Next generation radiation mapping using UAS assisted dynamic monitoring networks

Agathangelos Plastropoulos

MSc in Robotics

Module Number: UFMED4-60-M

Faculty of Environment and Technology – Engineering, Design and Mathematics,
University of the West of England, Bristol

Faculty of Engineering – Engineering Mathematics, University of Bristol

September 2015

This study was completed for the MSc in Robotics at the University of Bristol and the University of the West of England, Bristol. The work is my own. Whenever the work of others is used or drawn on, this is adequately referenced.

Agathangelos Plastropoulos

Number of words: 23736

Abstract

Nowadays, unmanned aerial systems (UASs) have become increasingly popular in a wide range of applications, from military to recreational. The Interface Analysis Centre (IAC) at the University of Bristol has developed a UAS, called the AARM (Advance Airborne Radiation Monitoring) system, which could be used to provide visual and thermal monitoring of radiation after a release of nuclear material. AARM system has successfully measured radiation in one of the most highly contaminated prefecture areas in the Fukishima Daiichi nuclear power plant after the disaster caused by the earthquake and tsunami in March 2011.

The purpose of this project is to study ways in which the capability for radiation surveillance at civil nuclear sites of the first generation AARM system could be advanced. In the context of this project the following improvements have been proposed:

- Equipping the UAS with advanced sensors. An extensive testing of proximity sensors has been conducted to identify the strengths and the weaknesses of the existing commercial of-the-self (COTS) sensors. The laser sensor and the ultrasonic sensor have been utilised as basic components for building a custom measurements device.
- Developing a realistic simulation and assisted flight method. A comprehensive simulation based on ROS and gazebo has been developed. Using that simulation, an assisted teleoperation mode for obstacle avoidance has been implemented.
- Improving human robot interaction. The main mission of the UAS is to map the radiation levels. An iOS mobile application for visualising the radiation intensities (cps) on a map has been developed. In addition, a second iOS application was implemented which can be utilized to teleoperate a ROS-enabled UAV.

- Developing a custom quad-copter for the obstacle avoidance applcation. At first, a basic approach was implemented. The operator flies a ready-made copter manually. The copter is equipped with a custom measurements device, which mesures the distances in front and at the sides and sends visual feedback to a smart device. In the advanced approach, a custom X4 UAV was built. It was equipped with an on-board computer and an enhanced version of the custom measurements device. Experiments were conducted succesfully that verify the basic behaviour of the UAV. Further testing is required for the tuning and the improvement of the autonomous behaviour.

All in all, during this project a set of useful tools has been developed. The successful combination of all those tools enables us to design a radiation mapping UAV that will advance the performance of the existing AARM.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr Thomas B. Scott. He was always there to support, inspire and care about the progress of my research. I would like to thank Peter Martin for his continuous help to supply the numerous hardware parts required for my project.

My sincere thanks also goes to Chris Abbott, Chairman and CEO of Imitec LTD, for allowing me access to the company's resources. Especially, I would like to thank Chris Telford for helping me in every technical aspect for the design and the implementation of the project's UAV.

I would also like to thank Josh Hugo, the Deputy Workshop Manager of the Physics Workshop, for his valuable help in the custom UAV parts.

I would like to thank my friend Theoni Reid for her tireless help the last two years.

I would like to thank my parents and my sister for their support every time that I needed them.

Last but not least, I would like to thank my wife Helen and my son Aris for always being there to support me, believe in me and make me smile.

Supporting Files

All the project's related files, including source code and documentation can be found in the online repository: goo.gl/8IKAeE

Videos related with the project's experiments are available online in: goo.gl/dhUPWn

Contents

Part 1

Introduction

1.1 Unmanned Aerial Vehicles (UAVs)

Over the last few years, one of the fastest growing fields of Robotics and Autonomous Systems is the Unmanned Aerial Vehicles (UAVs). There are also other names that refer to this specific fields likes drones, Unmanned Aircraft Systems (UAS) or Unpiloted Aerial Vehicles, all having similar meaning and subtle differences. In general, an Unmanned Aerial Vehicle is an aircraft that can be controlled without humans on board, either remotely or autonomously. Unfortunately, UAVs were made popular when they first appeared because they reduce collateral damage in wars and because were convenient seeing as the controller can be thousands of miles away from the UAV itself. However, this is not the whole truth and the UAVs' field has the potential to be exploited in a wide range of non-military activities with impressive outcomes. The development of this technology has a field of great competence of countries around the globe. Nevertheless, the deployments are not solely based on the technological progress but also on the operation framework that will be imposed by national administration aviation authorities like Federal Aviation Administration (FAA) the States and Civil Aviation Authority (CAA) in the UK.

Similar to any other revolutionary technological development, like the Internet for example, UAVs have faced controversial reactions. Many people feel that UAVs threat their privacy while others look forward to using them to facilitate their business. Furthermore, political and ethical problems pose an issue. The reality is that technologies develop regardless of public issues. It is a question of how they will be used and for what reason. There are cases though, where everyone unambiguously agrees that it is better to use an UAV than the human presence. For example, when the Fukushima Daiichi nuclear power plant was severely stricken by the tsunami, there was no way to assess the contamination since static radiation monitoring stations around the site were disabled. Because of that, a traditional manned airborne was conducted at the nuclear site. Beside the poor resolution of the measurements due to the unfriendly environment, it is clear

that the flight crew was exposed to a significant amount of radiation. Consequently, the exploitation of UAVs in places where humans must not be exposed to dangerous situations is of great interest and therefore deserves further research.

The development of the UAVs has related impacts on the national economies as well. However, because of the current airspace restrictions non-defense use of UAS has been extremely limited. A well-known case is Amazon's prime air concept, a futuristic delivery system for the company's products. The biggest hurdle facing Amazon Prime Air is that commercial use of UAV technology is not yet legal in the United States.

1.2 Discussion about various UAV aspects

Unmanned Aerial Vehicles is one of the most popular emerging fields in robotics. It attracted much criticism yet many proponents believe that UAVs' applications would be beneficial for humanity. One thing is certain; as the civil applications of UAVs increases the greater the debate will become. In general, I can refer to the following strengths of the UAVs:

- It is a cost effective aerial transportation and at the same time it has a smaller carbon footprint compared to the traditional aerial ways.
- Using UAVs, makes access to remote areas easier.
- Micro UAVs are the only type of aircraft in the market that can be designed and manufactured by small-scale companies anywhere in the world. It is very important that the parts that make up a UAV are based on existing technologies, but at the same time a significant amount of research is directed towards emerging technological approaches.

Nevertheless, there are certain problems that affect the progress of the UAVs. To summarize the weakness and the dangers, I can refer to the following:

- The national aviation regulations around the world need to be modified in order to allow for the healthy development of UAVs. In addition, legal frameworks should be amended to protect people from malicious surveillance. Simultaneously to the national update of regulations, unified

international standards should be developed to allow a homogeneous approach to issues such as crew certification and safety.

- The widespread of a new technology, such as UAVs, which has not been exhaustively tested thought the years, may have high failure rates at the early stages.
- The ethical impact related to the UAVs has several times been the field of conflict where non-governmental and social organizations have had a leading role. Sometimes the influence of these groups in technological issues is not so beneficial for technological developments and experimentation.

The development of UAVs can offer the following opportunities to the world economy:

- According to recent studies, it is a new emerging market with impressive forecasts in terms of economic revenue and new jobs created.
- The commercial and civil UAV market will create new UAV based services. Many entrepreneurs are awaiting the legal framework to deploy their services.
- A need field for the development of software applications and software development kits will be created.
- Use of alternative energy sources for longer flight times such as fuel cells and solar energy will become more popular.

To conclude, UAV came and will be part of our everyday life in the near future. Regulation and legal frameworks are required for a healthy development and application of this technology. In addition to that, I believe that common sense from us the users is required and will play a major role. Technology is harmless unless people use it in improper ways. The economic benefits are predicted to be very high in fields such as agriculture and public safety. Many new jobs will be created around the world. Ultimately, I am ambitious that this field of robotic and autonomous systems will be a beneficial development for humanity.

1.3 Interface Analysis Centre (IAC)

Interface Analysis Centre is a university research group, hosted in the School of Physics at Bristol University. IAC combines material scientists and engineers that concentrate on answering why the materials have certain properties and behaviors and how these can be exploited to compose something better or cheaper. Using special equipment, IAC aims to deduce how the properties (like strength, transparency, flexibility, conductivity) are related to the structure (like crystal size, alignment of fibres). After all, the challenge is to use this knowledge to design and manufacture materials, which have better properties for the required purpose (Bristol.ac.uk, 2015).

One of the specific tasks in which IAC focuses is the visual and thermal monitoring of radiation after a release of nuclear material. It has already developed a semi-autonomous drone, called Advance Airborne Radiation Monitoring (AARM) system, which was jointly funded by the Engineering and Physics Sciences Research Council (EPSRC) and Sellafield Ltd. The latter is the national organization tasked with the managing and safely storing the UK's nuclear waste. The initiation of this idea happened in response to requirements for radiation monitoring needed at Fukishima Daiichi Nuclear Power Plant and in the surrounding affected prefecture. After the first successful mission (Imitec, 2014) of flying at Sellafield nuclear site in spring 2014, a small team led by Dr. Tom Scott employed two AARM systems to provide an initial assessment of contamination in one of the most highly contaminated prefecture areas within the Abukuma river basin, in the Kawamata area of Japan (Scott, 2014). Sellafield and the Bristol-Kyoto Strategic Partnership Fund jointly sponsored this expedition.

1.4 The project

The purpose of this project is to advance the existing capability for radiation surveillance at civil nuclear sites of the first generation AARM system. The use of UAV in radiation mapping is a very convenient way with many advantages such as safety, accuracy and flexibility. However there are certain requirements that

need to be satisfied in order to yield high quality measurement data. The UAV should fly low, slowly and with precision. Therefore for reliable measurements, the UAV should fly close to buildings or near the ground and maintain this distance constant. In addition, nuclear sites where most frequently radiation mappings take place have a very complex geometrical shape (fig.1.1). This makes the task even more difficult and the need for a solution a necessity. Unfortunately, up to this point this type of flight is not feasible if using a helicopter or existing UAS technology. This is the exact point where this project gets context for existence.

Fig.1.1 Sellafeld nuclear site close to the village of Seascale on the coast of the Irish Sea in Cumbria (image taken from Wikipedia.org)

Having described the requirements and the problems for obtaining high quality radiation measurements, the aim of this project is to explore the development of a UAV, which will be equipped with devices and will execute algorithms that will facilitate this procedure. This is the next generation radiation mapping UAV that is addressed in the title of the project. In order to reach the ultimate goal, the plan is to develop a set of useful tools that will improve certain individual aspects in the behavior and performance of the UAV. The efficient combination of all will

enable us to have a radiation mapping UAV closer to the one that the initial requirements have set.

The overall task can be divided to the following individual objectives:

- Experimentation with a range of proximity sensors and comparison of them using factors such as functionality, performance and size. During that procedure, existing problems and limiting factors are identified. Afterwards, the most versatile sensor will be chosen to develop possible robotic applications and the potential to engage them in real robots will be explored.
- Employment of a realistic 3D simulation to experiment with various collision avoidance maneuvers that can be used in an assisted flight mode.
- Improvement of the human robot interaction. Design and development of a mobile application that depicts the level of radiation contamination on a map in real time, taking wireless measurements from a gamma spectrometer. Using this application, the users will be able to observe in real time the contamination levels being recorded by the UAS and to focus accordingly to the areas of interest. In addition, design and development of a mobile application that enables the control of a ROS-enabled UAV. By implementing this application, an extensive exploration in interconnection of heterogeneous systems is accomplished where the goal is to enable the control of UAV using a smart device. The rich visualization capabilities that these devices can provide enhance the experience in the human robot interaction.
- Configuration and implementation of the next generation UAV using the appropriate advanced components. Moreover, having a working prototype, which means that all the individual components are interconnected and functioning together, we can experiment with real flights and simple obstacles.

A more illustrative way to show the objectives of this project is depicted in the following figure (fig.1.2).

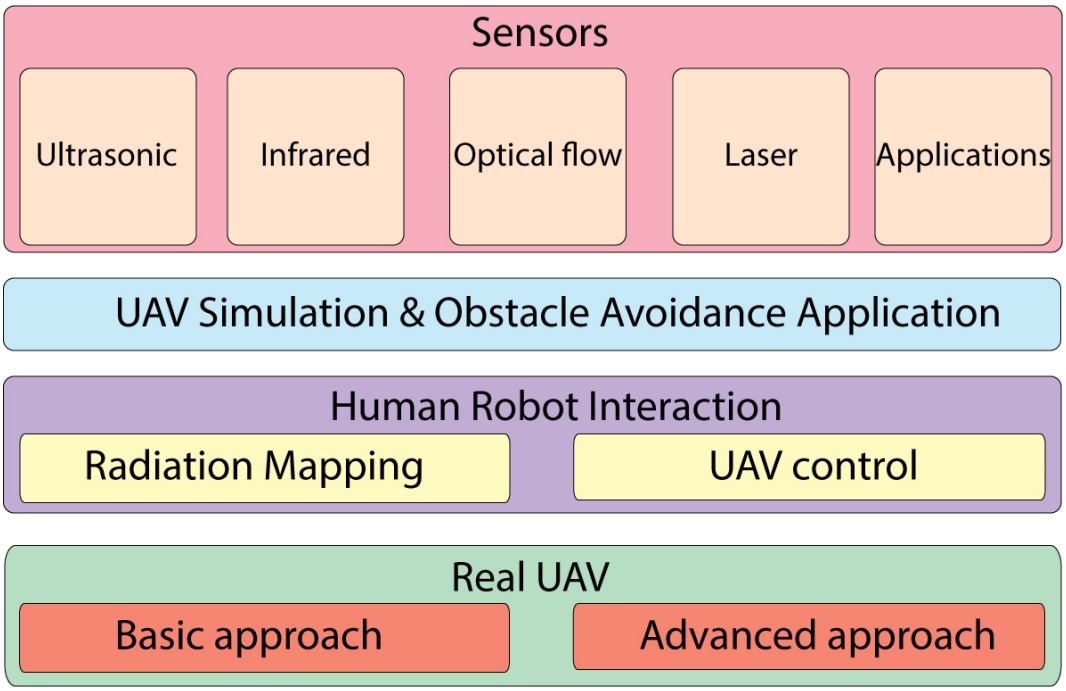


Fig.1.2 A schematic representation of the project's structure

To give a brief guide about the objectives related with the location in this dissertation the following figure (fig.1.3) is included. The partition of the topics in different location serves primarily the need to separate the description of the basic concepts from technical details regarding the implementation and the functionality.

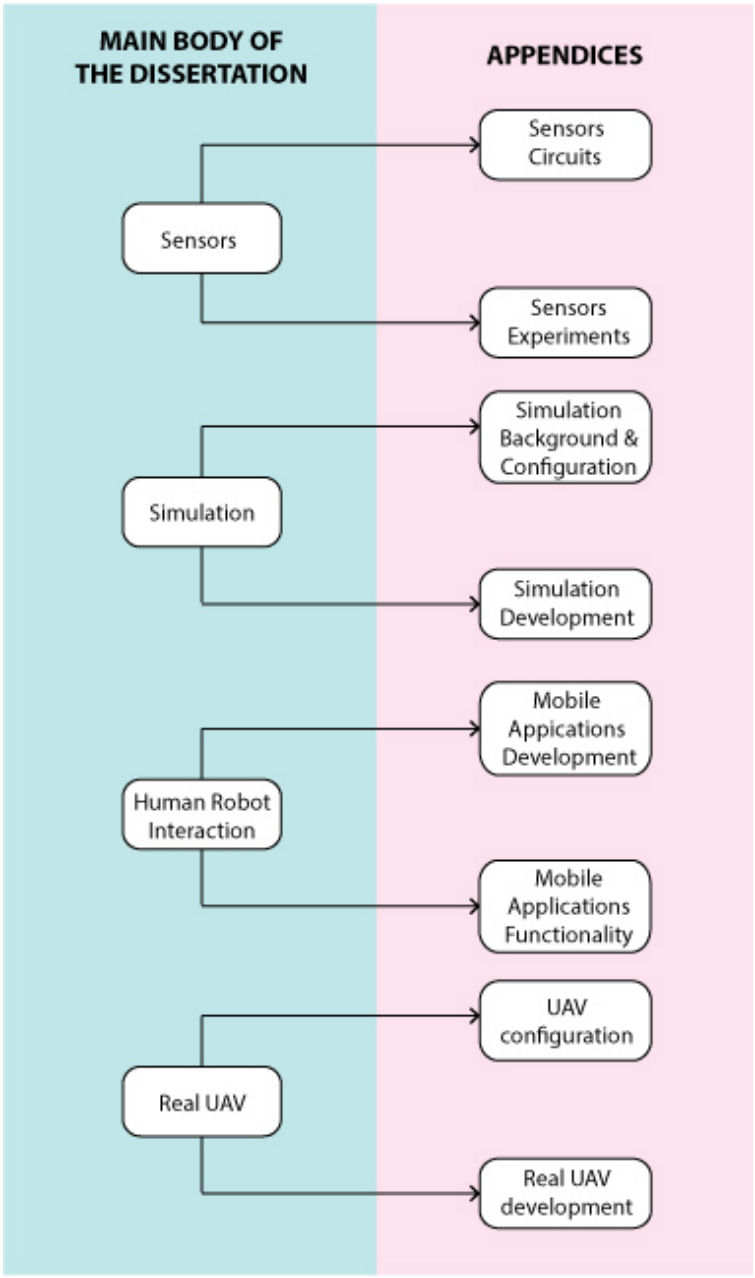


Fig.1.3 A schematic representation of the project's structure

Part 2

Literature review

In this section, an extended literature review is presented which is conducted in the individual sections of the dissertation. Most of the sources are conference papers and journal articles on the basis that they are published recently and thus are more up-to-date compared to books. They reflect the latest developments in each discipline. The vast majority of the sources are less than four years old (dating from 2011 to present). Thorough research is undertaken for every major part of the dissertation in order to get a rounded view of the undergoing research and development that is being carried out.

2.1 Radiation mapping

The radiation anomalies mapping procedure is one of great importance since it can ensure the public safety in cases where disaster happened or in monitoring of nuclear power stations. There are many examples that can show the importance of radiation mapping. In March 2011 The Fukishima Daiichi nuclear power plant was severely stricken by a tsunami causing Units 1 – 4 to contaminate the environment with significant atmospheric release of radioactive material (MEXT, 2015). This case clearly illustrates the requirement for technology that allows rapid and accurate mapping of ground and plume radiation contamination. Moreover, in nuclear power plants there is a constant need for validation of safe operation. This can be achieved by continuous routine surveys to identify contamination releases. Finally, radon mapping is also important because it can affect the public health as it accumulates inside buildings and basements, greatly increasing the risk of lung cancer.

There are three methods of radiation mapping. Static ground based surveys, which performed by arrays of above ground in situ measuring stations or handheld devices. In situ ground level measurements improve spatial averaging at the cost of vertical profiling. The handheld measurement devices produce an

accurate and sensitive representation of how radioisotope concentrations vary spatially or depending on depth. On the other hand, this method is time consuming and captures a limited snapshot of radionuclide distribution (Sanderson et al., 1995). Car-borne radiation surveys conducted using vehicles equipped with detectors, which are calibrated and deployed quickly, and can map 400–1000 km every 24 hours (Mellander, 1995). Using this process, more measurements can be collected compared to the first method but data collection is limited to the road network (hence rural data is often sparse) and to low driving speed (< 30 mph) to maintain an adequate resolution. The third approach relates to airborne radiation surveys. This can cover great areas of interest expeditiously. Coverage per unit is 10^6-10^7 times greater than that of ground based and 10^2-10^3 times that of car-borne surveys because of the greater ground clearance, coverage and the lack of obstacles (Schwarz et al., 1995). It can also be considered the safest since the remote nature of the sensing removes pilots from a percentage of the ground based radiation exposure, although this is not true of radiation plume mapping. In this method, the most influential factor is altitude. Air attenuates radiation and consequently as altitude increases, the intensity detected decreases rapidly (fig.2.1).

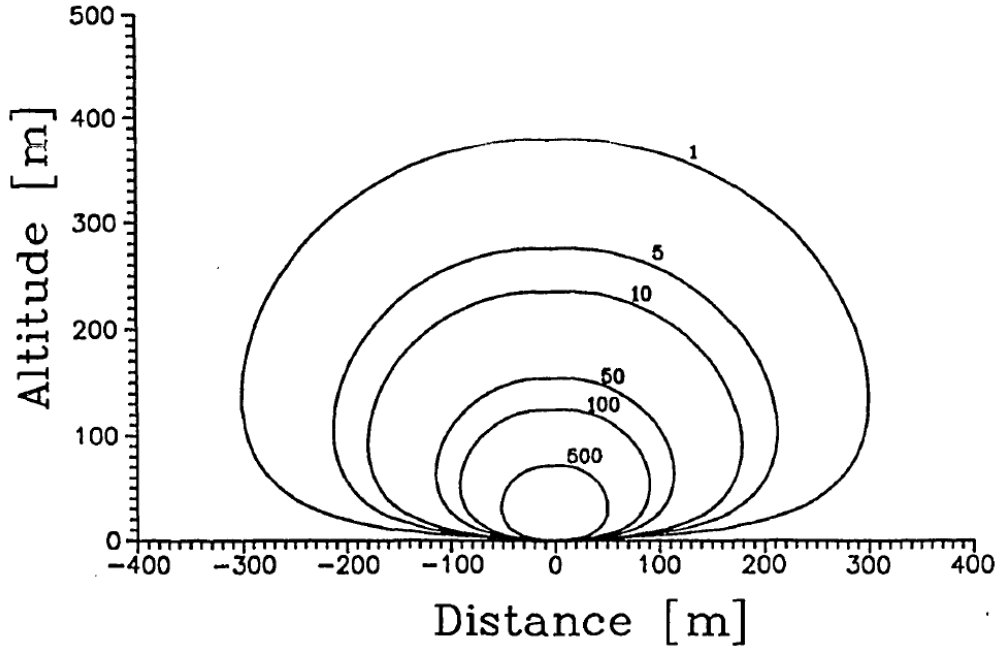


Fig.2.1 Modeled count rate (counts per second) isolines, using a large volume NaI scintillation detector, of a 1.9 GBq (50 mCi) ^{137}Cs source buried at 1 cm depth. (image from Schwarz et at., 1995)

A subcategory of the third approach, which offers the benefits from all three methods, is radiation mapping using UAVs. The utilization of this method seems ideal for the following reasons:

- It can produce high resolution mappings similar to the hand held based surveys since UAVs can hover in low altitudes and follow dense trajectories in given areas.
- UAVs approach the territories of above and avoid local obstacles resulting in a low sampling density. In addition, they have the ability to fly steadily with low velocity in close proximity to nuclear premises and collect accurate measurements.
- UAVs require lower initial expenditure than airborne radiation surveys.
- UAVs ability is not influenced from radioactive debris that could contaminate the UAV like car tyres, which are in constant contact with the road and stick materials from the surface of the road.

Fig.2.2 The Imitec's Advanced Airborne Radiation Monitoring (AARM) UAV (image from www.imitec.co.uk)

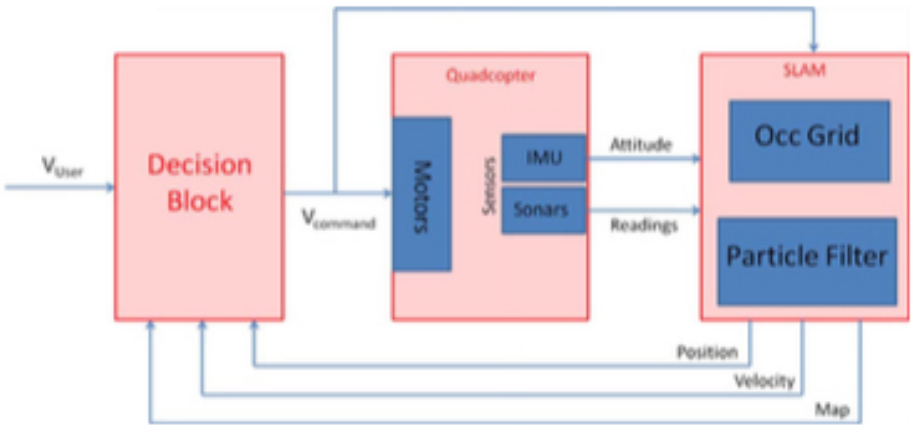
One successful approach that managed to implement a UAV based airborne radiation survey comes from Imitec (Imitec Limited, 2014), a University of Bristol spin-out company. They have developed a remote isotropic analysis system, which consist of a lightweight gamma spectrometer and positioning devices, integrated in a UAV (fig.2.2). Using this combination, it is possible to measure meter resolution maps of radiation including over high dose areas and inaccessible locations. It can also provide aerial imaging and observation in conjunction with the radiation measurements. Finally, there is the ability to

perform pre-programmed flight paths, which is advantageous for scenarios where routine monitoring is required.

2.2 UAV flight

In every mobile robot application, semi-autonomous or autonomous, the task of mapping, localisation, navigation and collision avoidance is important decision in design. The implementation of each part can be done using properly adapted versions of popular methods and probabilistic algorithms available in the robotics literature. This literature review includes the knowledge of related papers concerning UAVs. In order to be more precise I focused my research on quad or multi copters in order to exploit the strengths and combat the weaknesses in this specific configuration.

The Mendes and Ventura 2012 paper (Mendes and Ventura, 2012) proposes an interesting approach to tackle the problem of assisting tele-operation to safely pilot a UAV; a concept that is closely related to the current project. More precisely, according to the environment in which the UAV flies, driven by the operator, the researchers designed a process that overrides this operator input either by modulation, inhibition, or replacement with a different input. The desired goal is to use sensor data to determine an appropriate assisted tele-operation in order to guarantee, to the best of sensor capabilities, a safe flight.



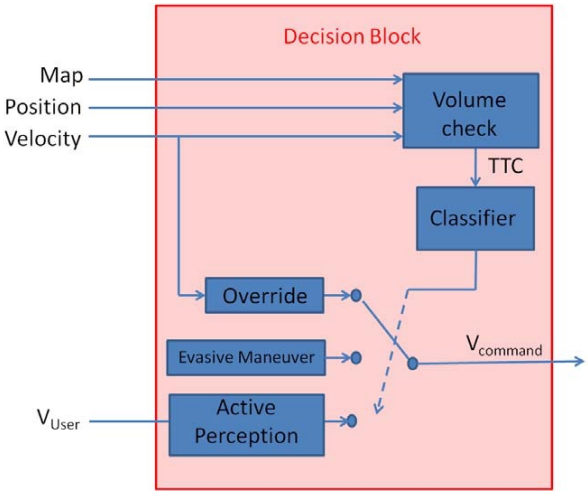


Fig.2.3 Full architecture of the proposed approach and flow chart of the decision block (image from Mendes and Ventura, 2012)

The investigators assume that the UAV operates in unknown, unstructured, GPS-denied, and indoor environments, therefore the main potential threat is collision with obstacles. Thus, distance sensors are used to detect and map these obstacles. The approach is based on the FastSLAM algorithm, together with 3D occupancy grid mapping. Using these techniques they are able to estimate the vehicle's position in the map. At this point a classifier makes a decision based on danger assessment. This classifier overrides the user's inputs if they compromise the quad copter's physical integrity in the near future. Overriding may extend from simple velocity reduction to, in extreme cases, an evasive manoeuvre (fig.2.3).

The approach assumes that the attitude is known using the on-board IMU so the problem is to define the position of the UAV. Using FastSLAM they can estimate both the position and the map of a robot simultaneously. The classifier takes the position estimation and the map as inputs. It categorises each cell of a 3D occupancy grid in one of 3 states *Free, Unknown* and *Occupied*. Given the operator's inputs, the classifier computes the desired direction for the vehicle and the distance to the first non-Free cell. If the closest cell is *Occupied*, the inputs are altered accordingly so that the UAV will remain safe and then they are applied to the navigation. If the closest cell is *Unknown* and if the distance to it is higher than a certain threshold, the algorithm checks whether there is any sonar aligned with the desired direction. Otherwise, the algorithm will autonomously rotate the vehicle and then applies the same speed that the user demanded.

This remarkably interesting approach is based on short-term, rough mapping of the nearby environment. The researchers argued that the real world experiments were successful and thus verified the design. However, it is also mentioned that the four sonars, which were used as proximity sensors, recorded the readings with the quad-rotor's motors set off, for safety issues. Despite the safety benefits in such a procedure the result of the tests could be inaccurate, considering that sonars with their rotors switched on take error prone readings.
In the Chen 2013 paper (Chen et al., 2013) the effort of the research team to develop a UAV system capable of autonomously navigating in a one-level building and producing a 2D map of the space is presented. Moreover, there were specific requirements from the academic partners that needed to be satisfied. More precisely, the system had to be based on an inexpensive, upgradeable, and reconfigurable solution with commercial off-the-shelf (COTS) products.

