embedded FPGA," *Int. Conf. Intelligent Robots and System (IROS),* 2015, DOI: 10.1109/IROS.2015.7353931

- Hollinger J., Kutscher B., Closeb R., “Fusion of Lidar and Radar for detection of partially obscured objects”, in Karlsen R.E. *et al.* (ed.) “Unmanned Systems Technology XVII “, Proc. of SPIE VoI. 9468, DOI 10.1117/12.2177050

- Horn B.K.P., Schunck B.G., “Determinig Optical Flow”, 1981 Artificial Intelligence Laboratory, M.I.T.

- Innosent GmbH, "Datenblatt_IVS-167_V3.1.pdf," 2013 - http://www.innosent.de/fileadmin/media/dokumente/DATASHEETS_2016/Datenblatt_IVS-167_V3.1.pdf

- Lucas B. D., Kanade T. (1981) “An iterative image registration technique with an application to stereo vision”

Proceeding of the 7th International. Joint Conference on Artificial Intelligence, 24-28 August 1981, Vancouver, B.C.

- Kaplan E., Hegarty C., “Understanding GPS, Principles and Applications”, Second Edition, Artech House 2006

- Kanellakis C., Nikolakopoulos G. , “Survey on Computer Vision for UAVs: Current Developments and Trends*”, J Intell Robot Syst* (2017), DOI 10.1007/s10846-017-0483-z

- Khaleghi B., Khamis A., Karray F.O., Razavi S.N., “Multisensor data fusion: A review of the state-of-the-art”, *Information Fusion* 14 (2013) 28–44

- Majumder S., Shankar R., M.S. Prasad, “Obstacle Size and Proximity Detection Using Stereo Images for Agile Aerial Robots”, 2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)

- Masali L., “Multicotteri & Droni, guida pratica”, 2015 Dronezine Editore

- MaxBotix, "XL-MaxSonar-EZ_Datasheet.pdf": http://www.maxbotix.com/documents/XL-MaxSonar-EZ_Datasheet.pdf

- Mueggler, Elias, et al. "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM." The International Journal of Robotics Research 36.2 (2017): 142-149.

- Nieuwenhuisen M. *et al.*, “Multimodal Obstacle Detection and Collision Avoidance for Micro Aerial Vehicles”, 2013 Eur. Conf. on Mobile Robots (ECMR), DOI: 10.1109/ECMR.2013.6698812

-Nieuwenhuisen M., Droeschel D., Beul M., Behnke S. “Obstacle detection and navigation planning for autonomous micro aerial vehicles”. International Conference on Unmanned Aircraft Systems (ICUAS) Pages 1040-1047, 2014.

- PulsedLight3D, "LIDAR-Lite-v1-docs.pdf," 2015: https://github.com/PulsedLight3D/LIDAR-Lite-Documentation/tree/master/Docs

- RadarTutorial, http://www.radartutorial.eu

- Sabatini R., Gardi A., Ramasamy S., Richardson A., “A Laser Obstacle Warning and Avoidance System for Manned and Unmanned Aircraft”, 78-1-4799-2069-3/14 ©2014 IEEE

- Sabatini R., Ramasamy S., Gardi A., Rodriguez Salazar L., “Low-cost Sensors Data Fusion for Small Size Unmanned Aerial Vehicles Navigation and Guidance”, *Int J Unmanned Systems Eng.* (2013), 1(3), 16-47

- ST Microelectronics, STM32F446xC/E datasheet: www.st.com/resource/en/datasheet/stm32f446ze.pdf

Sunflower Labs, https://sunflower-labs.com

- Welch G., Bishop G., "An Introduction to the Kalman Filter," *UNC-Chapel Hill TR 95-04* (2006) - https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

- Zufferey J.C., “Bio-Inspired Flying Robots, experimental synthesis of autonomous indoor flyers”, 2008 EPFL Press