The hardware of the project consists of an AeroQuad quad-copter, a 54-Pin Arduino Mega SDK microcontroller, a Samsung Galaxy SIII Android smartphone, and MaxBotix ultrasonic range sensors.

For the navigation task of the project they decided to use Voronoi Fast Marching motion planner. They argued that this approach offers a thorough description of the exploration process, the ability to integrate with SLAM, and the potential to safely and effectively navigate with a limited field of view. In the SLAM part of the project they evaluated different existing approaches like GraphSLAM, EKFSlam, GridSLAM and FastSLAM. They resolved on choosing the latter, since FastSLAM is computationally efficient and likely to overcome inaccurate data making it the preferred algorithm for this project

Having made these decisions about the algorithms the research team designed the simulation part of the project. In the paper, they presented the results related to the mapping algorithm and the navigation one. They cited that they hadn't managed to construct the localization prototype due to the lack of time and the complexity, thus they couldn't present results of the FastSLAM approach. As expected, they did not present results from any real experiments.

Due to the lack of actual experimentation and the incomplete state of the simulation results this paper could be deemed a premature project Nevertheless,

it is interesting to read since it assesses available methods for navigation and SLAM and also describes their experiences from the sensors' behaviour (in the calibration stage).

On the contrary to the previous study, a complete and interesting approach is presented in Hrabar's 2011 paper (Hrabar, 2011). In cases where UAVs performs real applications there is a need to move in cluttered environments, making collisions a real possibility. This paper comes to propose an interesting 3D reactive obstacle avoidance technique suitable for UAVs flight in point-to-point trajectories.

In the introduction the author presents common disadvantages in the existing approaches. He refers to techniques that check only current sensor's field of view without keeping memory and also refers to techniques that propose escape points without taking into account the goal point. Ensuingly, he counterpoises an approach where a cylindrical volume in front of the UAV checks for potential collisions and when one is detected an intermediate waypoint (Escape Point) is found in free-space that provides a route past the obstacles that have been mapped (fig.2.4). The Escape Point search proceeds on a series of concentric ellipses, the shape and orientation of which can be adjusted to favour horizontal or vertical avoidance manoeuvres (fig.2.5). Since a waypoint is produced, the technique is suitable for integration with global path planners that plan between waypoints.

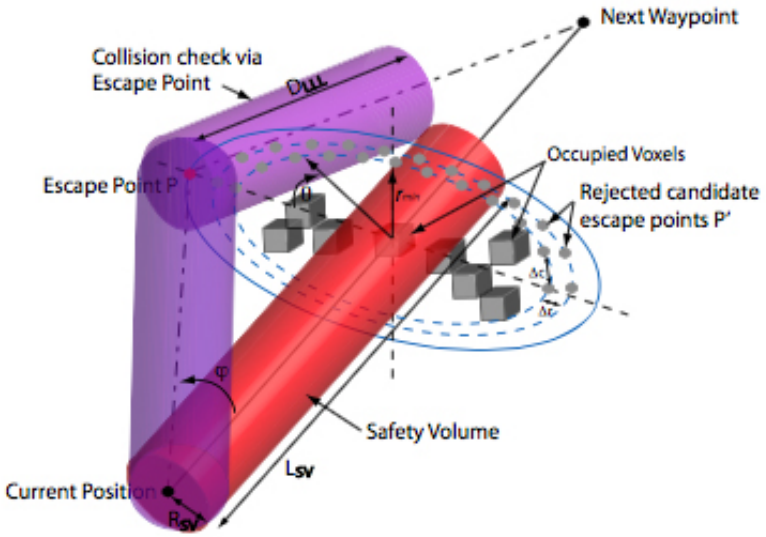


Fig.2.4 The expanding elliptical search for a valid Escape Point (image from Hrabar, 2011)

A voxel-based representation of the 3D environment has been employed to be the basis in which the algorithm is going to be operating. The algorithm maintains an occupancy grid in which both global path planning and reactive avoidance cooperate. In this map-based approach the reactive avoidance mechanism is easily tuneable to favour horizontal or vertical avoidance manoeuvres and generates reactions based on all the previously sensed obstacles, producing an intermediate waypoint which can be used as input for a waypoint-based global path planner.

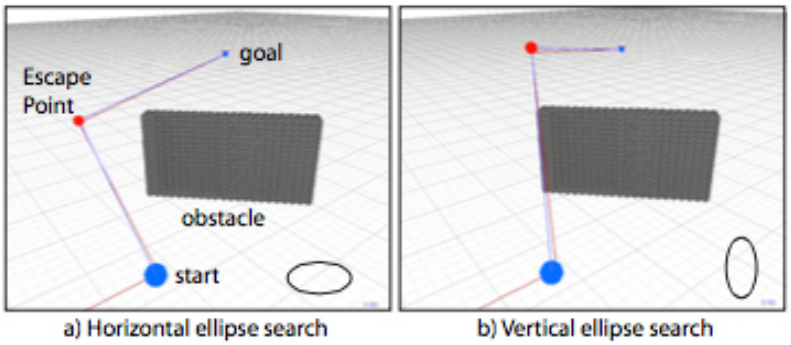


Fig.2.5 With the major axis of the ellipses aligned horizontally, horizontal manoeuvres will be favoured (a), while vertically aligned ellipses will produce vertical manoeuvres (b) (image from Hrabar, 2011)

Special care has been taken to enable the execution of the algorithm in the on-board hardware. The author cites that one of the most demanding tasks of the algorithm is the checking within the occupancy map for occupied voxels, required when testing for potential collisions ahead of the UAV and when evaluating candidate Escape Points. He used a balanced kd-tree (Bkd-tree) structure to represent the occupied voxel centres. Moreover, he compared the Bkd-tree based query times to a traditional brute-force linear search. For smaller voxel populations the techniques show similar performance, however as the voxel count grows the search time increases linearly for the linear search while remaining almost constant for the Bkd-tree search.

After an extensive and detailed explanation of the algorithm, the author presents the experimental results. The UAV was tasked to fly between pairs of waypoints either side of an obstacle, requiring it to detect and avoid the obstacle in order to reach the second waypoint. The results were very promising and the failures that occurred were due to obstacles not being detected and not due to the reactive avoidance algorithm.

The paper is well structured and presents its approach in great detail. Not only the theory of the approach is well defined but also the author presents a wide range of real experiments results. A possible limitation of this approach is the computational complexity. The UAV was equipped with (1.83GHz Core 2 Duo, 2GB RAM) system. However, the progress in the field of SoCs (System on Chip) makes these kinds of tasks easier to be executed. Therefore considering that the experiments were held successfully in 2011, using systems which are available today the task would be much easier.

One more paper that presents a flight control system for autonomous UAVs is described in Wang 2014 (Wang et al., 2014). The design is a low-power, small-size autopilot and the research team performed various tests in order to verify its stability and reliability. The hardware used consists mainly of a microprocessor ATmega 2560, GPS, barometer and an MPU unit (gyroscope and accelerometer). In addition, it has a laser distance sensor to achieve a more accurate altitude for automatic take-off and landing. The autopilot system controls a small UAV, having wingspan of 3.8m and powered by a gasoline engine, rather than a multi-copter but the principles of control remain the same.

The design of controller of the autopilot is based on a cascade control method. The outer loop is a trajectory controller; the inner loop is an attitude controller. Among them, the role of the trajectory controller is to calculate the desired attitude and throttle, based on the desired trajectory and current UAV position and height. The attitude controller controls the size of aileron, elevator and rudder, in order to control the attitude of UAV consistent with the attitude calculated by the trajectory controller.

The trajectory controller contains two parts: a lateral controller and a longitudinal controller. Firstly, the lateral controller's role is to reduce cross track error, making the UAV flight along the desired trajectory as far as possible. For this part, they used the L1 control method especially suited in the situation where the desired trajectory is a complex curved path. Secondly, the longitudinal controller tries to eliminate the altitude error. In the meantime, the airspeed of UAV remains within a safety range, avoiding under-speed. For this task, the researchers used the Total energy control system (TECS). The principle of this theory is that

coordinating the throttle and the desired pitch angle of the UAV controls altitude and airspeed.

Further to the model, the paper includes the experimenting results. In general, the small-scale UAV flight presented an accuracy of trajectory control of about 1.5m and altitude error no more than 0.5m. The paper was coherent but it failed to adequately explain the control methods. However, it presented in great extent the hardware that was used but this did not add something to the novelty of the approach they were trying to offer.

2.3 UAV Simulations

In recent years, computer simulation of robotic concepts has become popular since many tools have been introduced to facilitate the designer's work. The advantages of doing a simulation (Richards, 2015) is that it is cheaper and less risky to experiment using computer models rather than real hardware. Regarding the UAVs, there is a plethora of available simulators each of them targeting a specific task. The problem with these is that they complicate the evaluation of the designed tasks and have steep learning curves. On the other hand, there is always the option to develop a custom simulator only for the given task that the designer requires. The downsides of this approach are the amount of time is spent building things from scratch as well as the absence of some advanced features that the simulators have, such as visualization, readymade environments (indoor and outdoor) and advanced dynamics.

For the specific project I need a simulator that can be used in different environments, to assess the efficiency of the collision avoidance algorithms by employing different proximity sensor technologies as well as to prove the stability of the configuration. For these tasks, I need to be able to plan a path, to build different environments and finite states machines, which add the required intelligence to the UAVs. Most importantly, I need to change the UAV model and add additional sensors in specific parts of the UAV. This set of requirements makes the use of available simulators difficult for my task. I reviewed some of them in order to assess their suitability.

The Robot Operating System (ROS) is a framework for writing robot software. More precisely (Quigley et al., 2009), it is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. Gazebo is a robot simulation tool that is used in combination with ROS to test algorithms, robot designs and perform regression testing using realistic scenarios.

The usage of these tools is very popular in the robotics community and it is a standard in many robotics fields. For the quad-rotor UAV systems case there is a published package related to modelling, control and simulation called hector quadrotor (Meyer et al., 2012). This approach allows simultaneous simulation of diverse aspects such as flight dynamics, on-board sensors like IMUs, external imaging sensors and complex environments. In order to exploit them, the user needs to have experience in C++, preferably, or python. In addition they need to be familiar with working in Linux environment and server – client programming. Like many open-source projects, support is provided mainly from the support forums. There are several tutorials, but in my opinion these cover a small and low-level area of required knowledge. In addition to that, there is a wiki but one needs to be an experience user to take advantage of it. One drawback that I also discovered is the existence of many distributions (distros) and the complexity that this adds to the available third-party libraries and packages. On the bottom line ROS and gazebo is a very powerful (fig.2.6), professional and popular tool yet it needs lot of time to become a user-friendly tool and start producing useful results. It has a difficult learning curve if the user comes from different backgrounds.

Very recently (available in Matlab R2015a), complementary to the previous approach, is the release of a new robotics system toolbox trying to resolve some of the weaknesses and bridge some gaps I referred to previously. The system toolbox (Uk.mathworks.com, 2015) provides an interface between MATLAB, Simulink and ROS that enables the testing and verifications of applications on ROS-enabled robots and Gazebo. It also supports C++ code generation, enabling the generation of a ROS node from a Simulink model and deploy it to a ROS network. It is worth mentioning that if the users need to alter and build custom models they need again to cope with ROS architecture and code. Consequently, I admit that this approach is friendlier than the stiff approach of plain ROS but it needs to have knowledge of this in order to be flexible and creative.

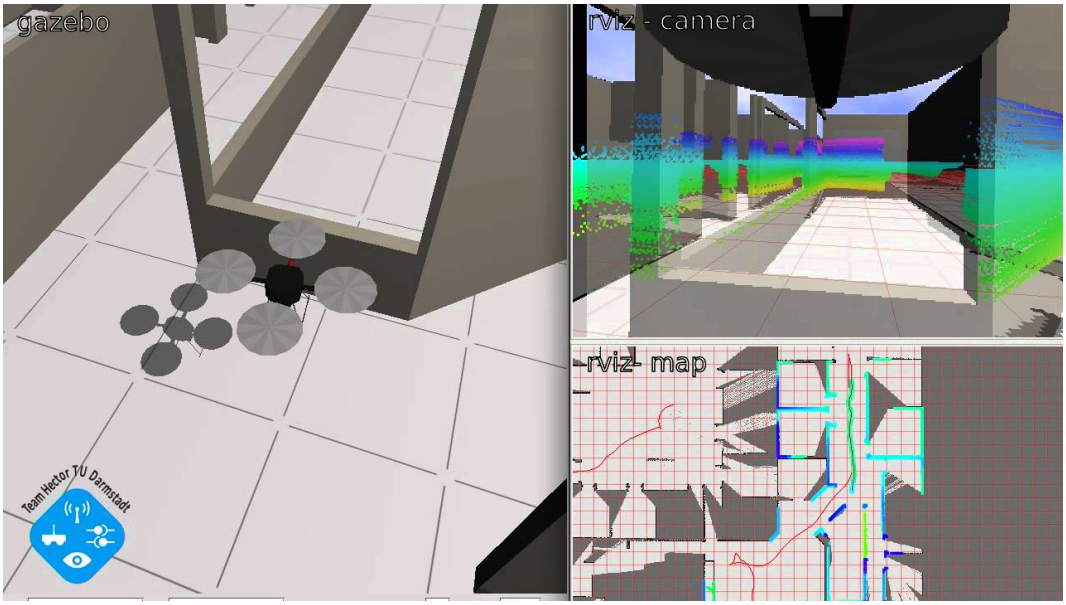


Fig.2.6 Screenshot of an indoor quad-rotor UAV simulation using ROS with rviz and gazebo (image from Meyer et al., 2012)

Another package that I tried was a 3D Integrated Simulation Platform for Indoor Quad-rotor Applications provided by (A.R. Al-Omari, A. Jaradat and Jarrah, 2013). It is built entirely in Matlab without the use of Simulink. It offers impressive 3D views and a nice graphical user interface (fig.2.7). On the other hand, it lacks serious documentation and the code is not well commented. There are no instructions on how to edit it, to add sensors or construct new maps. Unfortunately, in the current state it cannot serve its initial purpose, as stated, which was the ability to implement and inspect algorithms in different scenarios.

A better approach in UAVs simulations is presented in Hartman, 2014 (Hartman et al., 2014). They presented a quite detailed quad-copter dynamic modelling and Simulation package accompanied with extended documentation and instructions. The researchers have built the simulation making extensive use of Matlab and Simulink. Using this tool the designer can define the exact geometry of the dominant parts of the UAV like motors, electronic speed controllers, central hub and arms (fig.2.8) and simulate scenarios with different initial conditions and paths. The documentation does not include instructions on extending the model with various sensors nor does it include the ability to build custom maps with obstacles and to test different behaviours. It is an admirable attempt but, in my opinion, it is intended to different testing scenarios where the designers

experiment with the geometry and the configuration of a UAV rather than have a given UAV and test behaviours in different stimuli.

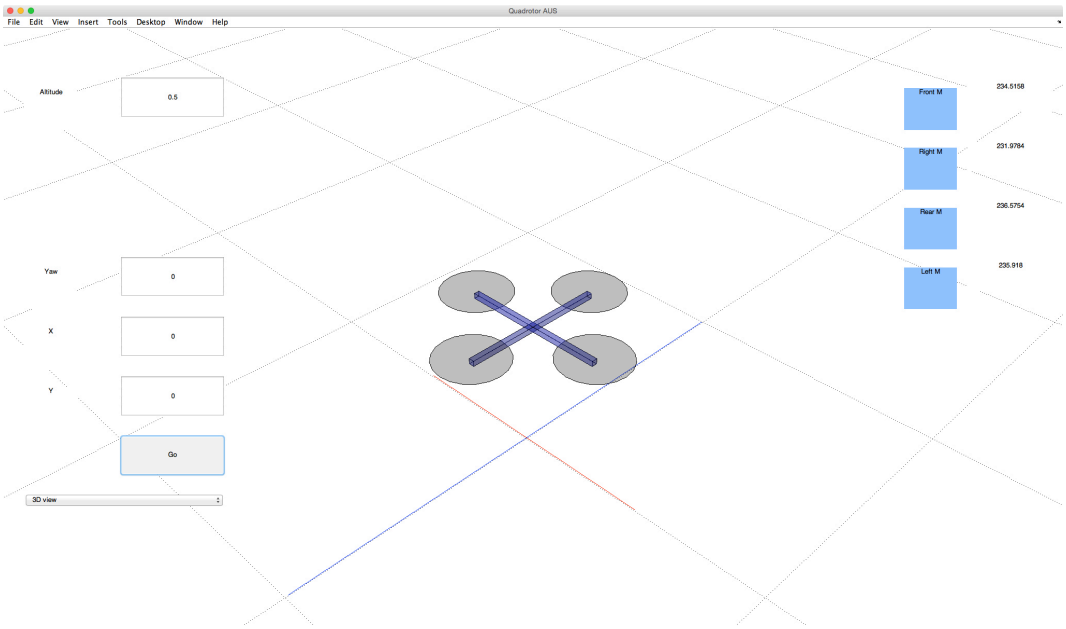


Fig.2.7 Screenshot of the Integrated Simulation Platform for Indoor Quad-rotor Applications (image from A.R. Al-Omari, A. Jaradat and Jarrah, 2013)

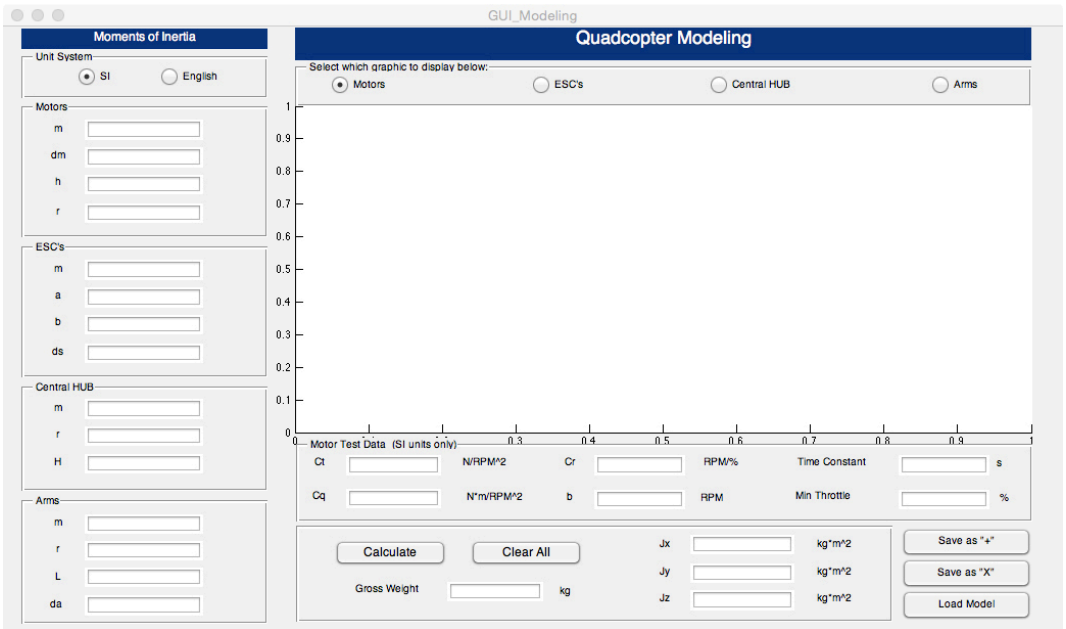


Fig.2.8 Modelling graphical user interface of quad-copter simulation using (image from Hartman et al., 2014)

2.4 Reviews summary

The first section was a review about radiation mapping. This section shed light to an unknown domain since it is not part of the robotics discipline. By examining related papers, I gained knowledge about the procedure, the different methods that can be applied and the required conditions that should be applied in order to get a reliable outcome. Moreover, I explored the configuration and the capabilities of the existing AARM system and I planned individual approaches that I could develop in order to optimise its performance in the domain of radiation mapping.

In the UAV flight section a range of papers -exploring topics such as navigation, localisation, mapping and obstacle avoidance- have been reviewed. In most cases the presented solutions had also been implemented and tested successfully in real experiments. By examining these papers I gained a deeper understanding of the fundamental principles of the aforementioned concepts and I understood under which conditions they could be applied and their related limitations. Furthermore, I was inspired from some specific concepts, such as the reactive methodology approach, using a decision block that performs the assisted tele-operation flight mode, the UAV setup and various collision avoidance strategies. However, in my project it is unlikely that I will reach the stage of applying advanced techniques such as mapping or SLAM since the time is very restricted and I don't have a prototype to apply them to directly and to start experimenting with the emergent behaviour. In real terms, I must set up from scratch both the simulation environment and the real UAV configuration, something that requires lengthy research and experimentation of all available methods, if I am to establish the ones best suited for my purposes.

Finally, I reviewed a number of different simulation methods. I discovered a wide variety of available methods aiming at different tasks. My specific needs will be satisfied when I discover a simulation that is realistic, reliable, and able to easily offer different testing environments and to be close to the real implementation. The best choice, yet hardest to get started with, was the combination of ROS and Gazebo. This approach is very popular among the professional roboticists around the world and it is the one that will be applied in this project as well.

Part 3

Sensors

In this part, a range of different proximity sensors is presented. For each sensor the strengths and the weaknesses are identified and various performance experiments are carried out. Towards the end of the part, two sensor applications are presented which could be potentially engaged to robotic applications.

The schematic representations of the experiments and the related measurement tables are illustrated in a separate section in the appendices, “Sensors Experiments”. Technical details about the circuits, which were used, and information regarding the source code can be found in the section “Sensor Circuits” in the appendices.

3.1 Ultrasonic sensor

Ultrasonic sensors are transducer devices that can convert ultrasound wavefronts to electrical signals. By way of illustration, a sound is emitted by the sensor, then it bounces on the surrounding environment and comes echoing back. This sound takes time to travel distances. The further the distance it needs to travel, the longer time it takes. After that, the sonar’s microcontroller, which keeps track of the time that passes, can calculate the distance of the reflective objects detected and it produces an appropriate voltage to the host circuit.

Devantech SRF02 ultrasonic range finder is a single transducer ultrasonic rangefinder in a small footprint PCB that supports both I2C and Serial interfaces (fig.3.1). According to the specifications, the minimum measurement range is approximately 15cm and the maximum 6m with 3-4cm resolution. Resolution is simply the smallest measurement increment that the device is capable of

reporting. It is the same as the smallest increment of measurement on a tape measure or ruler. SRF02 can report the distance in cm, inches or uS.

Fig.3.1 Devantech SRF02 Ultrasonic range finder

The advantages of this device can be summarized in the followings points:

- Low-cost
- Light-weight
- Low-power consumption: 5V, 4mA Typ.
- The use of serial bus, which can be connected up to 16 devices to any uP or UART serial port, since individual addresses can be allocated.
- There is no need for calibration since it supports full automatic tuning.

The first experiment was to test the sensor by measuring distances perpendicular to a wall (fig.A.1). The analysis of the results (tbl.A.1) leads us to the following conclusions:

- Reflective objects closer than the minimum range are in the sensor's dead zone.
- For distances lower than 2m it has an accuracy of 2cm. For distances longer than 2m it has an error approximately 3 to 5cm.
- For distances greater than the maximum it returns 0.

There is huge variation of the measurements according to the angle in which the reflective objects is hit by the sonar. The sensor cannot accurately measure the distance of an object that has its reflective surface at a shallow angle, as a result the sound will not be reflected back towards the sensor. The other case is when the object is too small to reflect enough sound back to the sensor. The sensor's behaviour becomes clear by examining the beam pattern in (fig.3.2)

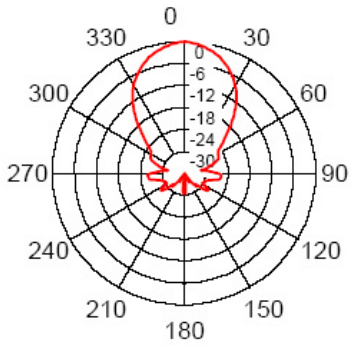


Fig.3.2 The SRF02's beam pattern, showing the sensitivity of the transducer in dB (image from robot-electronics.co.uk)

To experiment with the different behaviour, I performed a range of measurements in different angles (fig.A.2). As it is illustrated by the beam pattern, in angles larger than 30 degrees the sensors sensitivity has already been reduced 8 times. The situation deteriorates for angles larger than 60 degrees where the performance of the sonar sensor is unacceptable (tbl.A.2).

An additional unwanted behaviour that the SRF02 displayed in the experiments was the wrong measurements due to the beam width. As an example, if the sensor is used in a 1m-width corridor, even though the distance in front will be clear and the wall perpendicular to the sensor, the measured distance will most probably be approximately 124cm. This is clearly illustrated in figure (fig.3.3).

I conducted two experiments that verified this behaviour (fig.A.3). In both cases the measured distances were incorrect. The sonar detects the echo by listening for the returning wavefronts. In our case, points that were in periphery and not in the target responded with strong wavefronts and the sonar reported incorrect measurements. Another possible reason could be the phasing effect where the echo does not come from a point source. For example, in the case where a wall is in front of the sensor, not only the central reflection is encountered but also the reflections from the surrounding area are slightly behind the central. The sum of all reflections, which the sensor receives, can be either strengthened or weakened by phasing effects. Therefore, if the echo is fainted then it might be the following wavefront that is detected (Robot-electronics.co.uk, 2015).

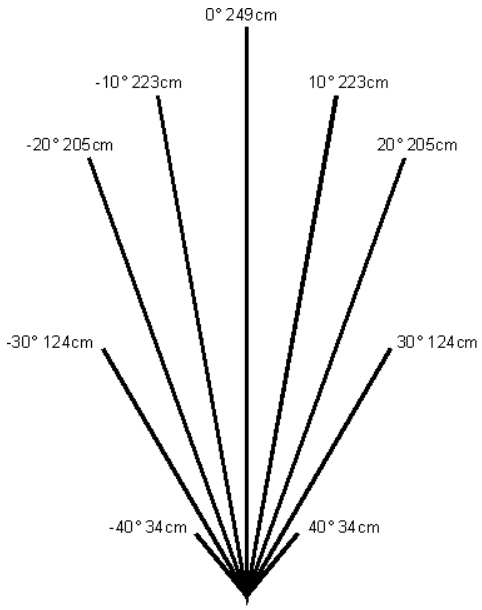


Fig.3.3 Measured beam pattern for the SRF02 (image from robot-electronics.co.uk)

The second experiment using an ultrasonic sensor was to explore its behaviour related to the wind. This is an important factor since it will be used outdoors, mounted on a UAV and the acoustic frequencies might be affected by it. Inside a lab, a simple way to simulate wind was to use a pedestal fan. Under certain conditions, it produces airflow, not too focused but in a certain directions. I tested the setup with the fan, in highest speed, in different positions related to the ultrasonic sensor (fig.A.4). I was cautious enough not to place the fan in a point where it could be identified as an object and thus could alter the results because of the presence of an object and not due to the airflow.

When the airflow was not directed towards the ultrasonic sensor, the influence was minor (tbl.A.3). The interesting behaviour was observed in a position where the airflow was towards the sensor. Whenever the fan was turned on, some of the measurements (approx. 1 every 5) were erroneous. Generally speaking, it seems that the ultrasonic sensor is affected only when the airflow is directed towards the sensor and it is near or inside its sensitivity field. Unfortunately, I couldn't move the fan in many other positions since it was sensed as an object. Moving the sensor to greater distances was not attempted since the sensor itself is prone to side lobes and the measurements would not be stable.

3.2 Infrared sensor

Infrared sensors base their functionality on the emission of light. This sensors use Light-Emitting Diodes (LEDs) as a source of light. The sensor detects a specific light wavelength in the infrared spectrum. Any light with wavelengths longer than 700 nm is called infrared and is not normally visible. The LED produces light at the same wavelength as what the sensor is able to receive. Moreover, it can calculate the distance by measuring the intensity of the received light. For instance, when an object is close to the sensor, the LED light bounces off the object and returns into the light sensor.

The Sharp's GP2Y0A02YK0F infrared sensor (fig.3.4) consists of an integrated implementation of a PSD (Position Sensitive Detector), IRED (Infrared Emitting Diode) and signal processing circuit. It exploits the triangulation method to detect the distance. Because of that, it is not affected by environmental temperature and operating duration.

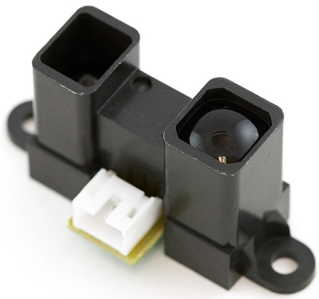


Fig.3.4 Infrared proximity sensor Sharp GP2Y0A02YK0F (image from sparkfun.com)

The key features of the sensor are the following:
- Distance range: 20 to 150 cm
- Analog output type
- Consumption: 5V, 33mA Typ

The conversion between the voltage and the distance is not a straightforward procedure. According to the specifications (GP2Y0A02YK0F, 2006), the measurements' characteristics of the output follow a certain graph given in figure (fig.3.5)

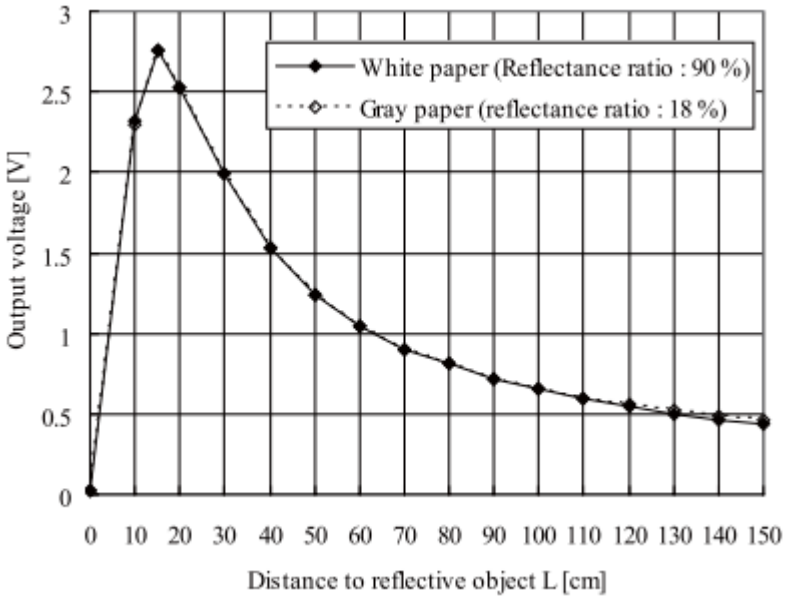


Fig.3.5 The relation of the output voltage with the distance for the Sharp IR sensor (image from (GP2Y0A02YK0F, 2006))

The relation of the output voltage over the distance is not linear. One approach could be to use this graph and attempt to find the best fitting line equation. To facilitate the coding, the output voltage is converted to values from 0-1023, which is the analog reads that the Arduino measures (5v/1024 units). Matlab was used to draw the graph and evaluate the corresponding best-fit equation (fig.3.6). Using a two-term power series model, the estimated equation is the following:

$$V = 3965d^{-0.6257} - 86.46$$

Where V is the output voltage and d is the corresponding distance. An approximate solution of the above equation regarding the distance is the following:

$$d \approx \frac{2.7655 \cdot 10^8}{50(50v + 4323)^{0.593625} \cdot v + 4323(50v + 4323)^{0.593625}}$$

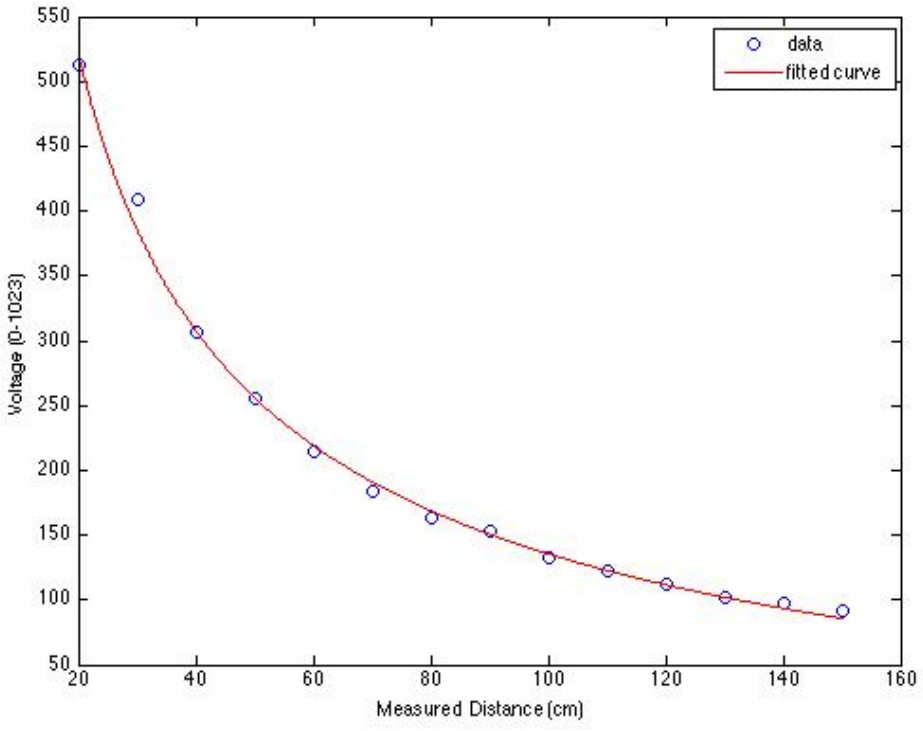


Fig.3.6 Best fit equation for the non-linear relation of voltage and measured distance

Using this equation, a range of measurements was performed. The set up was the IR sensor pointing towards a yellow wall (fig.A.5). The performance was not acceptable (tbl.A.4). The error in distances above 100cm is huge. This behaviour could be justified because of the following reasons:

- The best-fit approximation in the range from 80 to 150cm is not well adjusted. This happened because the Sharp's graph is not so detailed and minor changes in the voltage correspond to major changes in the distance. Consequently the resolution is poor. Moreover, an approximate solution was used in order to calculate the distance from the voltage since the exact form was too complex.
- The specifications mention that it is recommended to stabilize the power supply line, by inserting a by-pass capacitor of 10μF or more between V_{cc} and GND. Unfortunately, this was not implemented.

Similar behaviour to the sonar was observed in the cases where the objects were not perpendicularly aligned with the IR sensor. In addition, an influencing factor was the reflectivity of the object. For example, white paper has a reflectance

ratio of 90% while grey paper 18%. This is particularly important for distances above 110cm where the sensor's resolution is poor.

In conclusion, the performance of the sensor could be improved if a voltage to distance graph made from scratch and not based on the one shown in the specifications. This would facilitate accurate correspondences between the pairs instead of having to guess from the graph, especially in the low-resolution part. In addition, because it is analog, the output is depended of the stability of the input voltage and the flatter is the input the less noise will be present in the output. Nevertheless, the range of the IR sensor is very restricted and since it is not suited to the UAV applications, it is not worth spending more time on it.

3.3 Optical flow sensor

Optical flow sensors are mainly utilized to provide precise measurements of ground speed and therefore the position of mobile robots. They are particularly useful in GPS denied environments allowing long-term dead reckoning navigation.

One of the first attempts to use this type of sensors in the UAVs was based on mouse sensors. The mouse sensor returned the average movement of the visible surface's features. In order to convert these values into real distances, which correspond to the movement, the altitude should be known. By combining the measurement of a barometer or an ultrasonic sensor, this distance can be calculated. Other factors that affect the optic flow are the vehicle's roll and pitch changes. If these rotations have known values, reported from the IMU for example, they can be introduced to the calculations and provide the correct distances covered.

The main disadvantages of using mouse-based optical flow sensors are the following:

- They operate satisfactorily only in well-lit environments. In typical indoor and outdoor low-light conditions, their practical use is limited.
- They need additional devices to be present such as sonar and IMU to have all the required data for the calculations.

- They require a fast MCU (microcontrollers) for the calculations. The flight controller is already used and loaded with the coordination operation of all the other connected devices.

A highly improved optical flow sensor that can support more advanced operation scenarios is the px4flow smart camera (Honegger et al., 2013) (fig.3.7). The main features that improve its functionality can be summarized as follows:

- It has a CMOS image sensor with a 752×480 resolution having a very high light sensitivity.
- It is equipped with an ARM Cortex M4 CPU customized with a special imager bus peripheral for real-time image processing reaching a maximum rate of 250Hz in optical flow calculations.
- It has a high precision on-board gyroscope to compensate angular velocity.
- It is equipped with an on-board ultrasonic range sensor to measure the distance towards the surface and to scale optical flow values to metric velocity values.

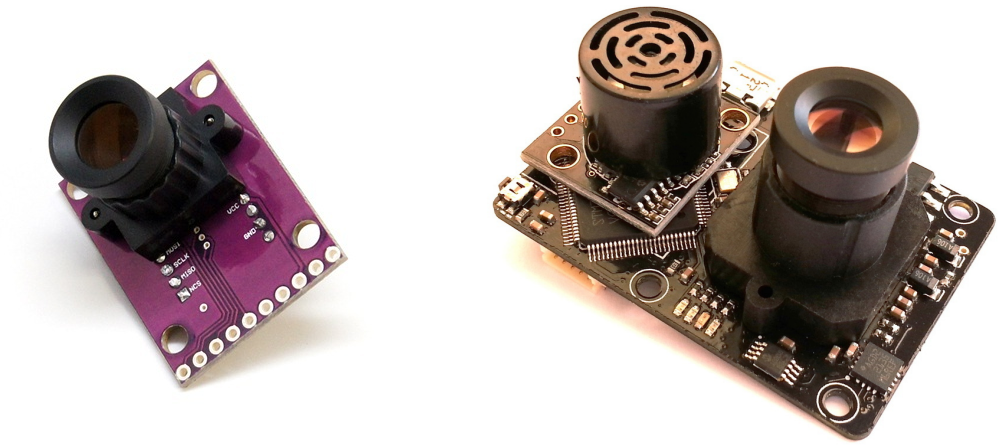


Fig.3.7 Mouse-based optical sensor and p4flow smart camera (images from copter.ardupilot.com)

Up to this point I have explained that the optical flow sensors can measure the ground speed or translational velocity. This is based on the following equation:

$$\upsilon_{trans} = \upsilon_{optic} \frac{Z}{f}$$

where Z is the current distance to the scene, f the focal length and u_{optic} the optical flow. It is assumed that the rotational velocity is ether zero or known, measured by the gyroscope. An interesting aspect of this equation is that it can be used to calculate, with high accuracy, the altitude instead of the translation speed if the latter is known. The ground speed can be known either using the GPS or the airspeed measurements.

Before moving to the experiments regarding the px4flow smart camera it is important to cite some restricting factors of the optical flow sensors in general. The optical flow equation (OFE) is defined as:

$$I_x \upsilon_x + I_y \upsilon_y + I_t = 0$$

where I_x, I_y and I_t are the spatial and temporal derivatives, respectively, of the image sequence I at a given point, and (u_x, u_y) is the 2D velocity at that point. The underlying model on which OFE is based on is that the luminance remains constant between frames along the motion trajectory, so that the total derivative of the image function is zero, hence the OFE. The OFE is one equation with two unknowns, u_x and u_y. Hence further constraints are needed to obtain a solution. The Lucas and Kanade method makes the assumption that the motion field within a small window is constant and this constraint is used with the OFE to construct a least squares problem and hence estimates of the vector (u_x, u_y). All points in the region are assumed to have the same velocity, hence giving multiple equations in 2 unknowns, enabling a solution to be obtained. All in all, the method should work well within textured regions with anisotropic intensity variation, which means changing in all directions, and in which the motion is locally constant and can be approximated as such.

There are various tests that can be carried out using this sensor. They can be divided to two categories, tests related to the ground speed estimates and tests related to the altitude. Unfortunately, for the first category the testing configuration is rather complicated compared to the other sensors. The best way to assess the ground speed is to compare them with data computed using a VICON motion tracking system. Alternatively, the ground speed can also be compared to the corresponding values obtained using an on board GPS unit.

Considering that the calculated ground speed will not be exploited from the radiation mapping UAV and that the testing is time consuming, it was decided not to proceed with this procedure. However, the device will be part of the configuration for the UAV's altitude hold mode and as such is going to be tested for the purposes of the project.

The px4flow device is equipped with a high quality ultrasonic range finder, the Matbotix HRLV Maxsonar EZ4. The specifications cite that it supports one millimetre resolution having a maximum range of 5000mm and a dead zone at 300mm. The special property of EZ4, which makes the difference from the competitors, is the narrow beam width providing the least sensitive to side objects. Consequently, the HRLV-MaxSonar-EZ4 is a very good choice when only larger objects need to be detected since it may ignore some small and medium targets. Another advantage of using the EZ4 is that it provides the most noise tolerant acoustic sensitivity compared to the other ultrasonic sensors of the same category. The Matbotix EZ4 range finder supports three different interfaces, analog Voltage, RS232 Serial and Pulse Width. However, since the sensor is part of the px4flow, the connection has to be accomplished through this. As it is described in the special section "Sensors Circuits", I utilize two individual ways to access the data, using an Arduino microcontroller and ROS packages.

Similarly with the Devantech SRF02 sensor I conducted experiments to verify the accuracy of the sensor. The setup of the experiment was the sensor facing a wall in the perpendicular direction (fig.A.6). The measured distances were very precise (tbl.A.5). Above a dead zone of the 30cm, the performance was excellent. A very interesting aspect in the behaviour of the sensor was the narrow beam width. I repeated the same corridor test that I had done with the SRF02 (fig.A.7). It is very impressive that the EZ4 sensor was measuring the correct distance up to the value of 350cm.

3.4 Laser sensor

Laser sensors belong to the high-end devices of the proximity sensors. They are more expensive than others but they offer higher ranges with enhanced accuracy. Unlike infrared LED sensors, the semiconductor laser emits light from

a very tiny area in only one direction and at one wavelength or a few closely spaced together.

PulsedLight's LIDAR-Lite laser is an innovative device. It offers competitive features in low price, compared to other laser sensors, and with the ease of I2C or pulse width modulation (PWM) connectivity. According to the specifications, it supports up-to 40-meter range capability with 2.5cm accuracy, 1cm resolution and fast measurement rate equals to 100Hz. In addition, it has a small size, low power consumption and it is lightweight (fig.3.8).

In the manufacturer's site it is stated that the success of the LIDAR-Lite is based on the specific implementation. They use a customised version of the Time-of-Flight distance measurements approach, originating in military applications. These systems transmit relatively high power infrared pulses using a laser diode. The distance estimation is accomplished by measuring the time delay between the transmitted pulse and the detected return pulse. In general, it is a long-range operation with low accuracy. In order to improve the accuracy of these systems they had to use signal processing. The challenge was to avoid complex and expensive hardware to achieve this, as other existing systems implement it. PulsedLight manage to develop a signal processing technique using a simple approach without the complexity and cost of direct analog-to-digital conversions (Technology Brief, 2015).

Fig.3.8 LIDAR-Lite Laser Rangefinder (image from sparkfun.com)

The sensor experiments were conducted in various distances and angles. In the first tests where the reflective objects were aligned perpendicularly (fig.A.8), the performance followed the specifications, showing accuracy close to 1-2cm

(tbl.A.6). In greater distances like the 10m, I should admit that even my verification method, using a tape measure, was error-prone so the accuracy is negotiable but in favour of the device. Especially for the 10m distance measurement, it should be mentioned that it was conducted in a corridor (1.1 meters wide). Consequently, in contrast with the sonar, where the wide acoustic lobe is problematic, laser sensors have a concentrated beam and perform uninfluenced of the location's shape.

Another useful aspect that emerged from the tests was the behaviour regarding the material of the object and the hit angle (fig.A.9). The experiments showed that glass degraded the performance only when the beam was hit in an angle (tbl.A.7). On the other hand, when the material was concrete the 'hit angle' was not an influencing factor.

To conclude with, LIDAR-Lite showed an impressive performance when compared to the previous tested sensors. It has a long range, good accuracy and the shape or the properties of the surrounding environment do not influence its behaviour.

3.5 Applications

An interesting and creative task is to use the sensors to build more complicated and useful devices, which can be used in localization and navigation applications in robotics. The most versatile and promising sensor to use from the previous presentations is the LIDAR-Lite.

The LIDAR-Lite is a reliable laser sensor having a high reading rate and accuracy. It can be used as a simplified version of a lidar that essentially provides laser scans. To leverage the full features of the LIDAR-Lite, I needed to combine it with a microcontroller and a servomotor. The most popular choice was to utilize an Arduino Uno with a small yet classic servo in inexpensive robotic applications Hitec HS-422 Deluxe (fig.3.9).

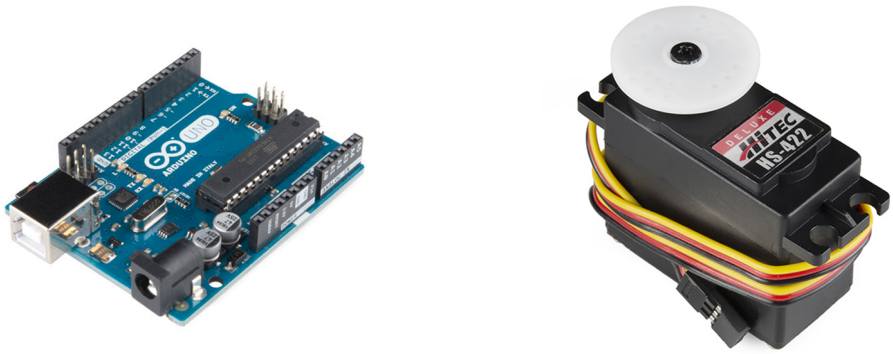


Fig.3.9 Arduino Uno and Hitec's HS-422 Deluxe (image from sparkfun.com)

3.5.1 Mapping application

One of the most common applications of a laser scanner is the mapping. To map the surroundings, I need a servo to rotate the laser in continuous angles and measure the related distances. In order to accomplish this task the laser was mounted on the servo using a specific bracket (fig.3.10). This mechanical construction was pretty stable although the integration of 3D printed housing could offer a more compact design for easy use in robotics applications.

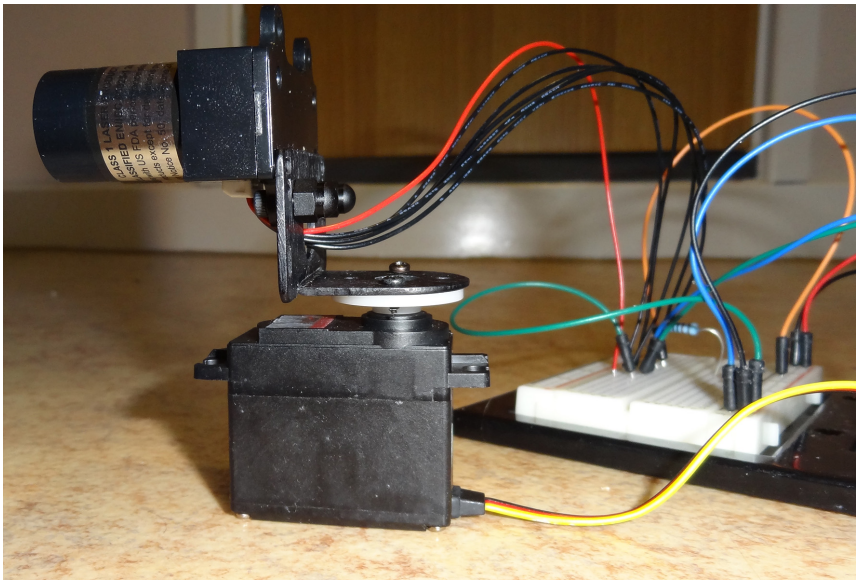


Fig.3.10 Mapping device consists of a servo, a mounting bracket and a laser sensor

The structure of this task consists of two working parts (fig.3.11). For the first part an Arduino sketch has been developed which is responsible for the following functions:

- To read the angle of the servo
- To measure the related distance. More precisely, it gets two measurements and provides the mean value of them so as to avoid a fair amount of noise.
- To send both data to serial port with a rate of 9600bps

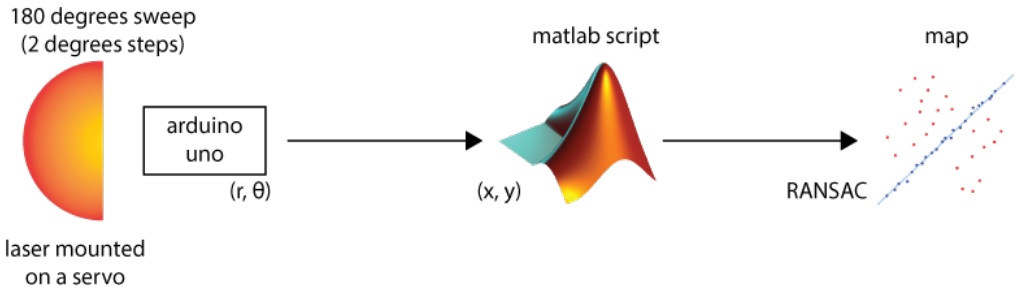


Fig.3.11 The logic flow of the proposed mapping algorithm

This part is attached to the hardware and utilizes the microcontroller to gather the useful information and transmit it to the other end for processing. The servo rotates the laser forward and backward by 180 degrees. Every 2 degrees it stops and performs the tasks I have previously described. Consequently in a loop, the total number of readings is equal to 180, the first 90 taken during the anticlockwise transition from 0 to 180 degrees and the remaining 90 during the clockwise rotation.

On the other side, a Matlab script has been developed as a client. This script takes inputs from the serial port using the USB, which is connected to the Arduino Uno. Particularly, the script accomplishes the following tasks:

- Reads the angle and the related distance
- Filters values outside the LIDAR-Lite range
- Converts from polar coordinates to Cartesian, x and y
- Feeds the points to a RANSAC method to detect potential lines in the data.

This script exploits the measurements to yield useful patterns in the space that could be the base of a map. It makes use of the random sample consensus (RANSAC) algorithm (Fischler and Bolles, 1981), which is an iterative method to

estimate parameters of a mathematical model from a set of observed data, which contains outliers. The assumption is that the data consists of inliers and outliers. By inliers I mean data whose distribution can be explained by some set of model parameters. However, inliers are expected to be noisy. On the contrary, outliers are data that do not fit the model.

For the estimated lines the following balance should be kept:

- Enough lines to describe details of the environment. Consequently, the total points should be divided into subgroups and feed the algorithm individually.
- Elimination of repeated lines of the individual subgroups of points that give the same equation. In order to avoid that, the implemented approach discards the lines which have less than 5 degrees difference in the direction.

A sample of the mapping output is illustrated in (fig.3.12). In this case the mapping device has been placed in the middle of the room. Since the device covers 180 degrees, the output should look like a π (pi). The outcome of the mapping approach is similar to the expected one for this simple case.

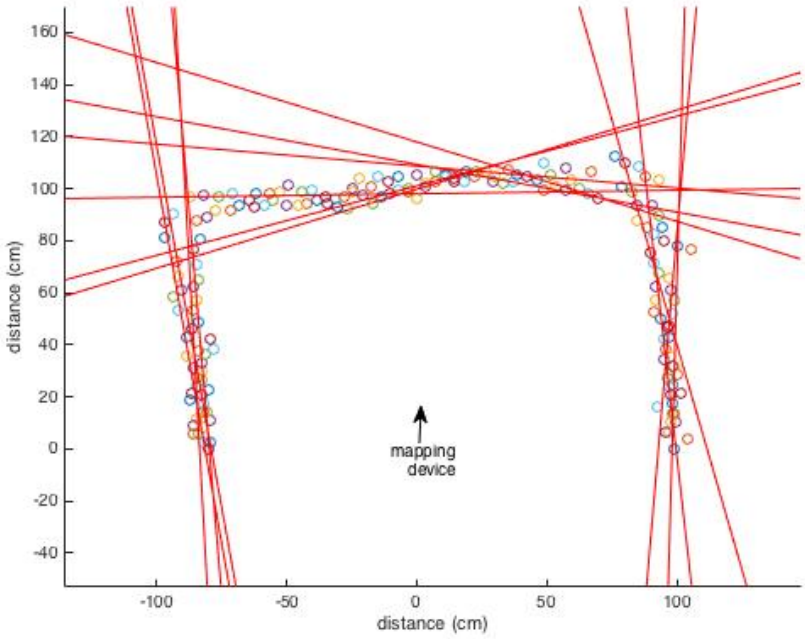


Fig.3.12 The setup of the experiment and the outcome of the mapping algorithm

3.5.2 Proximity application

For this section, and after having implemented a simple mapping algorithm, I wanted to develop a ROS-enabled application ready to by employed in any compatible system. My initial plan was to deploy a LIDAR-like application where the laser sensor would continuously rotated and transmitted, via a topic, an appropriate sensor message of type *sensor_msg/laserScan*. Although I put a lot of effort into it, this was not feasible, mainly due to the instability of the system. More precisely I encountered the following problems:

- The Arduino memory was 80% occupied by loading only the required libraries (*ros.h* and *i2c.h*), leaving limited space for the working sketch.
- The power source was overused since both laser and servo were fed from the 5V output of the Arduino Uno.
- The servomotor under complex and tedious continuous translation was drifting sometimes and in addition it couldn't provide an absolutely stable cycle to compose the laserScan ROS message.

The combination of the laser sensor and the servo driven from an Arduino Uno controller was not adequate to support such a complex and precise task and to

provide the expected reliability. As an alternative, since I wanted to develop a working ROS-enabled sensor, I utilized the existing hardware configuration to deploy a proximity sensor. This sensor provides three distances, the front and the sides (0^o, -90^o and 90^o). To improve the accuracy, the laser sensor measured the distance twice in each position and returned the average. In my attempt to implement more complex scenarios, like sweep ranges of 20 degrees for every main direction, I faced drifts and synchronisation problems. Consequently, I kept the design simple and reliable for the given hardware.

One of the great features that ROS provides is the ability to utilize in robotics systems simple and cheap sensors using the Arduino microcontroller. It exploits the fact that the Arduino is connected to the computer using a serial connection and data is transmitted using this port. Providing a specific interface package called *rosserial,* it allows transmission and reception of ROS messages by coding Arduino IDE sketches with including the ROS library. The entire flow of the information from the laser sensor to the published topic is summarized in the following figure (fig.3.13)

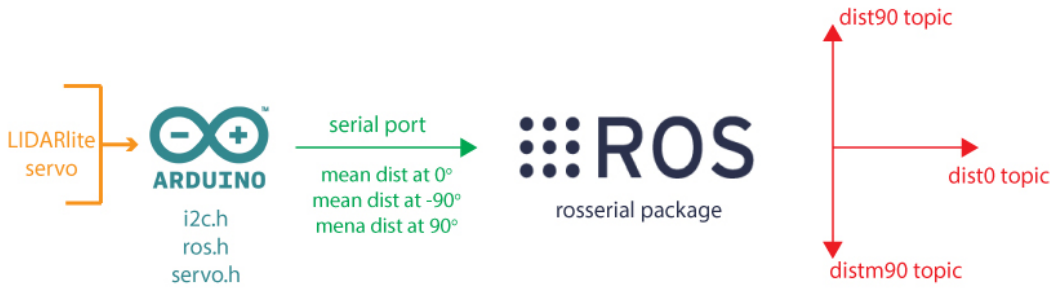


Fig.3.13 The flow of the information for the proximity application algorithm

To validate the implementation, I tried out the device at the end of a corridor (fig.A.10). By performing various experiments I verified that the measured distances lie inside sensible margins (tbl.A.8). It is clear that the deviation from the case where I tested the laser sensor in a static setup was greater but still usable in proximity check approach. One additional issue that I noticed was the degradation of the accuracy when I increased the servo speed. This had occurred not because of the laser sensor, since it had a high operating rate (~100Hz), but mainly because of the instability and drift of the servo.

By following this implementation I created a ROS-enabled device that could act as a laser proximity sensor. It is a feasible system since the housing of the device can be easily designed and composed in a 3D printer. In terms of the microcontroller, there is no need for an Arduino Uno since an Arduino Nano v.3.0 (45mm length and 18mm width) can effectively do the work. This is because both of them are based on the ATmega328 chip, which provides 2 KB of SRAM and 1 KB of EEPROM. The circuit with the Nano board was implemented so as to test the behaviour and the stability in a real context (fig.3.14). In essence there was no difference in the behaviour compared to the Arduino Uno. To conclude, it is possible to have a compact and simple design that can report three accurate distances in a reasonable rate.

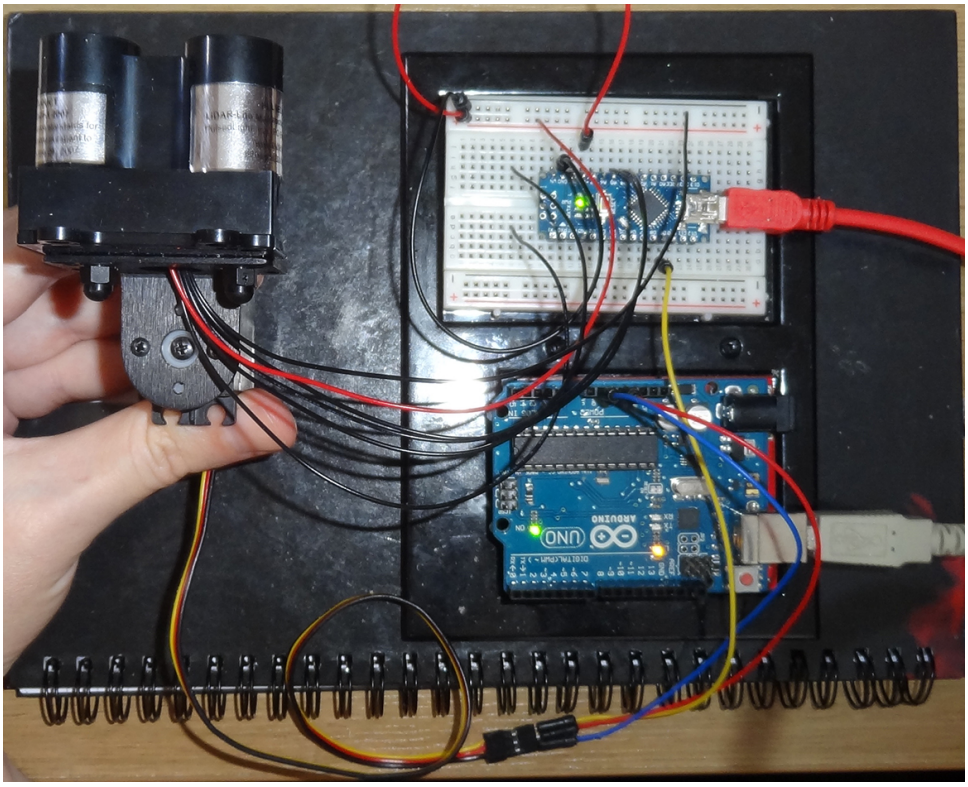


Fig.3.14 The proximity application implemented using an Arduino Nano board. In this picture Arduino Uno feeds only the V_{dc} and GND pins of the servo

Part 4

Simulation

4.1 Introduction

Simulation in Unmanned Aerial Vehicles is a very broad topic. It mainly depends on the application which is under testing. There are different simulators if the operators need to learn how to fly a UAV compared to the ones that are useful for the engineers to help them design an effective chassis. In my project's case, I need a simulation in order to investigate the behaviour of developed applications in certain circumstances. The key features that I needed were the following:

- A realistic UAV simulation in terms of the physics and dynamics that applied.
- Ability to utilize different types of sensors and customise their specifications.
- Ability to build custom landscapes and test the behaviour of the algorithms for various different cases.
- Be as close as possible to the structure of a real UAV in terms of the flight controller, to the input commands for navigation and to the individual devices from which it is built, like the global positioning system (GPS) or the inertial measurement unit (IMU).
- The ability to apply the developed algorithm to a real UAV with minor modifications.

After an extended research of many available approaches, all the aforementioned requirements lead us to a specific solution, which is the combination of ROS and Gazebo. Some of the solutions that were explored but were rejected were the following:

- Building a custom simulation from scratch using Matlab. During the second semester there was a related unit that gave us the opportunity to build a simple UAV simulation. However, it was assumed that the UAV moves in 2D and it did not simulate the flight dynamics of the UAV. Generally speaking, the simulation was intended to test UAV swarm

robotics algorithms. Consequently, the effort to build a simulation in this way from scratch was a time consuming procedure with uncertainty about the effective fulfilment of the requirements.

- Available simulation found on Internet based on Matlab and Simulink. There are several simulations of this type and a possible approach could be to adapt one of those and customise it. Most of them have been built as master or part of PhD theses. The disadvantages of using them are that they are primarily focused on simulating the dynamics and they do not include sensors. Moreover, they do not offer a build in feature of designing realistic landscapes. In general, depending on the quality, most of them have poor comments on the code and most importantly they do not offer support. Consequently, this solution was ineffective since it may have meant debugging others' code rather than building creative tasks.
- There was the possibility to use Webots (Cyberbotics.com, 2015). This is a development environment used to model, program and simulate mobile robots. It offers the choice of using a wide range of simulated sensors and actuators. The robot behavior can be tested in physically realistic worlds. It is a popular tool for research, well documented and continuously maintained for over 18 years. The reason why it was not adopted in my research was that it is not free of charge. It offers a 30-day trial version but after this period a valid license should be acquired.

In short, various approaches were considered for simulating a UAV executing different behavioral algorithms. The most promising and professional way seemed to be using the combination of ROS and Gazebo. Following this method in order to carry out my experiments finally proved a successful decision considering the outcomes of my research.

In the "Simulation Background and Configuration" appendix there is a section related to the general description of ROS, Gazebo as well as a terminology section. In addition, there is a configuration section that includes useful guides in order to allow the easy setup of the simulation systems, regarding both hardware and software. A technical description that explains the implementation details of the obstacle avoidance application can be found in the "Simulation Development" appendix.

4.2 UAV package

When using ROS and Gazebo in robotics simulation what the roboticist want to avoid is the reinvention of the wheel. Therefore having these tools installed and working, the users could utilise packages that implement UAVs equipped with all the required features. Depending on the application, there are UAV implementations available which have many common characteristics but also differences that help us when decision making. For my project, I explored the following three packages:

- Tum simulator

This package (goo.gl/ZBjaAY) contains the implementation of a Gazebo simulation for the Ardrone 2.0 (fig.4.1). It is a nice simple quad-rotor simulator developed at the Technical University of Munich. The simulator supports both Ardrone 1.0 and 2.0 versions but it is optimised for the latter. It has a built-in support for joystick manipulation, which means that the users can use it to fly the real drone as well as the simulator. The Tum-simulator is equipped with 2 cameras, one front and one bottom and a height sonar sensor.

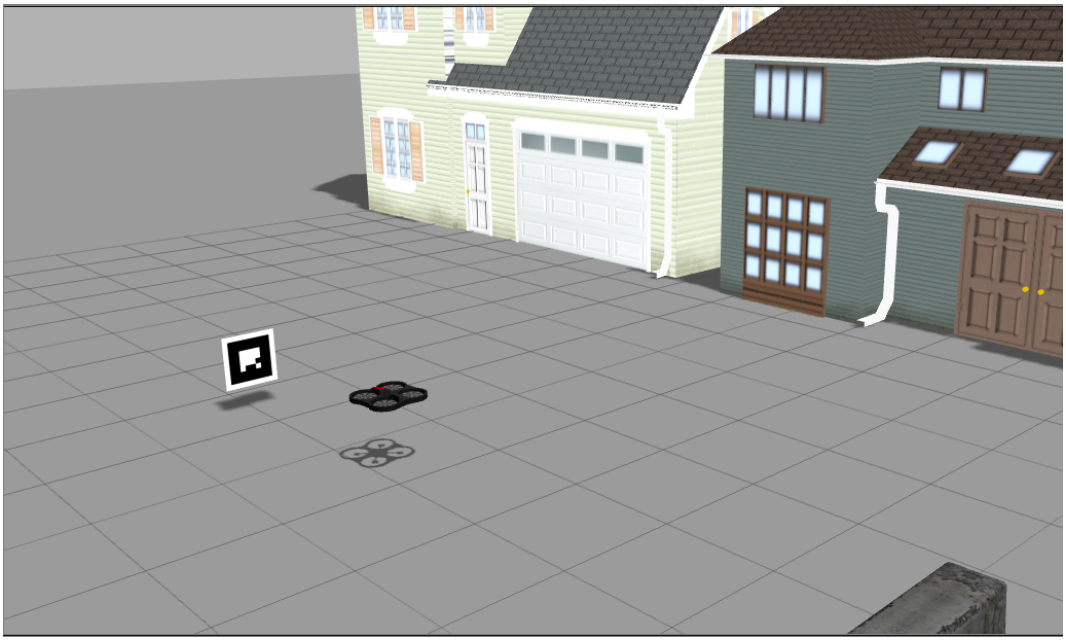


Fig.4.1 Tum simulator in action (image from answers.ros.org)

There are two reasons why this simulator was not adapted in this project. Firstly, it is executable in ROS Fuerte released in 2012, four versions older compared to

the current. After this version, ROS changed completely the low-level built system macros and infrastructure (catkin). Consequently, I wanted to avoid working with an outdated framework, which would downgrade all the other tools to ensure compatibility. Secondly, the provided sensors are not suitable for my project. This simulator is more focused on computer vision applications where the users capture video from the cameras and process data, using for example OpenCV, to make navigation decisions. There is always the choice of adding or developing custom sensors but it is safer to find a simulator that already integrates them.

- PX4 SITL

This simulator developed by PX4 Autopilot Project for the software in the loop simulation (Pixhawk.org, 2015). The ROS SITL setup allows the users to run certain PX4 Firmware modules, which use the multiplatform wrappers within ROS. These applications publish data in the related ROS topics. The PX4 simulator is compatible with ROS Indigo, which is a very recent ROS version. Likewise the previous, it allows joystick manipulation. It is noteworthy that includes two UAV simulators, one based on Ardrone and the other on Iris (fig.4.2).

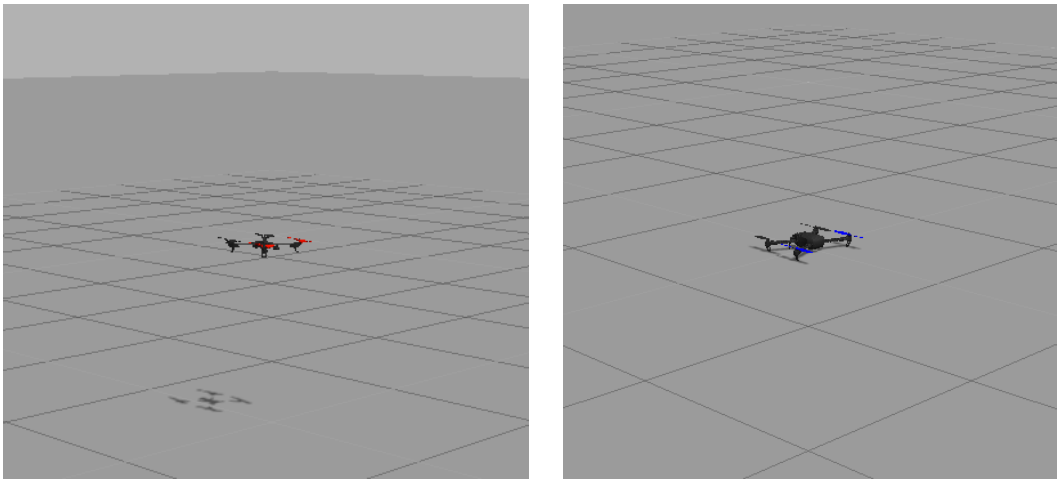


Fig.4.2 PX4 SITL simulation using Ardrone and Iris UAVs in action

The additional characteristic that this simulator includes, compared to the others, is the MAVROS (Micro Air Vehicle ROS) package. This package provides a communication driver for various autopilots with MAVLink communication protocol. In addition, it provides UDP MAVLink bridge for the ground control station. Using this protocol ROS can send specific commands to the UAV, which

is essentially what this simulator is specialised for. To illustrate this, using ROS in a companion computer, navigation commands can be sent directly to the flight controller produced by a robotics algorithm via MAVROS. This is called off-board control. This simulation is intended for those cases where the users want to test the MAVROS commands before applying them to the real UAV.

Since it is actually a port of the PX4 flight stack in ROS, it is a high fidelity simulation coming with plenty of capabilities like real UAV. For example, in order to begin it is required to arm. Some of the available features provided are:

- Inertial measurement unit (IMU) which provides linear acceleration, angular velocity, atmospheric pressure and altitude
- Compass which provides heading
- GPS which provides longitude, latitude and altitude
- PX4flow optical flow sensor which is integrated to provide ground distance and ground speed
- Manual set of the motor speed for each individual motor

There are reasons why I did not select this simulator for my task. Firstly, it has some problems when I used it in combination with the RViz, the ROS visualization tool. This is the all-seeing ROS visualisation utility. This tool becomes very important when there is a need for customisation of the sensors since it visualises the laser scans and the ultrasonic lobes. Secondly, it did not include laser or ultrasonic sensors and my preference was to have them integrated to avoid misconfigurations because of my short experience. Thirdly, that fact that it is equipped with MAVROS and includes highly detailed UAV configuration, it introduces an unwanted complexity for performing behavioural analysis of navigation algorithms. As I already explained, from my perspective the PX4 SITL is focused on different tasks compared to my requirements.

- Hector quad-rotor

The Hector quad-rotor simulator is a framework that implements the simulation of quad-rotor UAVs employing ROS and the Gazebo simulator (fig.4.3). It has been developed by the Research Training Group in Darmstadt Technical University (Meyer et al., 2012). The dynamics model of the quad-rotor has been parameterized using wind tunnel tests and validated by a comparison of simulated and real flight data. It is available for all the latest ROS versions from 2012 and on (Fuerte, Groovy, Hydro and Indigo).

Hector quad-rotor comes with a wide range of available sensors. It is equipped with an Inertial Measurement Unit (IMU) measures the angular velocities and accelerations of the vehicle's body in the inertial frame. It provides ultrasonic sensor to measure the distance from the ground in a specific range. In addition, it has a Magnetic Field sensor to measure the heading of the UAV. A GPS receiver is also installed to provide pseudo range measurements. Aside from the standard devices, it also offers additional and more sophisticated ones. More precisely, it provides the Hokuyo UTM-30LX as laser scanner, a depth camera like Kinect and a normal camera. Furthermore, like all the other simulators, it is possible to manipulate it using joystick (like x-box controller) as well as keyboard.

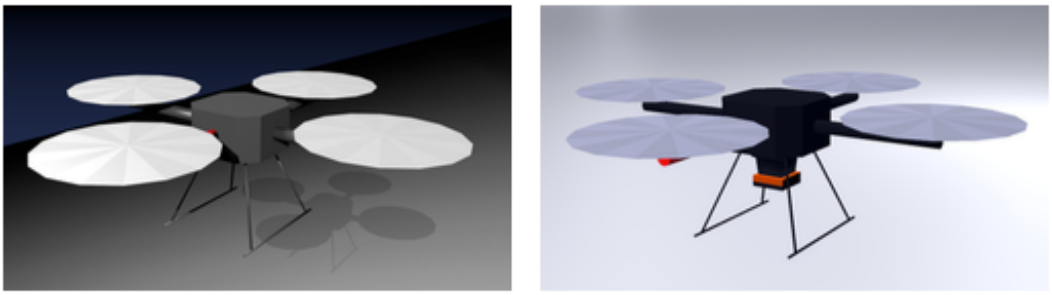


Fig.4.3 The Hector quad rotor model with a Hokuyo UTM-30LX laser scanner mounted in the second picture (image from (Meyer et al., 2012))

The Hector quad-rotor is delivered as a ROS stack, which means that is a collection of packages and not only a single one. So, aside from the packages that are responsible for the modelling, the control and the Gazebo simulation of the UAV, it also includes a visual SLAM (Simultaneous Localisation and Mapping) package to demonstrate the great potentials to build robotics applications.

Hector quad-rotor combines all the types of the sensors, which I wanted to employ in my simulation in a well-structured package, as complex as it is required for a comprehensive realistic simulation. Everything is located in the place where I expected to find it. This enabled me to customise the configuration and move it closer to my real setup. It is also well documented with many related questions in the ROS support forums. My working system is configured in ROS Hydro version with all the related tools working flawlessly like RViz. The same implies in the ROS Indigo as well. All in all, from my experience with working with it, it is a

reliable and stable tool that helps to focus on the details of the developing robotics applications leaving away all the other distracting points.

4.3 UAV configuration in simulator

The rich features of Hector quad-rotor package were utilized for simulation in the current project. In order to build a UAV close to my working scenario, I needed to refine the sensors positions and operation. The selection and the settings of the sensors were based on the available equipment I had for the real UAV.

From the available sensors provided in the hector UAV I utilized the following:

- IMU

  The IMU was exploited to report the heading of the UAV (yaw angle). An alternative method to obtain this value was the Magnetic Field Sensor (the compass).
- Sonar

  The ultrasonic sensor transmits short ultrasound impulses. Sonar reports the distance corresponding to the first echo returned from the ground or any other object within its FOV (Field Of View). The comprehensive way that the simulator was built allows modifying all the related parameters of the ultrasonic sensor. More precisely, in the file (goo.gl/RCZ1u8) the users can change the following characteristics:
  - Update rate (in Hz)
  - Minimum and maximum range (in m)
  - The name of the published topic
  - The position, the orientation and the parent frame link in which is attached

  If the users want to experiment with more advanced characteristics, using the parameters inside the file (goo.gl/daIwSo), they can alter parameters such as resolution, Gaussian noise, moments of inertia and many more. In this project the sonar was used twice. Firstly, an ultrasonic sensor installed under the central hub, facing the ground, to report the distance from the ground – that is the altitude. Secondly, an ultrasonic sensor installed in the frame arm oriented to the back in order to measure the distance between the UAV and

any object in the back. To add more devices in the UAV configuration, the users should edit this file (goo.gl/XJU7TU) accordingly.

- Laser scanner

  The Hokuyo UTM-30LX Scanning Laser Rangefinder is a great device having features like 270° area scanning range with 0.25° angular resolution at a speed of 25msec/scan. However, it is very expensive with a price approximately 5000$. In my project the principle of using relatively cheap COTS technology did not justify the use of this device. Therefore, I did not use this device as it is. The simulator structure allowed me to alter the characteristics and degrade it in a simple laser scanner. Using this (goo.gl/ZzL6HI) file I could change:

  - The minimum and maximum angle from 270^o to 180^o.
  - The ray count from 1800 (in the 270^o range) to 180 (one scan per degree for my 180^o range)
  - And the update rate.

  Likewise, the users that want to experiment with advanced features of the laser can alter the related parameters in this file (goo.gl/FhyLzY).

Aside from the devices that I tested and utilised, I additionally experimented with the following:

- GPS

  The GPS reports longitude, latitude and altitude measures. There are two useful ways to exploit these data. The first is using an approximation, to estimate the distance between two points. This accomplished by converting the (lat, lon) coordinates to meters. The second is to use the altitude. However, in my case it was not accurate, as it is expected. Furthermore, I couldn't rely on that value in order to hover constantly in low height.

- Barometric sensor

  This sensor measures the air pressure as the primary means for determining altitude, also called Pressure Altitude. Like with the sonar sensor, the reported height is utilised to lock the altitude during the flight. However, if the air pressure is changing in the flight area due to extreme weather, the UAV will follow the air pressure change rather than actual altitude. In reality when the altitude is less than 5m, the sonar is the main device of obtaining the altitude and not the barometer. In the simulation,

the measurement of the barometer in 1m above the ground was even a negative number. Consequently, this sensor was not used in the simulation.

- Front camera

  In the default configuration of the Hector simulator a front camera is enabled. I did not experiment in depth with this device because I did not have any intention to use computer vision as part of my application.

The complete setup of the simulated UAV is shown in the following figure (fig.4.4). In this screenshot, a hector quad-rotor UAV equipped with a laser scanner and two sonars is visualised in rviz.

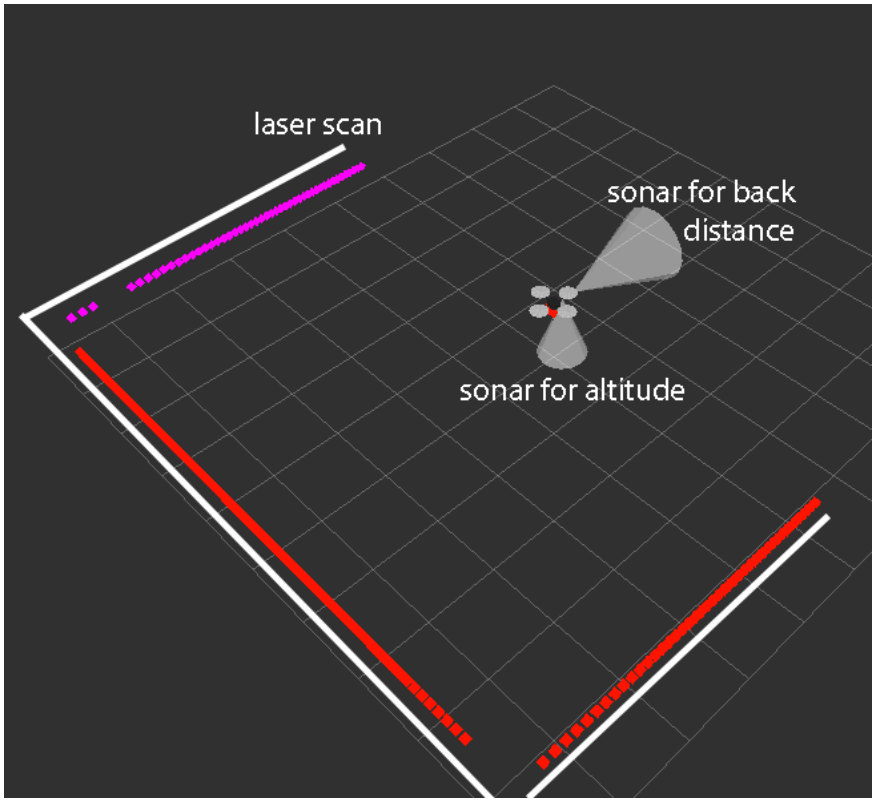


Fig.4.4 Visualisation of a simulated UAV using the ROS rviz tool. This screenshot illustrates the two sonar lobes and the laser scanning

4.4 The radiation mapping and simulation requirements

It is important to explain clearly the task of UAV taking radiation measurements so as to gain better understanding of the simulation. In a real UAV for radiation

measurements, a gamma spectrometer is also installed in addition to the proximity sensors. Based on the radiation mapping techniques, to obtain high quality measurements, the UAV should fly close to the inspected surface and in a constant distance. This means that the UAV should fly in low height above the surface and this distance should be locked. In the real UAVs there is flight mode for this case called Altitude Hold Mode.

A desired feature during the radiation mapping is the obstacle avoidance capability. To illustrate this, as the operators control manually the UAV, if this gets near an obstacle, in the direction of its movement, an assisted behaviour will be performed. Thus, if the UAV finds an obstacle in front of it, it will assess the surrounding environment and accordingly it will accomplish an avoidance manoeuver. At the end of this intervention the position of the UAV should be pointing to the open space and in the predefined locked altitude. After this point the control returns to the operators who are free to continue without worries till the next one. Simultaneously, if for any reason the UAV sense an object around it that is closer than one meter, it will proceed to emergency landing.

The described obstacle avoidance algorithm was implemented in the simulation environment to test its behaviour and get an assessment of its effectiveness. The source files created for the obstacle avoidance tele-operation algorithm has been grouped in a ROS package called *my_hector* (goo.gl/DpgSre).

4.5 Obstacle Avoidance

The heart of the obstacle avoidance algorithm is the finite state machine (FSM). This allows adding intelligence to the UAV since it is able to switch between different behaviors according to its sensors readings.

It is important to describe the scenario of the simulation. In this simulation, as in the real UAV as well, the operator controls the flight manually using a joystick or the keyboard (fig.4.5). If the UAV encounters an obstacle in front of it or senses something closer than 1m around it, then the control passes to the obstacle avoidance algorithm. From this point, the UAV based on the FSM will perform a single or a series of manoeuvers. When it assess that there is no obstacle to

avoid, it will pass the control back to the operator. After the avoidance procedure, the algorithm will try to deliver the UAV in the initial heading that this had before the algorithm took the control. In complicated cases, when the UAV performs a sequence of maneuvers this could not be possible. This description corresponds to an assistive tele-operation solution where the application helps the operator by performing the avoidance manoeuvers.

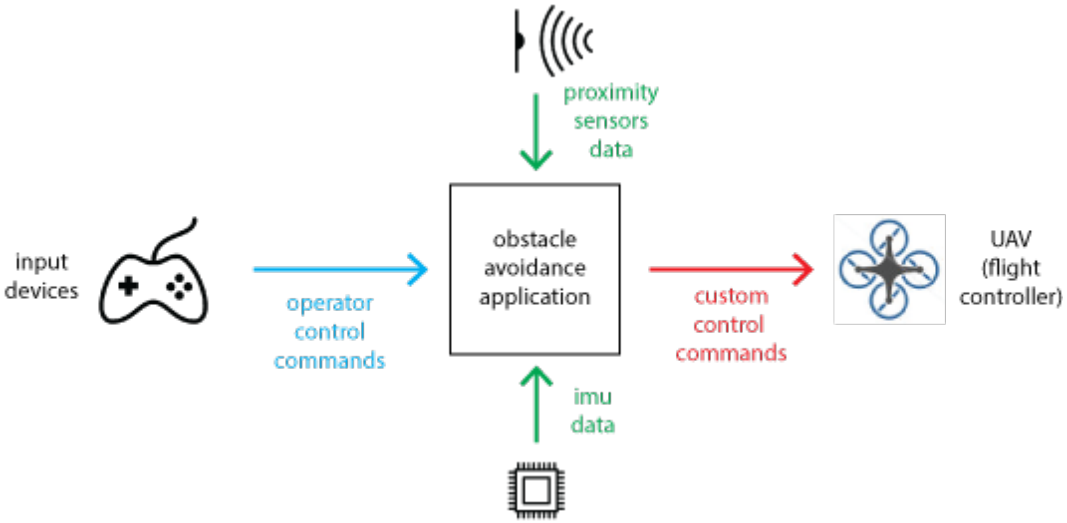


Fig.4.5 The schematic representation of the obstacle avoidance operation topology

At this point I should clarify the way that the UAV measures the distances that were used in the algorithm. The height and the back distance are measured using two ultrasonic sensors pointing towards the ground and back accordingly. For the distances in front and on both sides the laser scanner is used (fig.4.6). More precisely, having configured the laser to take measurements every 1-degree in 180^{o} range, it is possible to choose these directions from the related positions of the distances in the reported scanning. Consequently, the front distance is given by getting the minimum distance in the range from 70^{o} to 110^{o} heading. The left is the minimum distance in the range from 0 to 20^{o} and the right from 160^{o} to 180^{o}. This approach is closer to the real UAV since I do not have laser scanner sweeping of 180^{o}. Instead of this, I have an individual a single laser sensor taking measurements for every perpendicular direction.

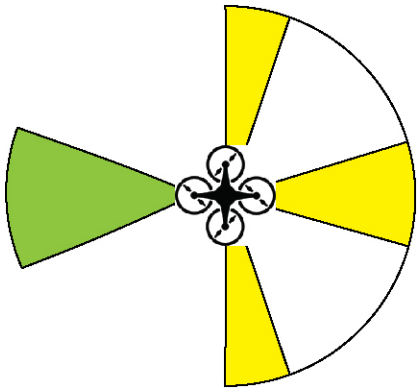


Fig.4.6 UAV proximity sensors. The sonar lobe in the back (green) and the segmented laser scanning in front and on the both sides (yellow)

Having talked about the finite state machine (FSM), a detailed explanation includes the description of the individual states and conditions that should hold to move between them (fig.4.7). The loop that executes the FSM is repeated at a constant rate of 20Hz.

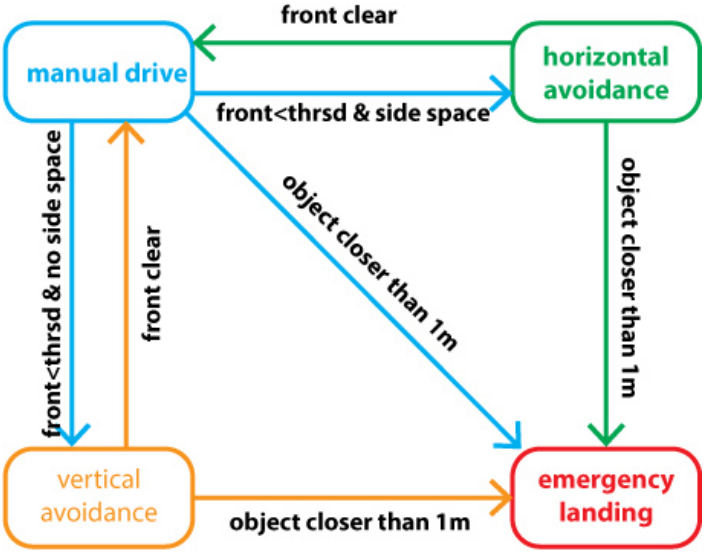


Fig.4.7 The UAV's finite state machine

State 0 – manual drive

In this state, input devices such as a keyboard or joystick control the UAV. Every time this state is executed, a safety check is performed. If the distance in front is less than a defined threshold a transition should occur. The destination of the transition depends on the distances on the left and the right of the UAV. There are the following possible cases:

- The distance on both sides is less than a threshold. This means that the UAV is at the end of a corridor, like a dead end. If this is the case then it changes to state 2.
- The distance on the left is greater than the one the right or vise versa. In situation like this, the UAV seems that flies near a corner. In this case the UAV will change to state 1 by rotating 90^o to the side where the open space is sensed.
- The distance on the left is equal to the distance on the left. This happened when there is no obstacle on both sides and the laser reports its highest range, or when there are obstacles further to the maximum range. In this case the UAV will randomly turn on one side or the other side and change to state 1.

Finally, if the UAV senses an object in distance closer to 1m at any direction (front, back, sides) then it performs emergency landing in state 3.

State 1 – Horizontal avoidance

In this state the UAV has already sensed that there is space to move in the one or the other side. Before entering this state a 90^o change of heading has been performed during the transition from state 0 to the side that has the greatest open space. The UAV starts to move forward and continuously measures the distance on the side where the obstacle exists. If at one point it is noted that the distance gets higher than a threshold, this means that there is no obstacle any more. This signs the end of the obstacle but not the end of the maneuver. After that the UAV will move a bit more and it will correct the heading back to the initial one which it had before the override. During the aforementioned procedure two simultaneous safety checks are executed at the same time.

- Since the UAV moves till it finds the end of the obstacle, it always measures the distance in front of it. If it finds that there is another obstacle ahead of it, it repeats the avoidance maneuver again and continues, maintaining the state to 1.
- If at any point, for any reason it senses an object closer to 1m on the proximity directions (back, front, sides), it changes to state 3.

At the end of the procedure, if the avoidance maneuver completed successfully, the UAV moves back to state 0.

State 2 – Vertical avoidance

In this state the UAV has already discovered that it is at the end of a corridor. There is no point to explore the sides for horizontal escape. At this point, it will attempt a vertical overcome of the obstacle. Depending on the height of the obstacle in front of the UAV there are two available options:

- The first case implies if the obstacle's height is lower than the sonar's range. In this situation, the UAV will start to elevate and simultaneously measure the front distance. At a point before it reaches the upper limit, the UAV will sense that there is space ahead. Consequently, at this point it will stop the vertical translation and it will move forward to overcome the obstacle. After a certain predefined distance, the UAV will move back to the altitude hold position and the control will return to the operator in state 0.
- The second case happens when the UAV reaches the limit of the sonar range. At this point I do not want to move further up and obtain the altitude measurement using the barometer or the GPS. These devices are not so accurate and the altitude hold mode becomes unstable. So if this is the case, the UAV will stop the vertical translation and it will change heading by 180^{o}. After that, it will move back to the predefined altitude hold position. Eventually, the UAV control will return to the manual operation in state 0. Similarly to the previous states, it is possible to change towards the emergency landing state 3 if any object is sensed closer than 1m.

State 3 – Emergency landing

During the movement of the UAV in manual mode or even when it performs avoidance manoeuvers a safety check is concurrently run. In this check the UAV gathers the measurements in every perpendicular direction (0, 90^{o}, 180^{o}, 270^{o}) and if it encounters an object closer to 1m it will proceed to emergency landing. Since the altitude is locked to a low height this solution is workable to enhance the safety of the UAV and the surrounding environment. The landing was preferred because it is difficult to trust a subtle manoeuver with a hazard so close. However, if there is time for exhaustive testing, a more complicated behavior could be developed. In this emergency situation the control doesn't return back to the operator and it requires the physical intervention in the location of landing.

The source file in which the obstacle avoidance application has been implemented is located in (goo.gl/eG2mXn). It is a ROS node that acts as a central hub since it is subscriber to all the sensor and input control devices, publisher to the velocity control topic and client to action servers related to the heading and altitude.

4.6 Testing

During the development of the code the proper debugging took place in order to produce a functional application that will allow me to experiment with the behavior of the model. At this point, I exploited the dynamic features that Gazebo simulator offers to the users. There is online an available library of objects which can easily be utilized to construct a landscape. These objects are declared in an XML-like file by defining the exact positions and orientations. In my case, the wall object was an ideal choice since I could place it in various combinations and explore how the UAV reacts (fig.4.8). The specific file in which the landscape is defined can be found in this location (goo.gl/vfi9NH).

In the following figures the scenarios of some of the experiments, which have been conducted, are illustrated (fig.4.9), (fig.4.10), (fig.4.11) and (fig.4.12). The experiments were successful and the behavior of the UAV was stable and as it is expected to be. There is available online video of the experiments in this address (goo.gl/JyTzGp).

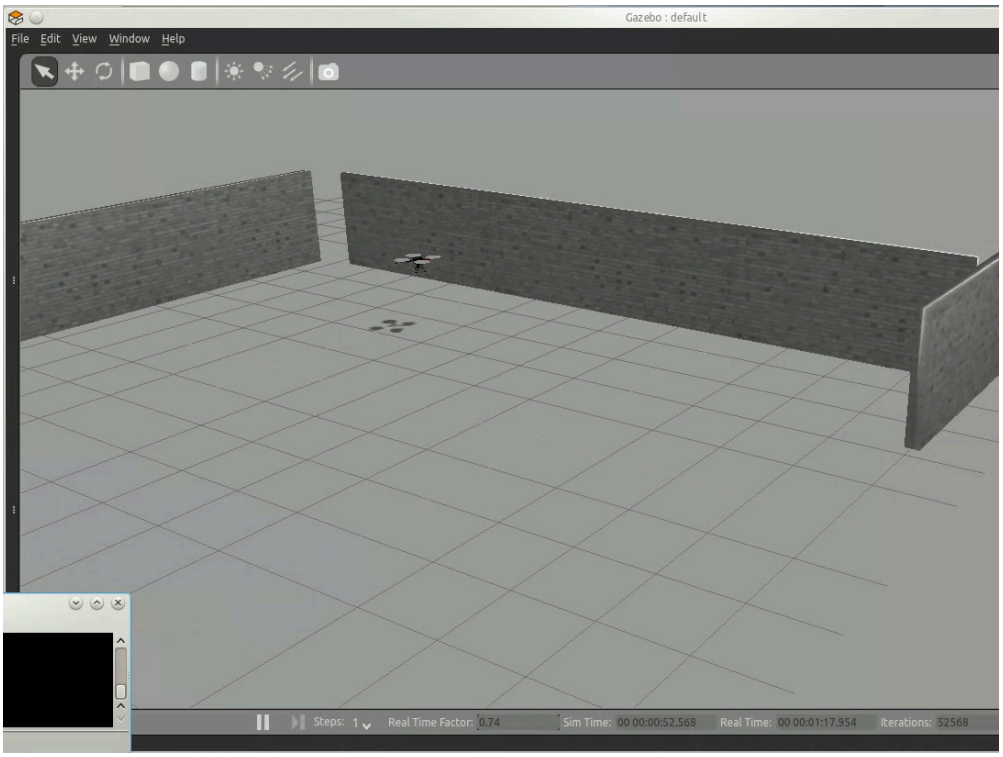


Fig.4.8 Screenshot of the Gazebo simulation during an obstacle avoidance test

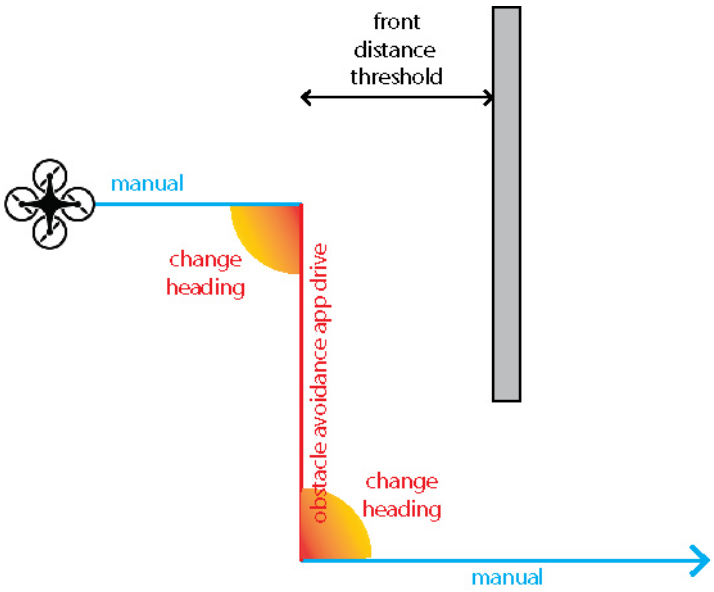


Fig.4.9 The avoidance maneuver when there is no obstacle in both sides of the wall

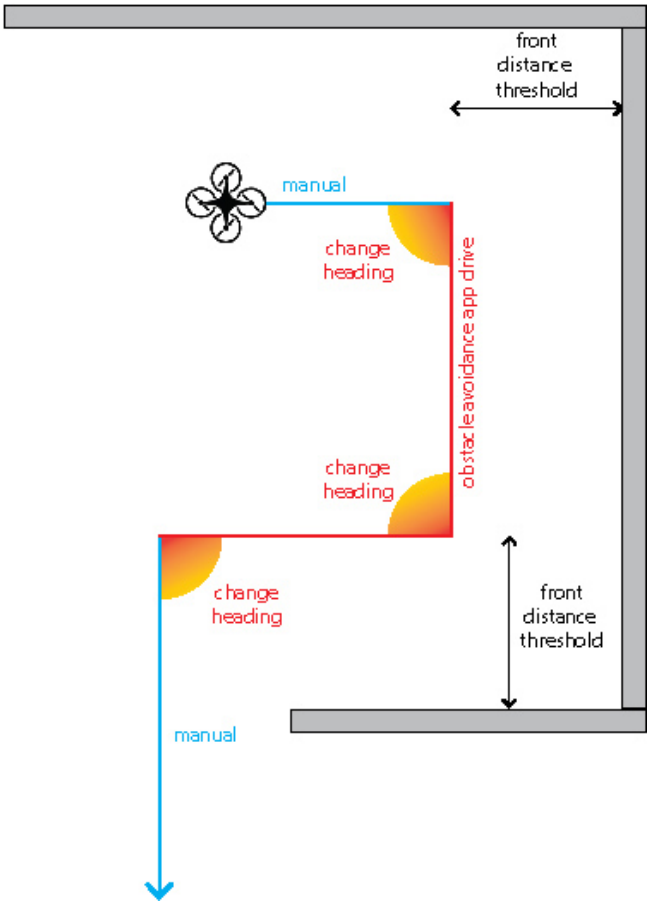


Fig.4.10 The avoidance maneuver when there are multiple obstacles in the path of the UAV. Two avoidance maneuvers in sequence.

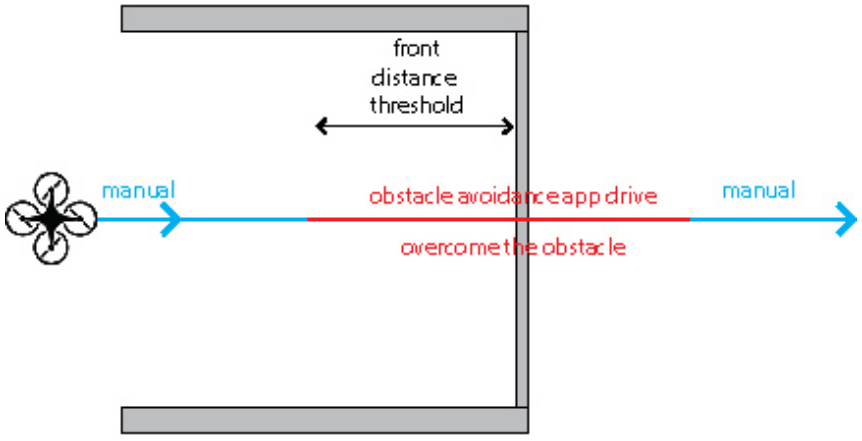


Fig.4.11 The UAV moves towards a dead end. The obstacle in front is lower than the maximum range of the altitude sensor. The UAV overcomes it and returns to alt hold.

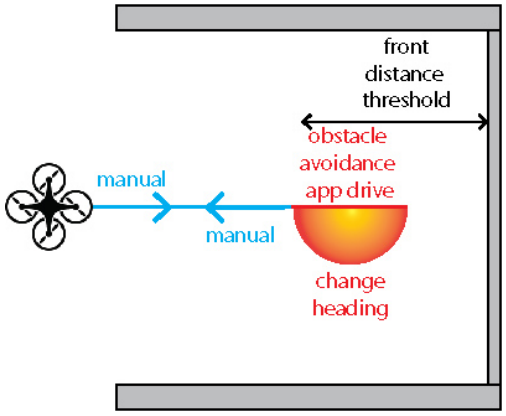


Fig.4.12 The UAV moves towards a dead end. The obstacle in front is higher than the maximum range of the altitude sensor. The UAV changes heading by 180^{o} and returns to alt hold.

Part 5

Human Robot Interaction

The design and development principals of the two mobile applications which have been implemented in the context of this project, are presented in this part. Both applications aim to enhance the user's experience in the field of human robot interface. The first is related to the visualisation of the radiation mapping and the second to the tele-operation of a UAS. Technical description and details about the development of the applications are explained in the "Mobile Applications Development" Appendix. Presentation of the functionality of each application can be found in the "Mobile Applications Functionality" appendix.

5.1 UAV Tele-operation application

One of the popular and traditional methods of tele-operating a quad copter is using a Remote Control (RC). It is a well-established and tested method, which has its origin before the era of the UAV. Nowadays, advanced UAVs are equipped with many sensors and specific purpose equipment such as cameras or radiation spectrometers. An alternative method of getting high bandwidth data from a UAV, which is also convenient, is using telemetry. This method uses a client application, installed in a ground computer, in order to present all the data transmitted from the UAV.

The approach, which has been developed for this project, is based on the utilisation of smart devices. These devices are equipped with powerful CPUs, extreme visualisation capabilities and a wide range of connectivity options. Needless to say that they are very popular and almost everyone in the developed countries has one of them.

The communication between the iPhone and the UAV is accomplished via wireless connection (Wi-Fi). I have chosen this method because TCP/IP is the

native communication protocol supported in ROS ecosystem and a wireless connection will be available in the next generation AARM system since it is going to have a companion computer.

Since I chose the communication method the next step was to decide the format in which the data should be encoded in and exchanged between peers. The selection of JSON (Javascript Object Notation) is a commonly accepted method. JSON is a lightweight data interchange format, easy for people to write and read yet, at the same time, it is easy for machines to parse and generate. In addition, there is a wide variety of available libraries that can be utilised to provide apparent and reliable implementations.

A. UAV Peer

In the side of the UAV there is one important requirement that has to be satisfied: UAV should be ROS-enabled. This is not restrictive because ROS is a very popular and trustworthy action plan in robotics' communities. Moreover, as it has been implemented in this project for safety and convenience reasons, it is possible to use a simulator for the development. The advantage here is that in ROS systems the same code can be used with a real robot without any change.

In a ROS system there is a messaging mechanism called topics (fig.5.1). The data from the installed on-board devices which are publicised in relevant topics and control commands related to the linear and angular velocities are also accepted from the appropriate receivers using associated topics accordingly. These topics should be available to the iPhone in order to have access to the data and control commands. At this point a specific ROS package comes to play. *Rosbridge* (Crick et al., 2011) which provides a JSON API to ROS functionality for non-ROS programs. This sounds ideal for the requirements of my project. *Rosbridge* provides a wide range of supporting interfaces. For my case, it was important that it included a WebSocket server for web browsers' interaction. The WebSocket Protocol is a network protocol designed for the web that enables browsers to have bi-directional communication with a server. Using this package, communication mechanisms can be provided to the non-ROS systems following a ROS-style approach. It supports topic publishing and subscribing, service request and response as serialized JSON objects.

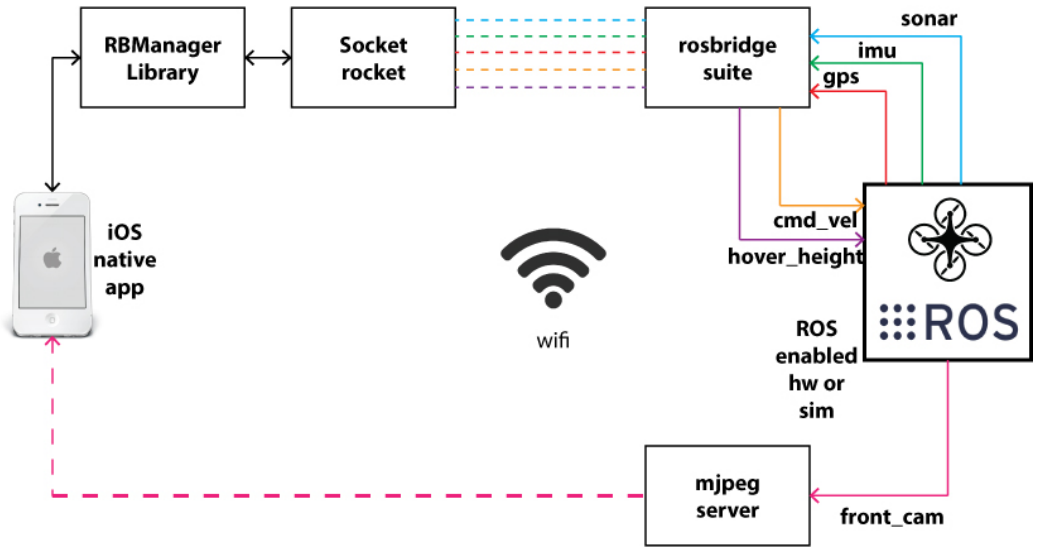


Fig.5.1 Topology of the tele-operation application

One of the advanced equipment that a UAV could carry is a camera. It is extremely helpful and interesting to have real-time streaming of the video to the application. While *Rosbridge* allows streaming videos, seeing as it is just another message type from ROS, I preferred to use a different component. *MJPEG* (Motion JPEG) server (Pitzer, 2011), which is a ROS package, optimised for efficiently transferring stream images in binary format. Consequently, an additional communication channel was used to increase performance and enable clients to request specific video quality and frame size according to their needs.

B. iOS device peer

On the other side there is a native iOS application developed using Objective C. As with the ROS side, the application should leverage the appropriate libraries to gain access in the data generated in the ROS ecosystem as well as to send compatible data. For this case, a *WebSocket* library should be used. *SocketRocket* is an objective C WebSocket client library (Lewis, 2015) useful in native mobile applications. By using WebSockets, existing tools built for the web can be easily reused for the mobile applications. It is similar to how most mobile applications reuse HTTP for their APIs. One level deeper in the architecture, there is the powerful *RBManager* library (Goodhoofd, 2014) for communicating between iOS and ROS. This library is a wrapper for the *SocketRocket* WebSocket library in iOS. This approach follows the standard publisher,

subscriber and service call architecture to send packets compatible with the *Rosbridge* protocol. It is well structured and easily expandable. It allows adding new message types and building on top of that. It has a simple syntax as long as the developer knows ROS.

Regarding the processing of the streaming video the case is simpler. The stream of the MJPEG server is essentially a video sequence in which each frame is separately compressed as a JPEG image. In the iOS library there is a *UIWebView* component (Developer.apple.com, 2015) available that enables the embedding of web content in iOS native applications.

5.2 Radiation mapping application

This mobile application constitutes an important part of this project since it is directly related to the radiation mapping visualisation. Up until now the radiation mapping is a procedure which is conducted offline. To illustrate, the operator could carry a hand held measurement device that stores the data in a SD memory card. Upon the completion of the procedure, the researcher could access this memory card by inserting it in a memory reader using a computer. The desired requirement at this point is that the researcher will be able to get the status of the radiation measurement unit and the measurements displayed on a map during the procedure in real-time.

The communication between the iPhone and the measurement unit is accomplished via Bluetooth low energy (LE). BLE is a wireless personal area network technology. It provides considerably reduced power consumption while maintaining a similar communication range with the classic Bluetooth. In addition, using BLE I avoided the process of requesting a specific license from Apple, called MFi ("Made for iPhone/iPad/iPod"), which is required when the classic Bluetooth is employed.

The operation of the measurement is based on an Arduino IDE sketch developed at Interface Analysis Centre (IAC). This sketch was used as a base to build the server-part of my application. The architecture of the sketch, as it was built, was to enable the GPS connection to get the longitude and latitude and then to obtain

cps measurements via the USB protocol from the spectrometer. When all the required data were available, the sketch was forming a data entry to the text file stored in the SD card. My task was to maintain the working part while simultaneously integrating the required parts for the iOS application.

The first addition for the measurement unit was to install the BLE shield. After that, the sketch had to change in order to access desired bits of information, form compound packets and send them to the link. The formation of specific data packets was a demand since the payload of the transmitted package has a maximum size of 20 bytes for the given BLE shield (fig.5.2)

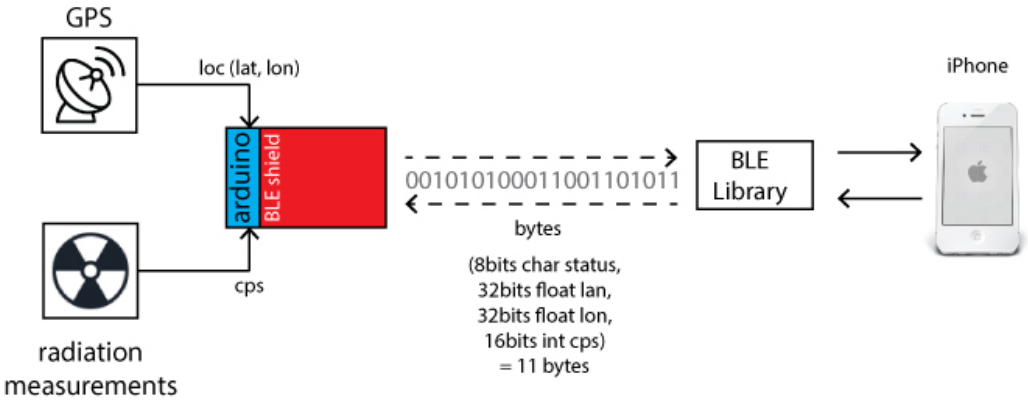


Fig.5.2 The overview of the radiation mapping scenario

According to the application's specifications the status of the GPS should be reported. To save space in the messages payload, I encoded the status information in one byte. Consequently, since the GPS lock is an on-off condition, it needs one bit. The remaining 7 bits of the byte are used to report the number of connected satellites. After that, the latitude and the longitude should be represented using 32bits floating-point numbers. To manage this, I break each value to four parts of byte size and I assemble them again in the iPhone side. Moreover, the radiation counts per second is encoded as a two bytes integer. Finally, having formed the payload of the BLE message (fig.5.3), this message is transmitted to the other side at the end of every cycle.

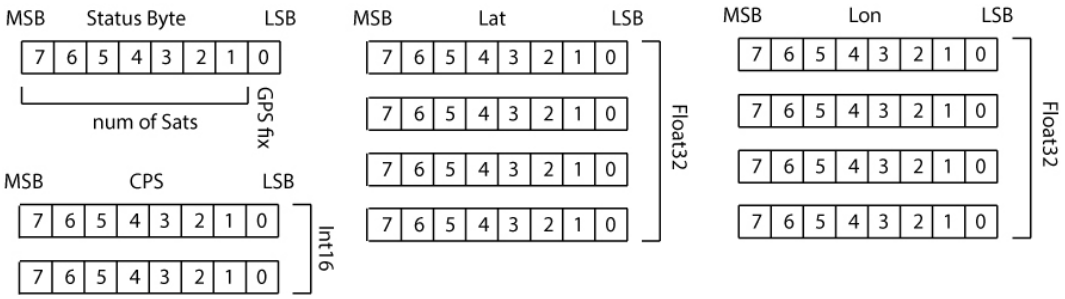


Fig.5.3 The payload of the BLE message

On the other side there is an iPhone device with the Bluetooth connection enabled. In order to decode the transmitted messages, a specific library is needed. The manufacturer of the BLE shield has released an appropriate library for this task, called BLE. Using the provided features, the iPhone can search for compatible BLE devices to connect and disconnect with them. When the iPhone has connected to the measurement unit, there is the possibility to measure the RSSI level (Received Signal Strength Indication) and access the contents of the payload. Using the reverse procedure, the iPhone script assembles the primitive data types from the raw bytes and visualises the information using various means.

Part 6

Real UAV

In the previous parts of the dissertation, a range of various concepts has been presented. The challenge is to combine these approaches and experiment with their behaviour on a real UAV. The synthesis of individual approaches into a working entity is not an easy task. To facilitate the procedure and increase progressively the complexity of the task, the exploration of the real UAV has been partitioned into two stages. The first is the basic approach and the second, a suggested advanced implementation.

A real UAV consists of various individual components. In the "UAV configuration" appendix there is a description of the relevant components and guides to avoid usual problems like vibrations and electrical magnetic interference. Additionally, technical details about the implementation of the basic and the advanced approach are presented in the "Real UAV development" appendix.

6.1 Basic approach

The basic approach is based on the detection of obstacles in the surrounding area of the UAV. This information is illustrated in a mobile application. The working scenario is that the operator controls a hex-copter (fig.6.1) via a radio transmitter and has real-time feedback about the distances around the UAV. To improve the usability, the operator simply checks colour-encoded regions around the UAV on the screen instead of actual distances, since it is very difficult to control a UAV and at the same time read distance values. Having this information, the operator can identify dangerous situations and keep away from obstacles in its proximity. Another possible utilization is to fly in an almost constant distance close to buildings. The latter is also a requirement for obtaining high quality radiation measurements.

Fig.6.1 Hex-copter with the custom measurements unit installed as it was used in the of the in basic approach

The set-up of the system consists of two parts (fig.6.2). The first part is a modified version of the proximity applications that I have presented in the "Sensors" parts. This main component of this custom device is the LIDAR-Lite range finder mounted on a servomotor. This structure is controlled by an Arduino Uno microcontroller. On top of the Arduino, a Bluetooth shield is installed. The laser sensor is rotated continuously in the three main directions, -90^o, 0^o and 90^o. In each direction, the sensor measures the distance twice and reports the mean value. With every new value, the microcontroller composes a specific message and transmits it via the Bluetooth link. The entire mechanism is installed inside an appropriately formed plastic box (fig.6.3). On top of the box, a special aluminum construction with a semicircle shape mounts the device bellow the central hub. Moreover, a Li-Po battery is attached under the device's box.

The second part of the system is the client application that can be executed in iOS smart devices. The messages sent via Bluetooth are received as input by the application. Every time a new message is received, the related value as well as the specific coloured area are updated accordingly. The user can manually set the value of the distance's threshold in which the application will identify an obstacle as danger. This distance can be different in front and at the sides.

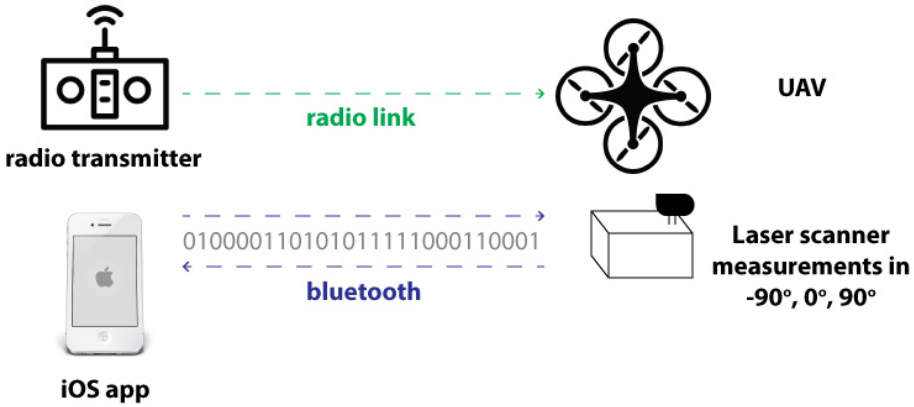


Fig.6.2 An overview of the obstacle detection system

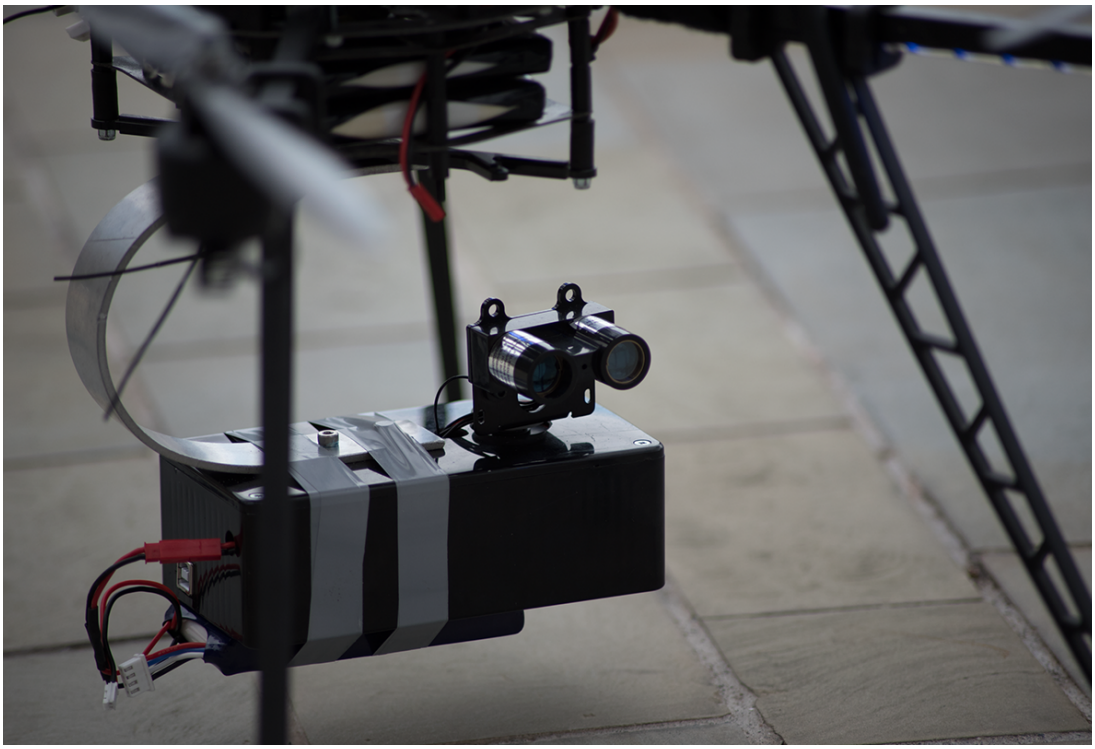


Fig.6.3 The custom measurement device installed on hex-rotor UAV

In the following online address (goo.gl/dhUPWn), a video from the testing of the basic approach can be viewed.

6.2 Advanced approach

Having gained experience from the test in the basic approach, the next step is to integrate knowledge and techniques from the previous parts and build an

autonomous vehicle that will be able to perform behavior algorithms like the obstacle avoidance application. An overview of the system, which was developed for the advanced approach, is illustrated in figure (fig.6.4). The primary difference compared to the previous system is that in this case there is an onboard computer that makes the decisions for the avoidance manoeuvres instead of an operator. Hence, the onboard computer receives input data from the appropriate sensors and executes the obstacle avoidance algorithm, which was analyzed in the simulation part, in order to execute the required application. The system was built using ROS providing us with the required framework to develop the code that is capable to engage with all the individual system's components.

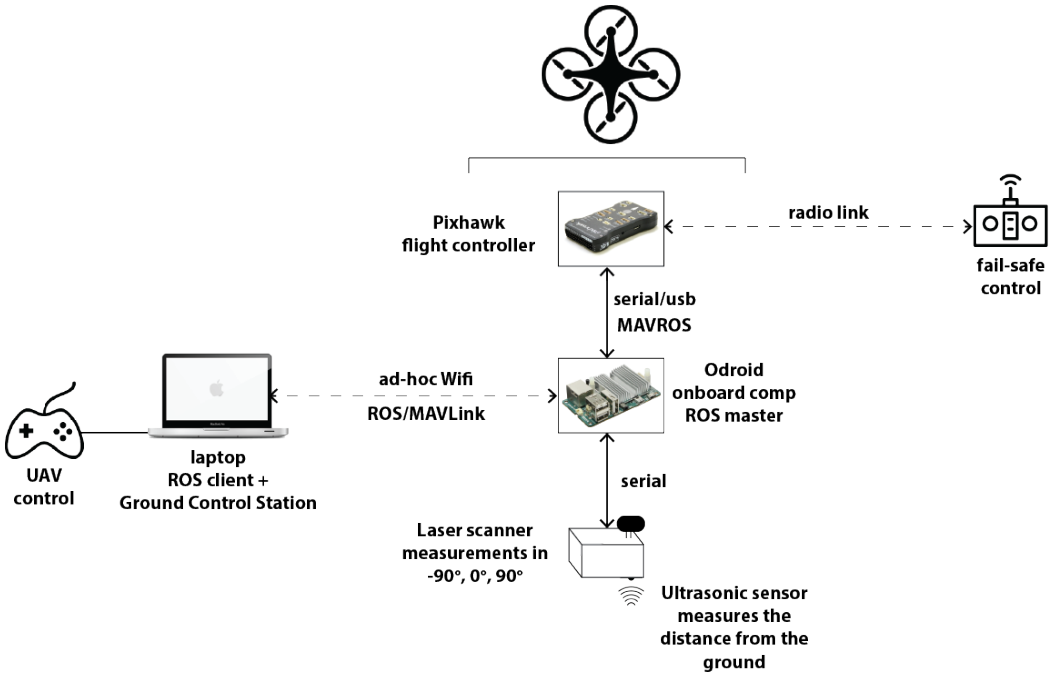


Fig.6.4 Overview of the system in advanced approach

The basic principle that was applied in order to design this approach is essentially the architecture of a simple robotic system (fig.6.5). In this system the sensor performs measurements of the environment. The sensor is driven by a controller. This controller is connected to a computer which receives the data from the different sensors and feeds them as inputs to the decision algorithm. The outputs of the executable script are the control commands. These commands are sent to the controller that drives the actuator.

Fig.6.5 The architecture of a simple robotic system

A detailed description of the system can be accomplished by breaking it down into its individual entities. In this approach, the custom measurement device has been enhanced (fig.6.6). More precisely, not only it measures the distances in the three predefined directions using the laser sensor but it also measures the distance from the ground using an ultrasonic sensor. This sensor is installed under the plastic box. Both sensors are connected with an Arduino Uno microcontroller having a breadboard shield on top. The laser sensor is connected using the I2C protocol and the ultrasonic using the serial. The microcontroller is also responsible for rotating the servomotor in -90^o, 0^o and 90^o direction. The Arduino sketch, which is executed in the microcontroller, utilises the ROS library. More precisely, it transmits two distances in every position, the laser measurement in this position and the distance from the ground. The custom measurement device is connected with the Odroid on-board computer using a USB cable.

The Odroid on-board computer is the brain of the system. It receives inputs from the custom measurements device and control commands from the ground computer. The communication between the latter is implemented using a peer-to-peer wireless network (ad-hoc wireless). Therefore at this point, the on-board computer has at its disposal all the sensors' data and operator's control commands required for the execution of the obstacle avoidance application -as described in the simulation part. The on-board computer is based on the Ubuntu server operating system and it executes ROS, as master in the connected network. In order to enable the communication with the flight controller, the MAVROS ROS package is installed.

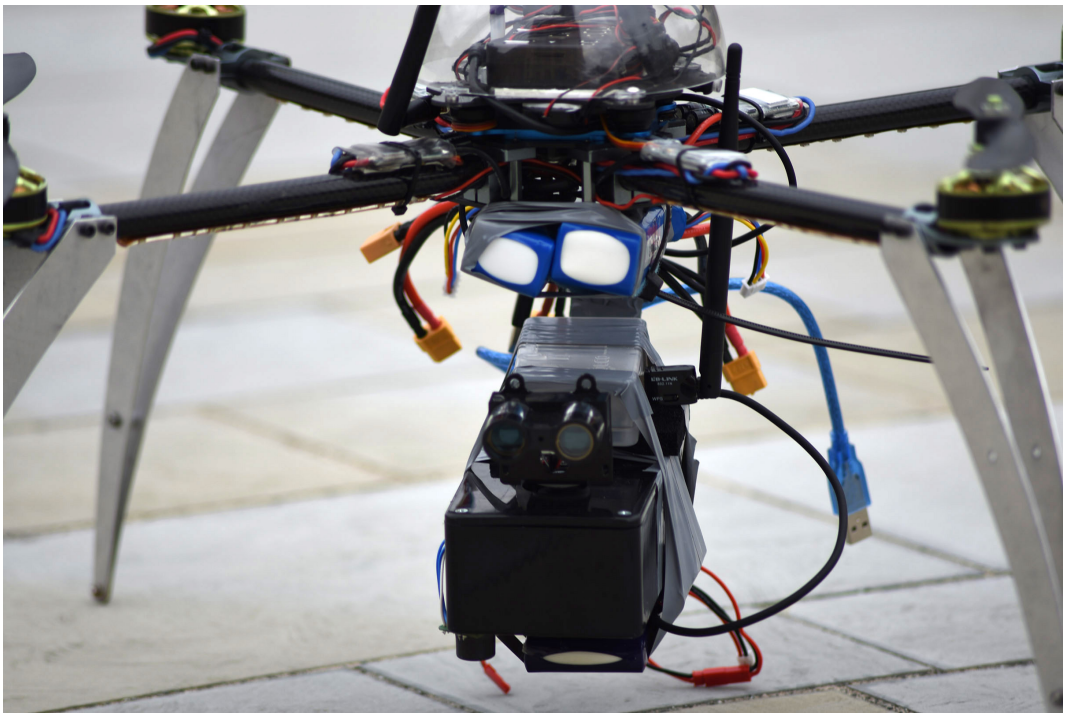


Fig.6.6 The custom measurements device. We can see the Odroid with the wireless antenna on top, the LIDAR-Lite in front and the ultrasonic sensor under the box

The Pixhawk flight controller is a ROS enabled device (fig.6.7). It is connected via USB with the on-board computer. The obstacle avoidance application issues the appropriate control commands that are formed following the MAVROS – MAVLink communication protocol and are sent to the flight controller. The implementation of the obstacle avoidance application for the real UAV is explained in the related appendix. Pixhawk is also connected with a telemetry serial board to enable the communication with the qGroundControl station, if needed. Moreover, one important component that is connected with the controller is the radio control receiver. This receiver handles the communication with the radio control. This is very important because using this link the failsafe option was implemented. More precisely, a radio channel is dedicated to the switch that controls the transition from the manual to the autonomous flight mode. Whenever the operator wants to interrupt the automated avoidance manoeuvre and take back the control, he changes the switch in the radio controller.

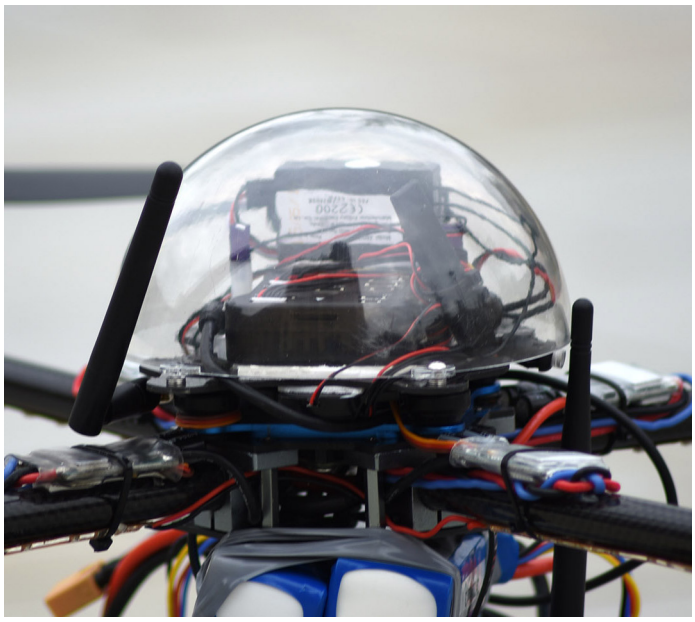


Fig.6.7 Inside the dome we can see the Pixhawk flight controller, the rc receiver and the telemetry board with the external antenna

As I have already mentioned, a ground computer is part of the system. The ground computer executes ROS as slave, having as master node the one that is executed in the on-board computer. This means that the joystick, which is used by the operator to send the control commands, even though is physically connected to the ground station, acts as being directly connected to the UAV. As with the simulator, the function of the joystick is slightly modified so as to issue commands towards the obstacle avoidance application and not to the UAV. This is necessary because the application acts as a getaway and reactively alters the control commands according to the situation.

Unlike the basic approach in which the custom measurements device was mounted on a ready-made hex-copter, in this case a UAV was designed and built from scratch (fig.6.8). At this stage valuable help was provided by Imitec to design and build the project's UAV. The frame was taken from an old UAV which was almost broken. The basic parts such as the frame arms and the brushless motors were kept unchanged and the entire UAV was built from the beginning (fig.6.9). It is a X4 frame equipped with all the aforementioned components such as Pixhawk, rc-receiver, telemetry board, Odroid, custom measurement device. In addition, a custom aluminium landing gear (fig.6.10) was designed and implemented in the University of Bristol Physics' Workshop. Special care has been taken in order to use a light material and join the sheets using elastic

connectors to improve that anti-vibration behaviour. Ideally, in a further improved version of the UAV, the material of the landing gear would be carbon fiber to achieve lower weight, higher durability and fewer vibrations.

Fig.6.8 The project's X4 quad-copter with the custom measurements device mounted as payload

In the following online address (goo.gl/dhUPWn), a video from the testing of the basic approach can be viewed.

Fig.6.9 The progress of building the new X4 quad-copter for the project

Fig.6.10 Quad-copter X4 with custom landing gear

Part 7
Evaluation

In the main part of the dissertation, the methodology regarding the project's objectives has been presented. In addition, the related technical details are included as appendices at the end of this dissertation. In this part, an evaluation of the results will be presented. Positive points will be identified as well as the limitations of the employed approaches. Moreover, ways to improve the implemented techniques will also be proposed. The evaluation will be performed separately for every individual part.

7.1 Sensors

In the sensors' part I surveyed the behaviour of a range of proximity sensors that can potentially be engaged in a UAV project. At the beginning, the basic principles of the operation for every type of sensor is explored. Following this, for the needs of the current project, I chose a representative sensor of every type to carry out the experiments.

Firstly, in order to access the performance of the sensors, I studied the accompanying specifications and the data sheets. Secondly, I used a common microcontroller such as the Arduino Uno and I carried out experiments so as to explore the behaviour of the sensors under various circumstances.

7.1.1 Ultrasonic sensor

The ultrasonic sensor is a very common type of sensor in the robotics application. It is a low-cost, small size and lightweight sensor. Additionally, it has low power consumption and a medium reading rate. For my tests I utilized Devantech's SRF02 ultrasonic sensor, a common sensor that does not belong to the high-ends of this category. It has a medium resolution and accuracy. Nevertheless, it is accepted in obstacle avoidance algorithms where the centimeter accuracy is not required. The important disadvantages of this sensor are the following:

- When the sensor is not facing the object's surface perpendicularly the distance measurements are erroneous. The specific sensor reports satisfactory measurements up to an angle of 70^o (20^o difference from the perpendicular direction).
- The sensor's measurements are influenced by the shape of the acoustic lobe. Consequently, in cases where the sensor is spatially restricted the measurements are incorrect. For instance, when the sensor measures the distance from the end of a corridor, the reported distance is equal to the distance between the sensor and the intersection of the lobe with the sidewalls, rather than the expected one.
- In UAV scenarios it is important to test the behavior of the sensor related to the wind. By performing experiments using airflow, it was observed that when the airflow was directed towards the sensor, some measurements were incorrect.

Although the sensor has drawbacks, it is usable in UAV applications. Most of the times it is utilized in order to measure the distance from the ground. Using this, it allows a smooth touch to the ground and stable altitude-hold flights. The measurement from the sonar in low heights above the ground is most of the times more realistic compared to the GPS altitude and the barometer's atmospheric pressure altitude.

Further experiments were performed, included the mounting of the sensor in a UAV and the investigation of its performance on board. In this case the sensor measured incorrect values the first 10-20cm during takeoff. This was happened because the blades rotate too fast and the airflow hits the ground and returns towards the sensor. An additional source of problems that is known but was not identified is the utilization of more than one ultrasonic sensors simultaneously. The problem is that each sensor interferes (cross-talk) with the others because the returning wavefronts are received from all of them at the same time. A way to compact the cross-talk is the method of chaining in which a sequential transmission of the wavefronts takes place in a loop. Unfortunately, this behavior was not tested since only one sensor of this type was acquired for the project. The disadvantages of cross-talk and sensitivity to wind noise are the main reasons for the ultrasonic sensors not being an ideal solution for obstacle avoidance UAV applications.

7.1.2 Infrared sensor

The infrared sensor is also a low-cost, small size and lightweight sensor. It has a fast reading rate and a moderate power consumption. Sharp's GP2Y0A02YK0F is one of the common infrared sensors which could be utilized in various applications such as touch-less switches, sensors for energy saving and amusement equipment. The experiments of this sensor showed that it is not suitable for the UAV scenario for the following reasons:

- It has a very restricted range of 0.2 to 1.5m.
- For distances larger than 1m, it has poor resolution.
- It is affected by the angle between the sensor and the object's surface as well as the reflectivity of the object's material.
- It supports only analog output type. Consequently, output is depended on the stability of the input voltage. To get better performance a by-pass capacitor should be installed to stabilize the power supply.

The maximum range of this sensor is the primary reason why it cannot be used in a UAV. The distance is very small to allow for a proper reaction even for a UAV that flies slowly.

7.1.3 Optical flow sensor

The px4flow optical flow sensor belongs to the high-end category of UAV sensors. It measures the distance using an on-board ultrasonic sensor and estimates the optical flow using a CMOS machine vision sensor. The recommended use of this sensor in a UAV scenario is to mount it below the central hub, towards the ground. In this case, the sensor will measure the distance from the ground, useful in landing and altitude hold flight mode and also will estimate the ground speed and therefore the position. The latter is particularly useful in GPS denied environments allowing long-term dead reckoning navigation.

The testing of the accuracy of the optical flow estimation can be accomplished by using either of the two following approaches. Firstly, the most accurate method to verify the reliability of the reported data is to use a VICON motion capture system. This method is best suited for indoor applications where the space is

restricted and there is need for more accurate movements. Secondly, the data from a GPS unit can be used, which -though less accurate than the first method- they can be used as ground truth data in outdoor applications. For the purposes of this project, I did not carry out any experiments regarding the optical flow estimations, primarily because there is no plan to engage these data in the obstacle avoidance algorithm. Another specific reason why this was not engaged is because in highly radiation-contaminated environments, cameras have limited and problematic functionality.

The installed ultrasonic sensor is a Matbotix range finder. This sensor showed considerably improved behavior compared to the Devantech's SRF02. It supports one-millimeter resolution but the most impressive is the narrow acoustic lobe. By performing the same experiment in a corridor, this sensor was able to measure the distance from the end having a threshold three times greater than the other one. Consequently, this sensor could be successfully employed in UAV applications owing to its enchanted accuracy and directiveness.

7.1.4 Laser sensor

The LIDAR-Lite ranger finder was the laser sensor which was used in order to carry out the related experiments. This sensor has a maximum range up to 40m, 1cm resolution and high accuracy. It also has a very high reading rate by taking 100 measurements per second. By performing various experiments, its accuracy was verified and the ability to measure distances even when the sensor was not oriented perpendicularly to the objects' surface. An interesting behaviour was also identified related with the surface's material. More precisely, for the cases where the sensor's orientation was not perpendicular, for instance 45^{o}, if the material was glass the measured distances were incorrect. In cases where the material was concrete, such as a wall, the measurements were correct.

LIDAR-Lite has already been a popular sensor used in a wide range of robotics applications. In UAV scenarios, it can be used in two cases. Firstly, It can substitute the ultrasonic sensor that measures the distance from the ground. Using this approach, the UAV will be able to hover with enhanced stability in greater altitudes compared to the maximum range of 4-5m, which a typical ultrasonic sensor can support. In addition, a laser sensor is not affected by wind

noise, which could be an advantage in outdoor flight scenarios. Secondly, the laser sensor could be used in obstacle avoidance applications either by employing an individual sensor for every direction or more effectively by having a sensor mounted on a servomotor. The latter enables us to sweep a range of 180^{o} degrees and essentially build a laser scanner.

All in all, during the experiments, this sensor showed the greatest reliability and accuracy. It supports I2C and PWM connectivity that facilitates its utilization in combination with the Arduino platform. All these advantages made this sensor the most versatile piece of hardware among the others in this sensor survey and that is the reason why it was selected as the basis for the sensor applications.

7.1.5 Sensors applications

The exploration of the strengths and weaknesses of the proximity sensors was not adequate without trying to engage them in useful robotics applications. The most versatile of those is the LIDAR-Lite sensor. This sensor was mounted on a servomotor and connected with an Arduino microcontroller. The latter provided the required connectivity to the laser as well as a way to control the servomotor.

In the first application, the laser sensor performs 180^{o} sweep and every to two degrees the current angle and the measured distance are transmitted to the serial port. On the other side, Matlab converts the data to Cartesian coordinates and applies the RANSAC algorithm to identify existing lines. By implementing this approach my primary intension was to explore the behaviour of this structure, laser sensor and Arduino, handle the measurement data and process them properly in order to get a meaningful output. On the processing part, more sophisticated procedures would have been applied. Nevertheless, this application can stand as a draft approach to mapping and as a way to get familiarised with the hardware and the data. Obviously, this application cannot be employed in a real UAV.

The second application was implemented in order to measure the distance in individual predefined directions. This application was implemented using the ROS framework and can be utilised by any ROS-enabled robotic structure. For this project the predefined directions are at -90^{o}, 0^{o} and 90^{o}. In terms of accuracy

the measurements are less precise compared to the static tests that I conducted with the laser sensor. This was expected since the sensor is mounted in a rotating platform that cannot be synchronised accurately. Nevertheless, this does not prevent the utilisation of this structure in an obstacle avoidance application in which the centimetre accuracy can be compromised. One of the main disadvantages of this structure is the rotation speed during the measurements. In cases where the speed was high, lost of synchronisation and drifts in the servomotor had occurred. This posed a limit in the reading rate of the measurements in each direction. However, this structure could be usable in an application like the radiation mapping UAV since one of the requirements is to fly slowly.

A further step for improving the proximity application was to reduce the required volume of the structure. In this direction, and having identified that the functional properties are the same (processor and memory), I substituted the Arduino Uno microcontroller with the considerably smaller Arduino Nano. The performance was the same, as it was expected. An additional positive side effect was the improved stability of the system since I used a separate power supply and I did not feed the servomotor from the microcontroller's 5V output. Consequently, having achieved a smaller size structure, it is easier to design a 3D printed housing and end up with a compact device.

This modification of this system was actually used in the basic approach of the real UAV experiments and the results were satisfactory. Further optimisation in the system could be the use of a more specialised microcontroller. It is acceptable that Arduino is a great tool for rapid prototyping but in order to enhance the performance, more specialised hardware is necessary. In addition to that, a lower level language will be helpful in order to leverage the full reading rate of the laser sensor and achieve precise synchronisation. The existing Arduino IDE, can offer the simplicity of quick setup applications using convenient commands but for an enhanced performance it is required to code in lower level. Finally, another improvement that can be attempted, although it would add some volume, is the utilisation of a stepper motor, instead of the servo, for 360^{o} sweep with a slip ring for the cables.

7.2 Simulation

For this part the evaluation will be made for the simulation approach and the obstacle avoidance algorithm. For the simulation part of the project I needed a method that would satisfy the following requirements:

- Realistic simulation of the UAV dynamics
- Ability to customise the configuration of the UAV by adding or removing components
- Availability of a wide range of proximity sensor and ability to modify their operational properties
- Ability to build custom landscapes

All these requirements lead to an environment in which an effective application simulation such as the obstacle avoidance algorithm could be implemented. The option that satisfies all the aforementioned conditions, giving even more potential capabilities, is the combination of ROS and Gazebo using the Hector quad-rotor UAV simulation. It was very hard to build an initial working environment but after a period of learning and experimenting, it became friendlier. Another advantage of selecting ROS as the underlying infrastructure is that it can be easily applied to ROS-enabled robots without any -or with minor- modifications. For this reason Pixhawk has been selected as the UAV's flight controller since it supports ROS using a specific communication protocol called MAVROS.

In the related sections there are detailed guides explaining how a user could configure a working environment and start directly to experiment with navigation algorithms. All in all, this approach offers a realistic and reliable way to simulate UAV behaviour, giving the freedom to customise the UAV configuration according to project's requirements.

The obstacle avoidance application was designed as an assisted tele-operation flight mode. In this case there is the UAV operator which controls a UAV that performs radiation mapping. By saying this, there is the assumption that the UAV flies slow, low and with precision. Having these conditions applied, the application takes the control, when an obstacle is observed, and performs the appropriate avoidance manoeuvre. The actions performed in these manoeuvres are designed to facilitate the radiation-mapping task by implementing the following behaviour:

- As the application has been designed, the UAV flies in a constant altitude and when the avoidance manoeuvre is completed it returns to this altitude.
- Whenever a manoeuver is performed, before the application returns the control back to the operator, it moves the UAV further from the end of the obstacle and returns in the initial heading that the UAV had before this procedure.

The obstacle avoidance application was tested using various landscapes in the simulation environment. Not only basic shapes but also more complex scenarios were tested, where there is another obstacle during the obstacle avoidance procedure. In cases where for some reason there is a failure, either because of a measurement error or human mistake in operation or an undiscovered bug in the algorithm, there is the fail-safe option of the emergency landing. This option is activated when an object is sensed closer than 1 meter.

The simulation of the application has been designed using realistic sensors' configurations and properties. In addition, all the functional settings such as the distance thresholds are fully parameterized in the related files. The simulation has been built using the object-oriented approach in which an individual entity has been developed for every action. For instance, there is a separate action for changing the heading, or for changing the altitude. These actions are utilized by the obstacle avoidance algorithm but at the same time they can be used in other scenarios as external library tools.

The implementation of this algorithm relies on predefined reactive manoeuvers that performed as commands of an embedded finite state machine. In order to achieve a higher performance level, more advanced approaches should be implemented. These approaches include navigation and mapping. The UAV starts to act more autonomously and the control shifts from the operator to the UAV side. In ROS, there are available libraries that can support these tasks, but in order to perform effective mapping there is a need for more advanced proximity sensors such as a laser scanner or a depth camera, which are not available in a radiation mapping UAV.

7.3 Human robot interaction

One of the factors that can improve the experience of interacting with a robotic system is the friendliness of the interface. More precisely, the desired interface should be adapted in order to facilitate the specific usage, in our case the radiation mapping. Moreover, the implementation of an easy to use interface, designed by engaging visual controls and rich media such as streaming video, improves the operator's experience of use, which is the case for the application about the UAV control.

The aim of the first application is to implement an immersive interface in order to control a ROS-enabled UAV. The application is designed in such way that it can facilitate the control during the radiation mapping procedure. Hence, the operator can set the constant altitude that is desirable for the measurement procedure and then by controlling the linear velocity in x, y as well as the heading, he can manipulate the UAV in a plane parallel to the ground. Moreover, the scaling of the velocities is adjusted to low levels in order to prevent the fast transitions that could lower the quality of the measurements.

A useful feature that can further improve the operator's use is to add waypoints. In cases where routine radiation checks are performed in known places, this could save time and effort instead of having an operator to control the UAV manually. The application has already a communication channel to obtain GPS measurements. These values are visible in the control screen and the map. Thus, the operator, by setting annotations (waypoints) to the map, can design a trajectory to enable the UAVs autonomous flight. Then the application could issue the appropriate control commands to accomplish the desired route. Unfortunately, this feature was not implemented due to time restrictions, although the underlying infrastructure exists and it is compatible to expand. Another addition is to improve the communication between the smart device and the UAV. The existing wireless setup is based on the infrastructure mode that requires a central access point that all devices connect to. An improvement, especially for outdoor flights, is to set-up a peer-to peer wireless network, also known as ad-hoc, in which the smart device connects directly to the on-board computer.

To conclude, working in this application was very beneficial since I explored available ways for connecting a ROS-enabled UAVs to external systems. The interconnections are generic since they use web-services as a base, which is an advantage because this experience can be utilized in other platforms as well – that is platforms different to iOS (web-clients, android). Experiments using this application have been conducted using ROS simulation. Nonetheless, since ROS uses the same principals and structure both in simulation and in the real hardware, it is expected that the application will be usable in a real UAV as well.

The radiation mapping procedure, as it is performed when using the existing equipment, is an offline-procedure. More precisely, using the radiation mapping base unit, which has been developed in IAC, the operator performs the radiation measurements and after this the data are uploaded in a client application on a computer. Up to this point, the operator cannot check the correctness of the data. Various failures can happen such as problems in the GPS lock so, after long mapping surveys and only at the very end, the operator can find out that the registered measurements are useless.

The radiation mapping application was designed to give real time connection and feedback from all the functional components in the radiation mapping base unit. Thus, the operator can have direct information from the Bluetooth link, the GPS connection and the spectrometer device. In addition to that, the radiation measurements are visualized on a map using color-encoded dots. This can improve the quality of the mapping because as the operator has a live view of the radiation levels, he can focus the survey to this region instead of blindly measuring around the place. In addition, the application offers a set of parameters that can facilitate and tune the mapping. The operator can define dynamically the scaling of the measurements range and get a better resolution in low contaminated areas as well as define the radiation mapping method. According to the method the zoom level is changed and additionally an attenuation factor. This factor is related to the attenuation imposed by the vehicle type, so it is different when measurements taken inside of a car or on a UAV. This last setting was not implemented since accurate tests have to be performed in order to define these factors.

IAC researchers in real radiation mapping surveys in Cornwall tested the mobile application. The results were very satisfactory and they mentioned that could be

an important mapping tool if certain features are added. The application has been developed as a prototype and further improvements need to be added. For instance, the application operates only in live mode with means that the operator cannot save or load measurements files. This is a very important feature to add. In addition, a change in the communication channel could improve the performance. The application uses a Bluetooth link to exchange data with the base unit. However, inside the base unit there is a wireless shield that can be utilized in a peer-to-peer communication. This will increase the range, since Bluetooth is low energy, and at the same time it will reduce the complexity of the base unit.

Ideally, both applications should be merged to one. Therefore the operator, using a smart device such as a tablet, will be able to control the UAV and have a real time assessment of the contamination levels by seeing the measurements on the map. This is a viable approach since the radiation-mapping UAV can be equipped with a radiation mapping base unit and an onboard computer which are required in order to enable the aforementioned applications.

7.4 Real UAV

The obstacle avoidance behavior was applied in a real UAV using two different approaches. The first one could be characterized as the basic one where the human still plays the role of the decision center. More precisely, in this approach the operator gets real time information about the surrounding obstacles, and changes the control of the UAV accordingly.

The vital parts of this approach are a custom measurements device and an application. The device is based on the proximity application, which was developed in the sensors part. It measures the distances in front and on both sides. The application receives those measurements and visualizes accordingly in each direction whether there is an obstacle or not.

This application has been tested by mounting the custom measurement device on a hex-copter and performed tests at IAC. The performance was as expected and although it was difficult to assess the accuracy of the measured distances,

the obstacle presence was always identified. Besides, the radiation mapping procedure requires low flight velocities that are positive for the performance of such measurements devices. This is because, as I have already mentioned, the measurements device has low measurement rate due to limitations in the servo motor rotation speed and the synchronization.

The concept of this approach is simple but the aim was to get experience from the overall behavior of the UAV and the performing of distance measurements onboard before moving to a more autonomous approach. However, it's a working prototype which can be merged with the control application and can inform the operator interactively about the presence of obstacles in the same screen where the controls of the UAV takes place.

In the advanced approach, the human in the loop is substituted by the on-board computer. More precisely, the on-board computer executes the obstacle avoidance algorithm which was presented in the simulation part. The custom measurements device of the basic approach was improved. An ultrasonic sensor was used to measure the distance from the ground and the Bluetooth shield was removed since, in this version, a ROS-enabled sketch was developed for the direct communication with the on-board computer. On the other side, the on-board computer is connected with a flight controller, which is able to receive ROS messages using the MAVROS interface.

The robotic system, which I have described, was installed in a custom X4 quad-rotor which was built for the real experiments of this project. Custom parts, such as the custom device mounting support and the landing gear, were designed and built in the Physics workshop.

In a complex system like that, extensive testing is required in order to ensure the safety and the correct operation. At the beginning, all the individual parts were configured and tested separately, like the on-board computer, the custom measurements device and the flight controller. The next step was to connect them and test the communication between them. This part was also completed successfully. After that, the flight controller was installed in the UAV. In this part I had to test the behaviour and the stability of the custom made UAV. After the calibration, flights were performed to verify that the behaviour of the UAV was as expected. After that, the custom measurements device was mounted in the UAV

as additional payload and the same tests were performed. In this case, although the extra weight could be lifted by the UAV, minor adjustments were needed to improve the stability. Additionally, interesting observations were made about the need of further filtering of the measurements. Another useful improvement could be the increase of the measurements rate of the custom device using a more advanced implementation.

Having ensured that the UAV was able to fly safely, the autonomous behaviour had to be tested. At the beginning, simple tasks, which form the avoidance manoeuvres, were tested in the UAV without blades. The aim was to identify the proper function and overall connectivity safely without any risk. After that, simple versions of the avoidance application were formed in order to gradually test the UAV's autonomous behaviour. Unfortunately, this stage was not finalised due to time restrictions.

Overall, the operation and the performance of the UAV system were satisfactory. Certainly, further testing is needed to tune the obstacle avoidance application. This attempt could be considered as an underlying structure to be used as a base in order to implement more advanced implementations.

Part 8

Conclusions

One of the research interests at the Interface Analysis Centre (IAC) is the study of low altitude unmanned aerial vehicles for radioactive contamination monitoring. The centre has already developed a UAV, called Advanced Airborne Radiation Monitoring (AARM) system, which is able to perform radiation mapping in industrial and urban areas. It has been used successfully at the Sellafield nuclear sites and the Fukishima Daiichi nuclear power plant. In order to achieve the best resolution and the highest quality measurement data in radiation mapping, the UAV needs to fly low, slow and with precision. Depending on the case, this could be close to the ground or to buildings. Till now, this has not been feasible using a helicopter or current UAV technology.

During the project, the exploration of the development of the next generation radiation mapping UAV was accomplished. It was attempted to engage with a range of actions that could progress the existing approach and not focus in only one direction. Therefore, the following improvements were developed:

- After an extensive survey in a range of different types of proximity sensors, the strengths and weaknesses were identified. Advanced sensors like the laser sensor were utilised further by developing a custom laser scanner, and common sensors like the ultrasonic were used with knowledge of the specific limitations.
- A realistic and reliable UAV simulation was built using ROS and Gazebo. Using this approach, it is feasible to simulate a UAV with custom sensors and with properties that are adjusted to reflect the real ones. Additionally, it is easy to build custom landscapes and test different scenarios. Having set up this environment, an obstacle avoidance algorithm was developed which acts as an assisted flight method. More precisely, this gathers the measurements from the

proximity sensors during the manual flight of the UAV by the operator. When an obstacle is identified, it takes control and performs manoeuvres to avoid it. After that, when in clear space, it returns the control back to the operator. The algorithm was tested by carrying out various experiments in custom landscapes. The tests were successful and the UAV behaviour was the expected one.

- In the part of the human robot interaction two applications were developed. Both of them take advantage of the enhanced visualisation and interaction capabilities of current smart devices. The first application was designed in order to control ROS-enabled UAVs. Using this application, the operator sets a desired altitude and moves the UAV in a parallel to the surface layer. The application also has the capability to display stream video from the on-board camera as well as displaying data and status from the on-board components. The second is a real-time radiation mapping application. It receives data from a radiation mapping base unit such as the GPS coordinates and the radiation intensity via a Bluetooth connection. These values are illustrated on a map using colour-encoded dots according to the contamination. The application offers a way to parameterise the mapping and thus helps the operator to focus on the region of interest. The application was tested in the radioactive hotspots of Cornwall and got very positive feedback from the operators.
- A custom measurements device was developed to identify objects by sending real-time data in a mobile application. In this basic approach the operator flies manually the UAV and gets visual information about the presence of obstacles on a screen. The operator decides how to avoid the obstacle according to the feedback. This approach was tested successfully using a ready-made hex-copter. Additionally, during the project a quad-copter UAV was build from the beginning. It was equipped with an advanced flight controller and an on-board computer. In addition, the custom measurements device was improved by adding an ultrasonic sensor and developing a ROS interface. The on-board computer executes the obstacle avoidance

algorithm which was developed in the simulation part. It receives the sensor measurements and issues the appropriate control commands, if an obstacle is identified closer to the predefined thresholds. This advanced approach was not tested extensively due to time restrictions. However, successful experiments were conducted which verified the new copter's flight control, the performance of the custom measurement machine, the cooperation of the connected devices and the algorithm operation in simple tasks.

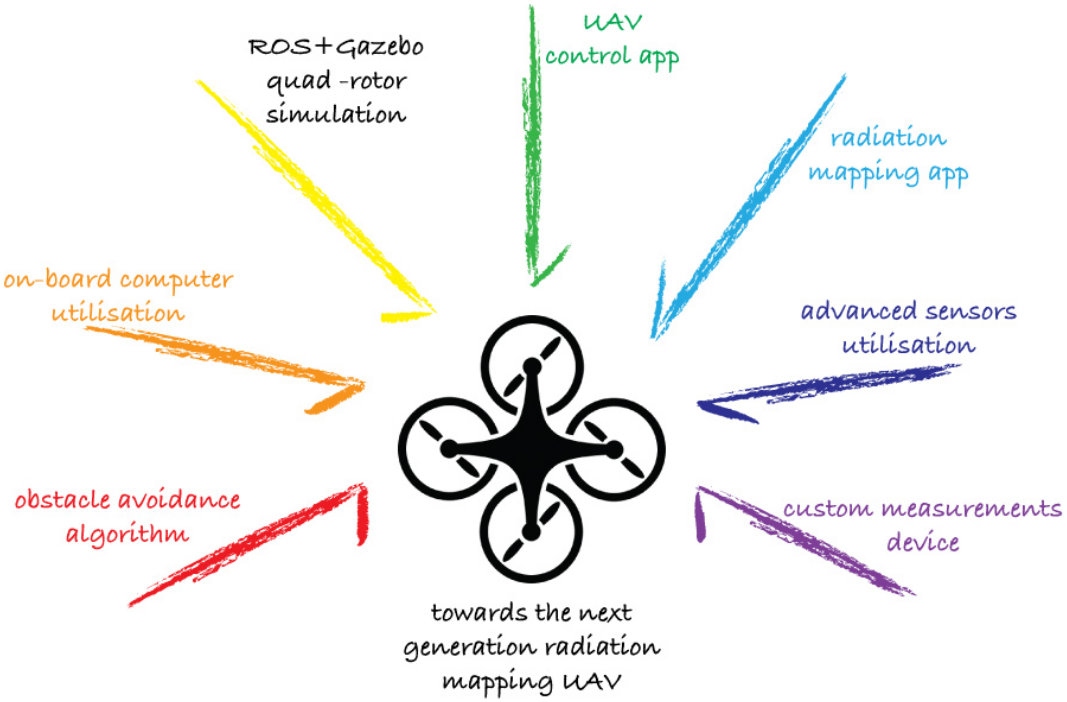


Fig.8.1 All the improvements aim to the next generation radiation mapping UAV

During this project a set of useful tools has been developed. Each of them improves a specific part in the effort to have the UAV flying low, slow and with precision. There is an application that enables the precise control of the UAV in a desired constant altitude. Additionally, the other application illustrates in real time the radiation mapping procedure. A realistic quad-copter simulation was set up to experiment with custom applications. Based on that, an obstacle avoidance application was designed and tested in different landscapes. This is really helpful in complex geometrical shape environments like the nuclear sites are. Different types of proximity sensors

were examined and used to develop a custom measurements device. Using this device and a real quad-copter with an on-board computer was built to be the vehicle of the real experiments. After that, simple tests were designed and performed in order to experiment with the capabilities, the performance and the behaviour of the newly built UAV.

In conclusion, all these different approaches aim at the same target (fig.8.1). The successful combination of all will enable us to have a radiation mapping UAV closer to the one that the initial requirements had set.

Appendix A

Sensors Experiments

This part illustrates the set-ups of the sensor experiments as well as the related measurement tables. Explanations about the emergent behaviour and comments can be found in the "Sensors" part of the dissertation.

A.1 Ultrasonic sensor

For the ultrasonic sensor the following experiments were carried out. At the beginning the sensor was placed towards a wall (fig.A.1). The measurement direction was perpendicular to the surface of the wall.

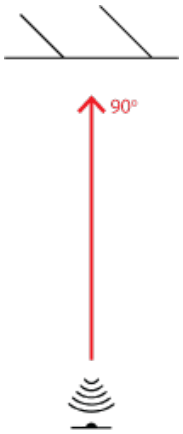


Fig.A.1 The ultrasonic sensor measures distances towards a wall

The following table (tbl.A.1) shows the distances were measured.

Table A.1

(All measurements taken with a surface perpendicular to the sonar)

Distance (cm)	Measured Distance (cm)	Difference (cm)
10	27	17 (dead zone)
30	29	3
80	78	2

130	128	2
170	169	1
210	207	3
310	306	4
400	396	4
500	495	5
600	595	5
> 600	0	NaN

The next set-up (fig.A.2) illustrates the experiment in which the performance of the sensor, which was pointing to the surface in various angles, was tested.

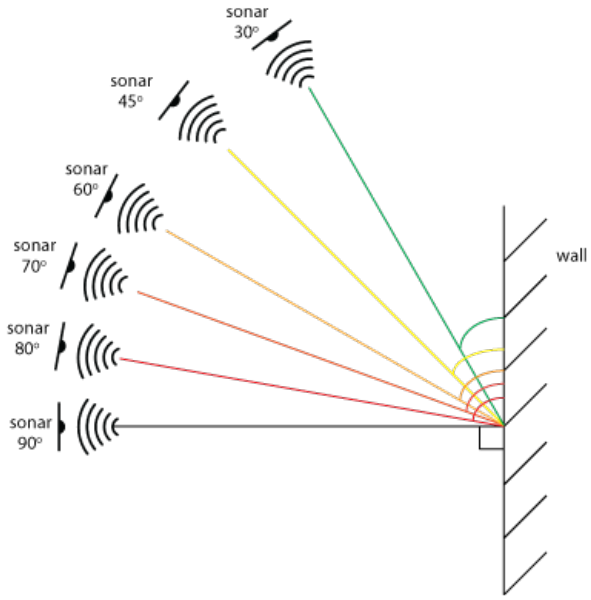


Fig.A.2 Measured distances in different angles to the wall

The results for the distance measurements can be found in the table below (tbl.A.2),

Table A.2

(Measurements taken in different angles towards the wall)

Distance (cm)	Angle (degrees)	Measured distance (cm)	Difference (cm)
100	90	98	2
100	80	97	3

100	70	94	6
100	60	90	10
100	45	145	45
100	30	149	49

The next experiment explores the behaviour of the sensor regarding the shape of the acoustic lobe. The test took place in a corridor (fig.A.3), approximately 1m wide.

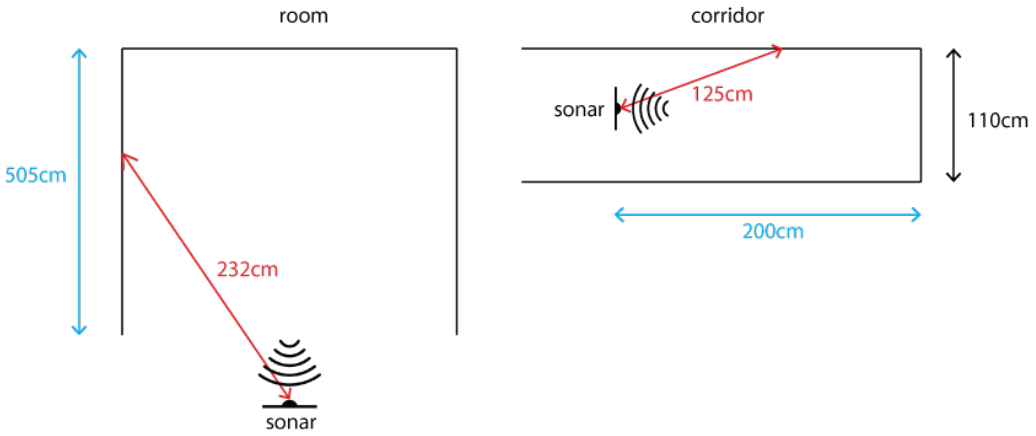


Fig.A.3 Side lobe points lead to incorrect distance measurements even though the space in front is clear and the reflective surface perpendicular aligned.

Finally, the impact of the airflow in the ultrasonic sensor was tested. A pedestal fan was used, as a source of directed optical flow, operating at the highest speed. The distance measurements were performed by placing the fan in different places and different directions related to the sensor (fig.A.4).

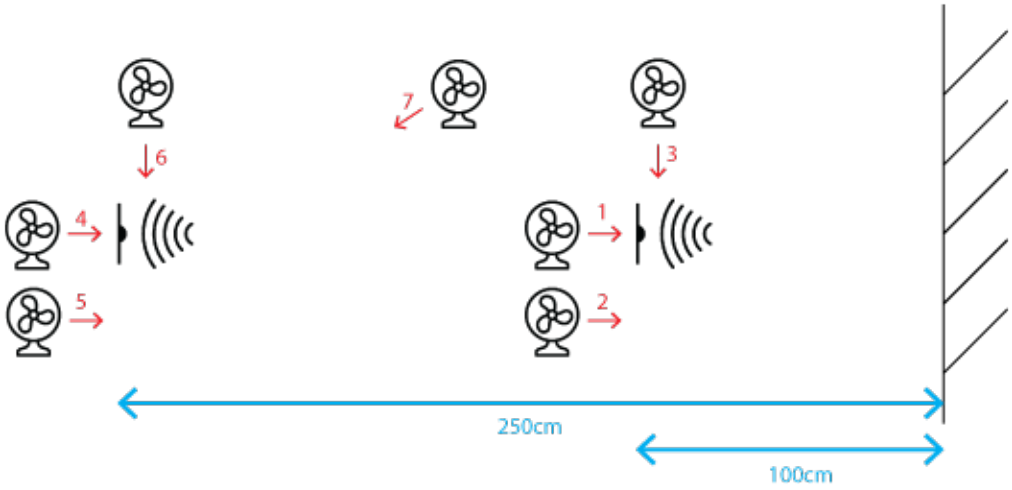


Fig.A.4 Experiments of ultrasonic sensor with the presence of airflow (the fan was operating at the highest speed)

The following table (tbl.A.3) registers the measurements obtained with the fan in various positions.

Table A.3

(Measurements with a source of airflow in different positions and directions)

Distance (cm)	Position	Measured distance (cm)	Difference (cm)
100	1	98	2
100	2	100	0
100	3	103	3
250	4	248	2
250	5	250	0
250	6	247	3
250	7	249*	1*

A.2 Infrared sensor

The infrared sensor was primarily tested in order to verify its accuracy for distance measurements towards a wall in the perpendicular direction (fig.A.5).

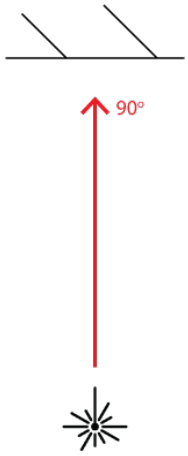


Fig.A.5 The ultrasonic sensor measures distances towards a wall

The distance measurements are displayed in the following matrix (tbl.A.4)

Table A.4

(All measurements taken with a surface perpendicular to the infrared sensor)

Distance (cm)	Distance measured (cm)	Difference (cm)
13	18	(Dead zone)
30	30	0
50	53	3
80	89	9
110	128	18
140	164	24

A.3 Optical flow sensor

For the px4flow smart camera only the reported distance from the ground was tested. In order to test the reliability of the ground speed measurements a

VICON motion caption system is necessary. The first set of experiments related to this sensor was to measure distances facing a wall perpendicularly (fig.A.6).

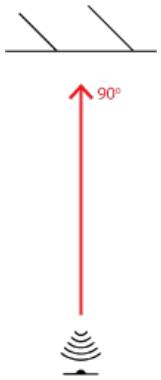


Fig.A.6 The px4flow smart camera measures distances towards a wall

The related distance measurements are shown in the following table (tbl.A.5).

Table A.5

(All measurements taken with a surface perpendicular to the sonar)

Distance (cm)	Distance measured (cm)	Difference (cm)
25	30	(Dead zone)
40	40	0
60	60	0
100	101	1
150	151	1
200	201	1
300	301	1

In order to examine the performance of the Matbotix sensor related to the specific shape of the acoustic lobe, the corridor test was conducted -as shown the following figure (fig.A.7).

Fig.A.7 Matbotix EZ4 tested in a corridor experimenting with the beam width.

A.4 Laser sensor

The first experiment regarding the laser sensor was to assess its accuracy by measuring the distances against a wall in the perpendicular direction (fig.A.8).

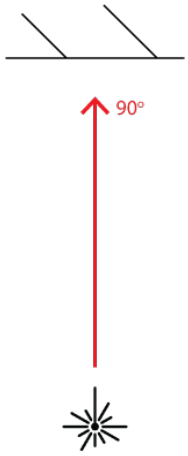


Fig.A.8 The laser sensor measures distances towards a wall

The measured distances are shown in the following table.

Table A.6

(Reflective objects are perpendicular aligned)

Distance (cm)	Distance measured (cm)	Difference (cm)
30	34	4 (close to dead zone)
60	60	0
100	100	0
250	250	0
400	402	2
500	501	1
1050	1040	10
> 4000	0	NaN

The next experiment explores the behaviour of the laser sensor as related to the material of the surface that it hit from different directions (fig.A.9).

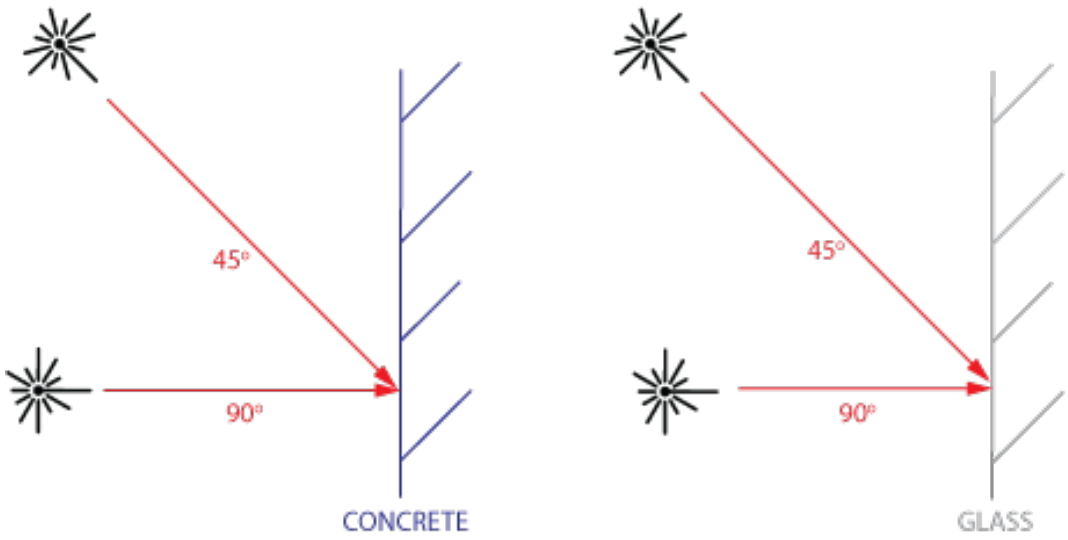


Fig.A.9 The laser sensor measures distances in different directions and towards different materials.

The measurements related to this experiment are shown in the following table (tbl.A.7).

Table A.7

(Measurements in different angles and materials)

Distance (cm)	Angle (degrees)	Material	Distance measured (cm)	Difference (cm)
100	90	Concrete	100	0
100	45	Concrete	101	1
100	90	Glass	101	1
100	45	Glass	135	35

A.5 Proximity Application

In order to investigate the behaviour of the proximity application, the custom laser scanner was placed at the end of a corridor (fig.A.10).

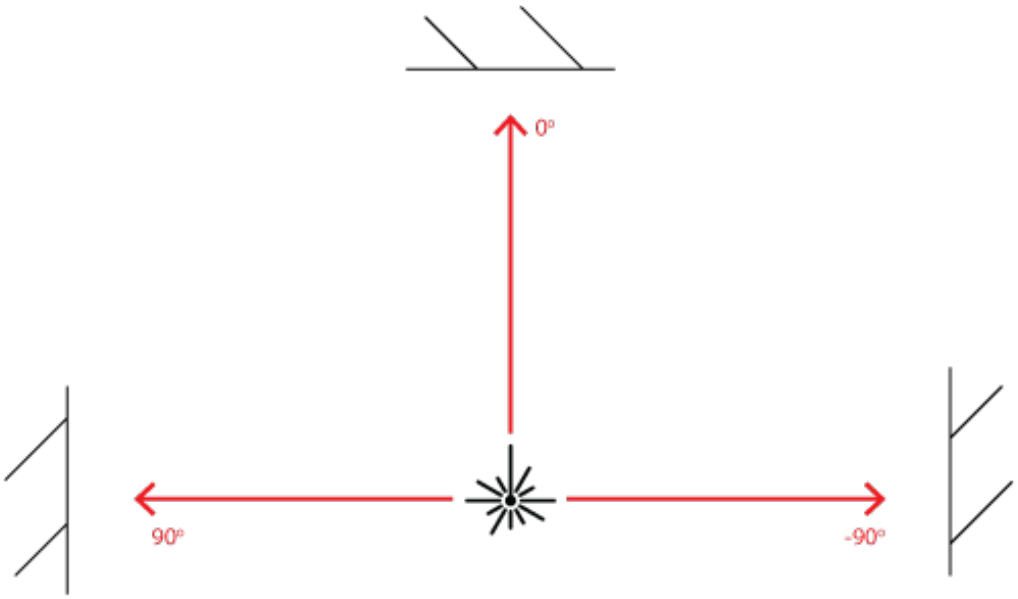


Fig.A.10 The set up of the experiment for testing the proximity application

The measurements in each direction were published continuously in the appropriate ROS topic. The table below (tbl.A.8), contains measured distances displayed in 0^{o}, 90^{o} and -90^{o} angle directions.

Table A.8

(Measurements in the three individual directions)

	Distance Measured (cm)	**Distance (cm)**
dist0	52, 57, 55, 54, …	55
dist90	77, 79, 78, 81, …	79
distm90	84, 86, 87, 88, …	86

Appendix B

Sensors Circuits

In this chapter I present all the circuits I utilized to experiment with the sensors. It includes the schematics representations, the tables with the pins connections as well as comments about specific aspects and behaviours I experienced.

B.1 Ultrasonic sensor

The Devantech SRF02 features both I2C and Serial Interfaces. For the experiments, the later was used. The serial interface is a standard TTL level UART format at 9600 baud and can be connected directly to the serial ports on any microcontroller. In my case the microcontroller was an Arduino Uno and the exact circuitry is shown in figure (fig.B.1).

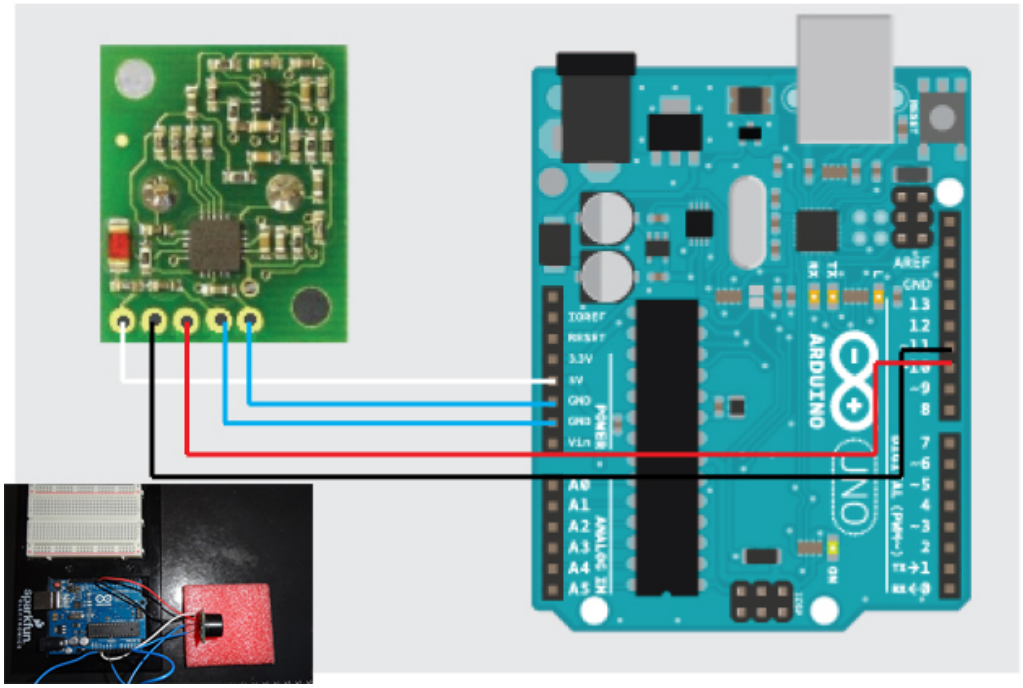


Fig.B.1 SRF02 interfaced with an Arduino Uno

The circuit connections are described in the following table (tbl.B.1):

Table B.1

(Connection between SRF02 and Arduino Uno)

SRF02	Arduino Uno
5V V_{cc}	5V
Rx	11 digital (Tx)
Tx	10 digital (Rx)
Mode	GND
Ground	GND

To select Serial Connection the mode pin should be connected to ground. The Arduino sketch that was used for the measurements is included in the appendices and the online repository (goo.gl/buq1hn).

B.2 Infrared sensor

The Sharp GP2Y0A02YK0F infrared sensor has an analog output that varies from 2.8V at 15cm to 0.4V at 150cm with a supply voltage between 4.5V and 5.5V. In order to experiment with the IR sensor, the most convenient way was to use the analog port of an Arduino Uno microcontroller instead of using multi-meter and calculator. The interface between the sensor and the microcontroller is shown in figure (fig.B.2).

The explanation of the circuit connections is depicted in the table (tbl.B.2). The required Matlab file for the conversion and the Arduino sketch is available in the online repository (goo.gl/oHyf0l).

Table B.2

(Connection between GP2Y0A02YK0F and Arduino Uno)

GP2Y0A02YK0F	Arduino Uno
5V V_{cc}	5V
V_o	A0 analog
Ground	GND

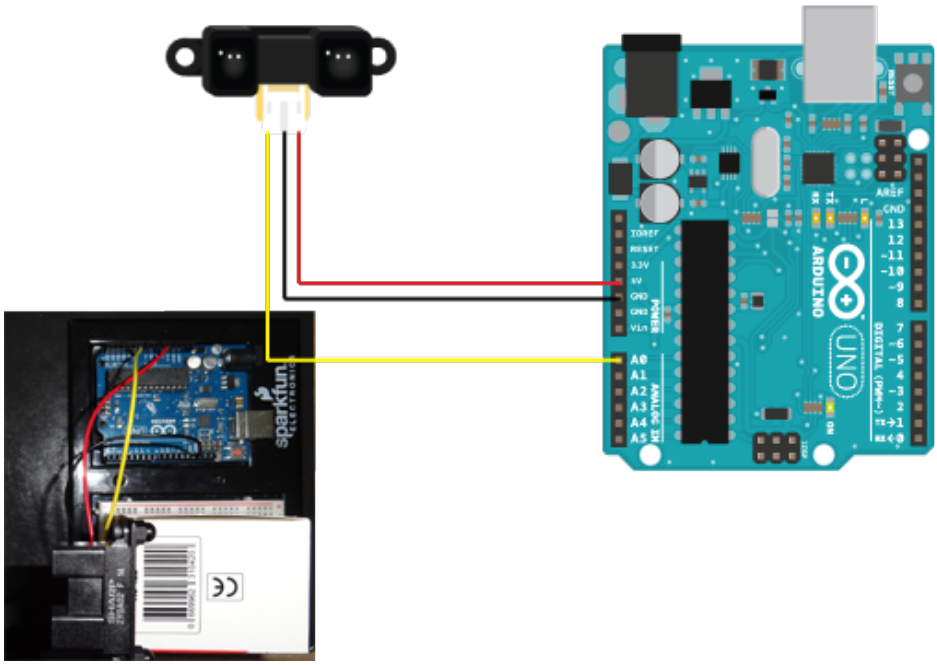


Fig.B.2 GP2Y0A02YK0F interfaced with an Arduino Uno

B.3 Optical sensor

The px4flow smart camera supports various interfaces. The available connections include I2C, micro USB and two USART serial communication ports. For the project's experiments I utilized the first two inputs. For the I2C interface, there was no provided cable on the delivered package. To overcome this lack, I assembled a custom cable using the required Hirose DF13 4 positions plug.

Before starting any use of the px4flow, there are two required procedures that have to be performed. Both tasks were performed using the QGroundControl program (goo.gl/71fQ2M).

- The device should be updated to the latest firmware.
- Using the Video Downlink widget from QGroundControl, the focus should be adjusted by loosening the locking screw and turning the lens at an object at 3m distance.

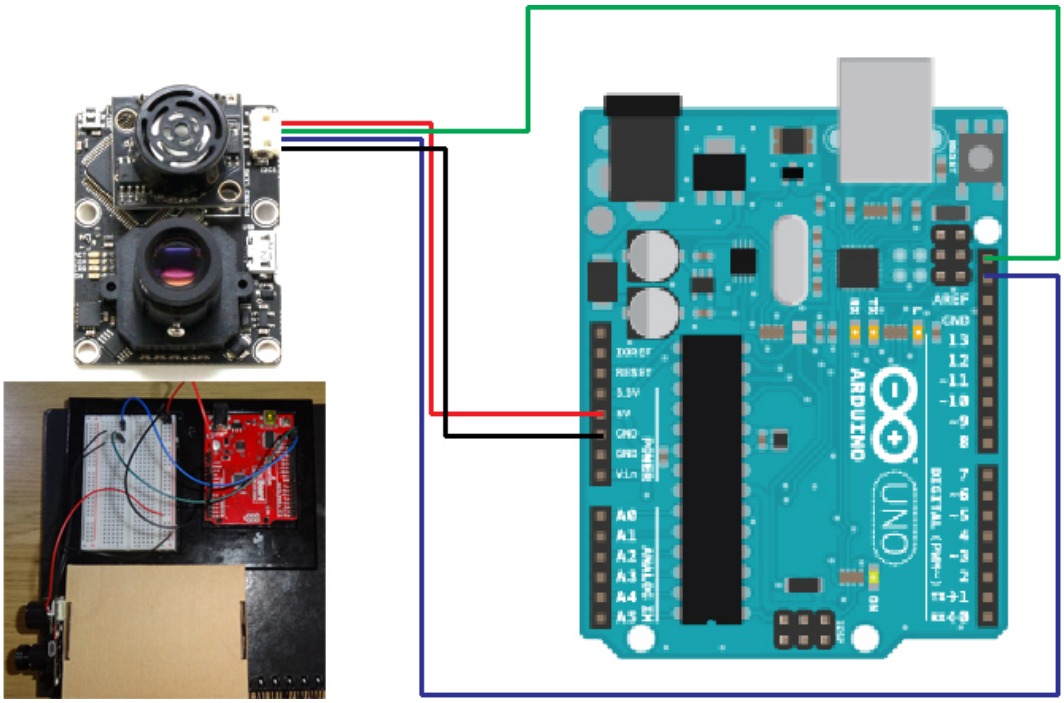


Fig.B.3 Px4flow interfaced with an Arduino Uno

The diagram of the circuit is shown in figure (fig.B.3). Moreover, the following table (tbl.B.3) illustrates the required connections to use the sensor in the I2C mode.

Table B.3

(Connection between px4flow and Arduino Uno)

Px4flow	Arduino Uno
5V V_{cc}	5V
SCL	SCL
SDA	SDA
Ground	GND

The required sketch is included in the appendices and the online repository (goo.gl/MGtnhO).

In addition to previous setup, it is possible to utilize px4flow using ROS drivers. For this case, the device is connected via a USB cable to a computer running ROS (fig.B.4). In order to obtain data from the device, it requires installing the *px-ros-pkg* stack (goo.gl/bvo1cf) in a working catkin workspace. After that, following the guide from the px4flow node documentation (goo.gl/n8IMX9) it is very easy to have access to the video from the camera as well as to the *opt_flow* topic. The *OpticalFlow.msg* type messages that are published in this topic

include information such as the ground distance, the flow in x and y direction, the velocity in x and y direction and the quality of the optical flow estimate. The experiment was performed using ROS Hydro installed on Ubuntu 12.04 LTS. In order to access the device using the proposed launch file, the proper access permissions should be set for the device, for instance (*sudo chmod a+rw /dev/ttyACM0*). In addition, the exact device, for example *ttyACM0*, should be declared in the *px4flow_parameters.yaml* file that stores the global settings of the corresponding launch file.

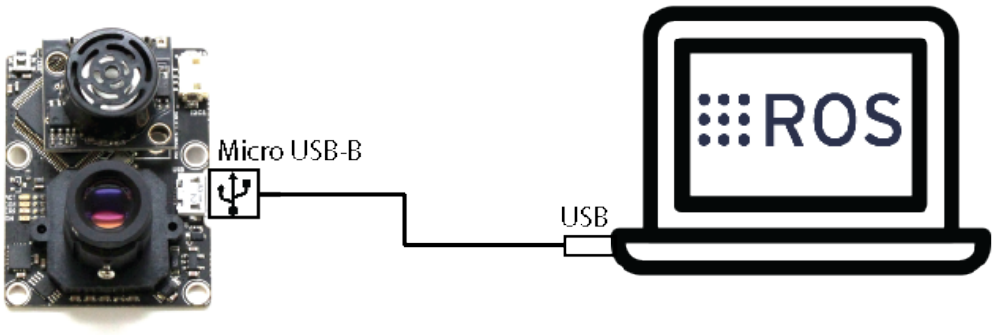


Fig.B.4 Px4flow interfaced with ROS

B.4 Laser sensor

LIDAR-lite laser sensor supports two communication methods, I2C interface and PWM (Pulsed Width Modulation). For the experiments, the former was employed. As with the previous sensors, an Arduino Uno was used to interface with the sensor. More precisely, the sketch utilizes the "Arduino I2C Master Library" from the DSS Circuits (Truchsess, 2015), which is a more sophisticated I2C library than what the default Wire library Arduino IDE offers.

The diagram of the circuit is shown in the figure (fig.B.5). Furthermore, the following table (tbl.B.4) illustrates the required connections in order to use the sensor in the I2C mode.

Table B.4

(Connection between LIDAR-Lite and Arduino Uno)

LIDAR-Lite	Arduino Uno
5V V_{cc}	5V
PWR EN	(Unused)
MODE	(Unused)
SCL	SCL
SDA	SDA
Ground	GND

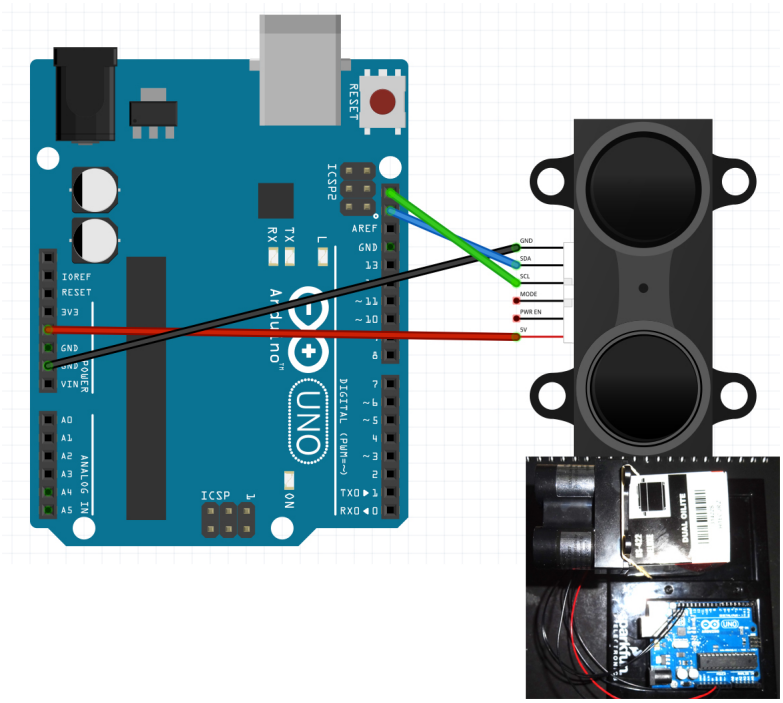


Fig.B.5 LIDAR-Lite interfaced with an Arduino Uno

The required sketch is available in the online repository (goo.gl/DSToIp).

B.5 Sensors applications

For the mapping and proximity application the same setup was used. Particularly, both applications required the combination of the laser sensor with an Arduino Uno microcontroller and a servomotor (fig.B.6). For the interface between the laser sensor and the Arduino Uno, the I2C library, which was developed by the DSS circuit, was utilised. At the same time, I experimented with taking a distance reading using Pulse-Width Modulation (PWM), which does not require I2C communication. Although it is a simpler approach, it lacks of precision. Using

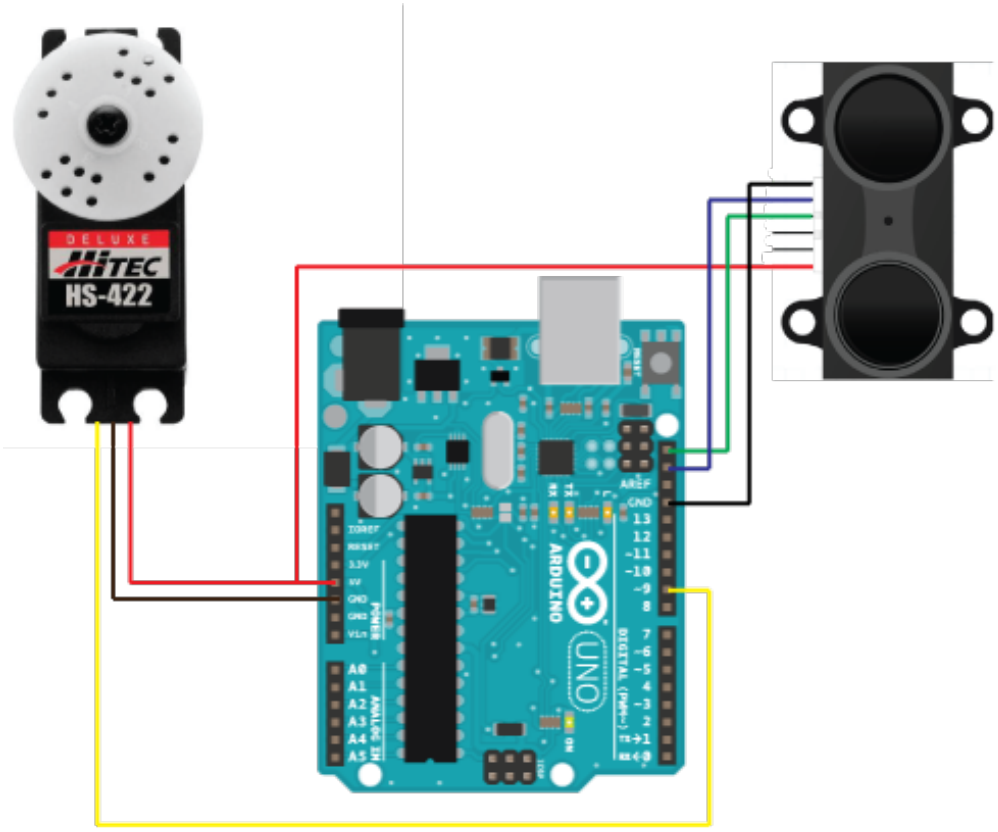
this approach there is no available method to calibrate the sensor. On the other hand, using the I2C approach there is an easy zeroing procedure where the user should only set the value of a specific register (*CalibrationOffsetVlue*) with the required offset.

The motion of the servo was implemented using the Servo library. For the mapping task I used the convenient methods *read* and *write.* The first command reads the current angle of the servo while the second writes a value to the servo, controlling the shaft accordingly. Using the provided servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation. For the proximity application, which was more demanding in terms of accuracy, speed and synchronisation of the movement, the *writeMicroseconds* command was used. It writes a value in microseconds (uS) to the servo, controlling the shaft accordingly. For the provided servo, this will set the angle of the shaft. Further on that, using that method the servo was calibrated exactly to the proper (uS) so as to correspond to the required angles. Thus, the 555uS corresponds to 0 degrees, the 1500uS to 90^{o} and the 2390us to 180^{o} for this specific servo. After the calibration, I investigated the synchronization issues related to the time that it allowed the servo to move before issuing the next move command. I optimized to the shorter interval without having instabilities. As soon as the servo had moved to the desired position and the laser has estimated the average distance, the microcontroller emitted the distance by publishing a message to the specific topic.

Particularly regarding the implementation of the proximity application, on the Arduino side the sketch included specific ROS commands. More precisely, aside from including the servo and the I2C libraries, this time the ROS library was added in complementarily. By doing this, I could declare in the Arduino IDE objects like ROS nodes and publishers. Consequently, I declared three publishers, one for every measured direction. For every publisher, an appropriate message was transmitted with a rate between 0.5 and 1Hz, as measured with convenient time function. This unstable period was one of the reasons why I couldn't compose a laserScan message, because for this type of messages, it is a requirement.

As I have described so far, the ROS topics are composed by the Arduino and are transmitted. However, the last stage -in order to have a working system- is to

employ the *rosserial* protocol. There is a need for a specific protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as a serial port or a network socket. In my case, I needed to execute the serial node python script, having as input the USB that the Arduino was connected to. After that, the proximity topics were visible in the entire ROS ecosystem.



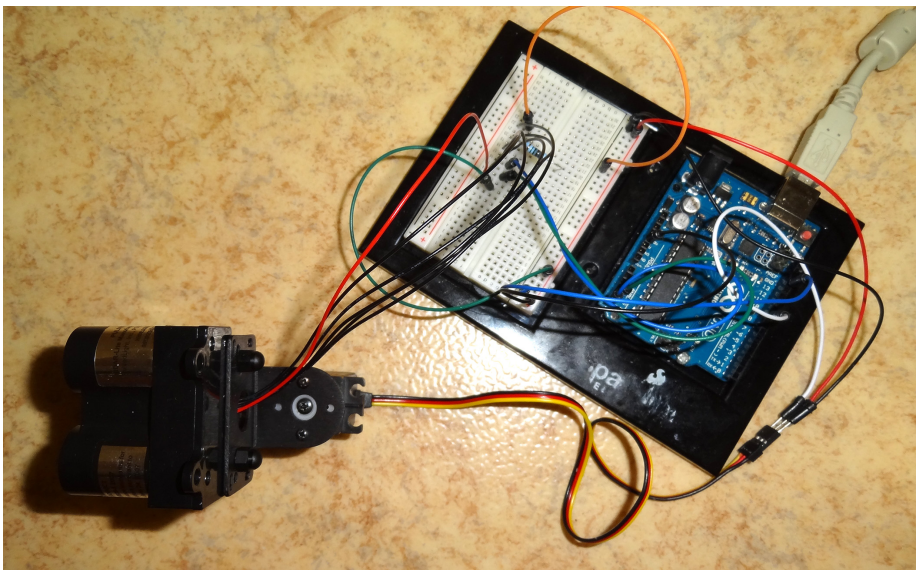


Fig.B.6 LIDAR-Lite, servo HS-422 deluxe connected to an Arduino Uno. The schematic of the circuit and a photo of the real implementation.

The connections required between the individual components for the proposed circuit are clarified in the following table (tbl.B.5)

Table B.5

(Connection between LIDAR-Lite, servo HS-422 and Arduino Uno)

LIDAR-lite	Arduino Uno
PWR EN	(unused)
MODE	(unused)
SCL	SCL
SDA	SDA
GND	GND

HS-422 Deluxe	Arduino Uno
SIGNAL	9 Digital PWM
5V V_{cc}	5V
GND	GND

The code, which has been developed for the presented applications, is included in the appendices section and the online repository (goo.gl/99Nxqj).

For the mapping application the user should first execute the Arduino sketch and afterwards the related Matlab file. There is also the requirement to add in the Matlab's path the *ransac_homography* library (goo.gl/5yR8IX).

For the proximity application the ROS and Ubuntu version depends on which board is used. Therefore, Arduino Nano is compatible with ROS Hydro and Ubuntu 12.04 LTS while the Arduino Uno is compatible with ROS Indigo and Ubuntu 14.04. It is observed that in the case of Arduino Nano (fig.B.7), it is better to provide external power supply at least to the servo since the system, if based only on the Nano feed, is unstable. In order to upload the code in the board the appropriate access permission should be set for this device. For instance, in Ubuntu the user need to type: *sudo chmod a+rw /dev/ttyUSB0*, if the board is connected at *ttyUSB0*. After that, and assuming that the *roscore* is run, the *rosserial* package can be executed by giving a command like: *rosrun rosserial_python serial_node.py /dev/ttyUSB0*.

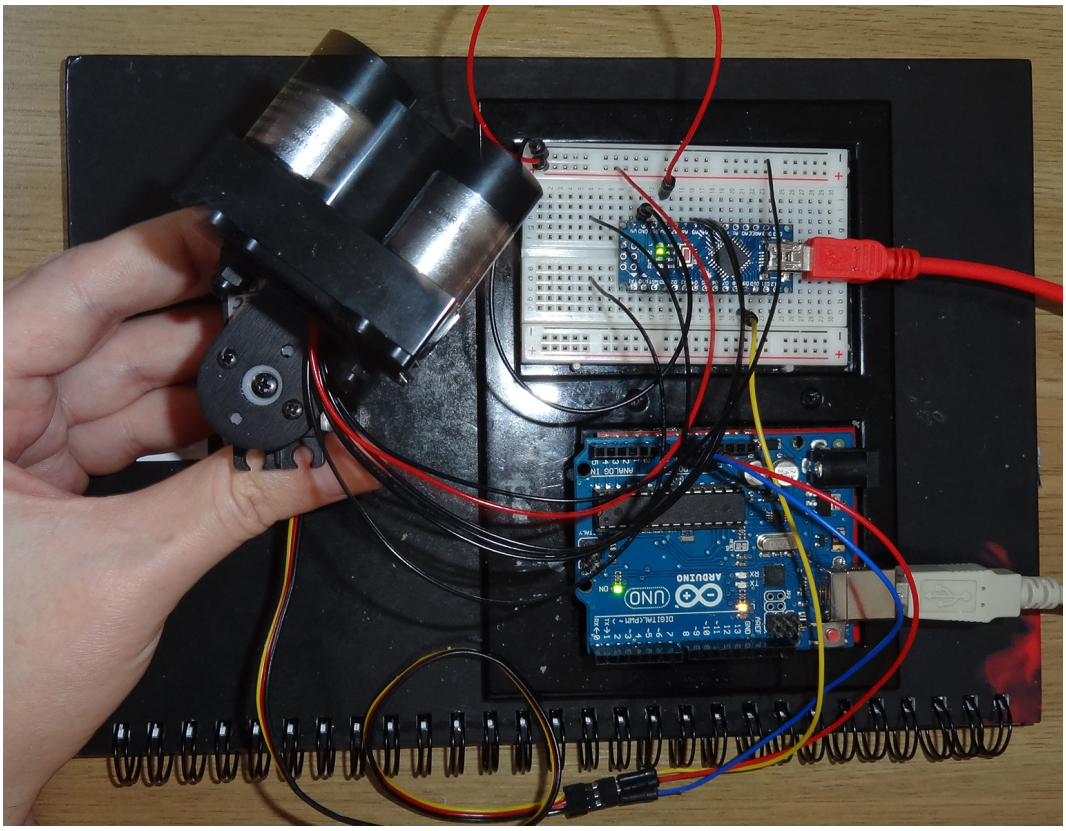


Fig.B.7 LIDAR-Lite, servo HS-422 deluxe connected to an Arduino Nano. Arduino Uno provides only power supply to servomotor.

Appendix C
Simulation Background & Configuration

An introduction to ROS and Gazebo simulation is presented at the top of the appendix. A useful terminology chapter is included ensuingly in order to briefly explain basic principles of ROS that are employed in the development of the obstacle avoidance simulation. Having introduced the underlying structure of the system that is utilised, a more retailed description of specific approaches followed in the development is clarified. Finally, a detailed description of the hardware, the operation systems and the tools distributions used is presented in order to allow the reader to reproduce a working copy of the simulation without any technical or compatibility issues.

C.1 ROS

The Robot Operating System (ROS) is a set of software libraries and tools that make it possible for the users to build robot applications. It is an open source project created by Willow Garage and maintained by the Open Source Robotics Foundation (OSRF). It is sponsored and supported from a wide range of well-known organisations and corporations such as NASA, DARPA, NSF and Bosch, Qualcomm and Mathworks.

The ultimate goal of using this framework is to enable rapid prototyping in robotics. Similarly the web 2.0 could be cited. By using tools like apache, MySql and python in a Linux environment, it was made possible to move from static web pages to dynamic or user-generated content and the growth of social media. Likewise this example, there is a need for a group of tools, compatible to each other that will boost the development in robotics field. Fortunately these days this is can be realized, by exploiting the work of many individual developers' communities; there are many sophisticated and well supported tools like ROS as a framework, Gazebo for 3D simulations, Stage for 2D simulation, OpenCV for

computer vision and many others more specialized in specific domains. All these packages are open sourced and structured in way that can be utilized easily by other platforms such as iOS and android.

ROS offers many advantages to the developers and based on these it has become a popular tool used in labs, classrooms and companies around the world. There is a 62% increase in the number of papers citing the ROS overview paper (Quigley et al., 2009) from August 2103 to July 2014 according to the community metrics report (Conley and Foote, 2014). Some of the strong points that make ROS a dominant solution in robotics are the following:

- It is an easy to use, cross-language (C++, python, Lisp) inter-process communication system that is fairly versatile. It is based on TCP-IP as well as shared memory communications.
- It allows easy integration of a wide range of tools such as visualization of robot kinematics and sensor data, path planning and perception algorithms. In addition, it includes a wide range of low-level drivers for commonly used sensors.
- It is a fully scalable platform that can support anything from simple ARM CPUs to Xeon clusters.
- There is a large international online community that offers support in possible problems and requests, registering a total of 18.144 questions in the duration of one year (Aug.13 – Jul.14). Something that indicates the growing popularity of ROS is the fact that this number of questions is 38% higher compared to the previous year, according to the community metrics report (Conley and Foote, 2014).

Aside from the loud advantages that ROS has, there are also inevitable drawbacks. From my perspective, it has a very steep learning curve; it is not easy for new users to grasp. The users should spend considerable time to install the packages in Linux machines and solve any dependency incompatibilities between the different versions and computer configurations. As a result at the beginning of the process a lot of effort is spent for the setup rather than the design and understanding of the algorithms. Despite it not being clearly stated on the software, having good C++ skills is a strong advantage. Moreover, there are only a few books available to learn it, although the ros.org website offers many tutorials. As a result there are not many teaching alternatives to experimenting with a working ROS installation and starting from the classic turtle

simulation, then moving towards more complex structures like manipulators and mobile robots. Two very helpful books related to ROS programming are (Goebel, 2014) and (Martinez, 2013). At the end of the day, it is all about effort and time.

C.2 Gazebo

Gazebo is a very popular tool in robotics simulation. Gazebo is a 3D simulator, while ROS serves as the interface for the robot. Combining both results in a powerful robot simulator. It allows rapid algorithm testing, robot design and regression testing using realistic scenarios. It can be used for both indoor and outdoor scenarios. In addition, it supports simulation of population of robots. Some of the great characteristics of Gazebo are the way by which it facilitates the integration physics engines and its high quality graphics. Furthermore, it offers convenient programmatic and graphical interfaces. Using Gazebo the users have access to a wide range of ready-made robot models and sensors with noise. Having said that, it is also possible for the users to use a specific language called SDF to build their own models and sensors too. Finally, by using Gazebo, users have access to multiple high-performance physics engines, which can easily simulate effects on illumination, gravity, inertia and other physic entities.

All in all, Gazebo is a great tool that every roboticist should have available in their toolbox. It speeds up the testing in difficult or dangerous scenarios without any harm to the robot. Most of the time it is faster to run a simulator instead of starting the entire procedure on a real robot. From my personal experience of using this tool, I concluded that it is that sometimes unstable and gives errors while other times it executes flawlessly without any change. It is computational-resources 'greedy' and highly affected by the graphics card's characteristics and capabilities. It is probably inadvisable to execute Gazebo in machines with low resources.

C.3 Terminology

There is a solid structure that a ROS system should follow independently of the programming language or the nature of the application. This allows us to have a simple standard code structure and to follow a schematic way of work.

Henceforth, there is a concise presentation of the ROS ecosystem (fig.C.1). References taken from (Bohren, 2015) and (ROS.org, 2015)

ROS Core

ROS core is a collection of nodes and programs that are pre-requisites of a ROS-based system. More precisely, it consists of three programs that are necessary for the ROS runtime. These programs are:

- *ROS master* which manages the communication connections. It provides name service in ROS.
- *Parameter Server* which stores persistent configuration parameters and other user-defined data.
- *Rosout* which is a network-based standard output for human readable messages.

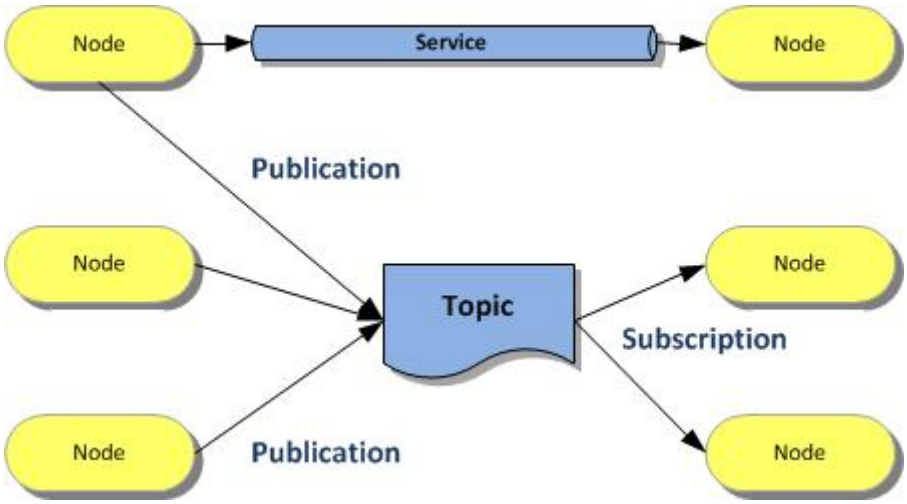


Fig.C.1 ROS structure of the basic elements (image from generationrobots.developpez.com)

Nodes

Nodes are executable files within ROS packages. They represent processes in a ROS network. Nodes use ROS client library to communicate with each other. They can publish or subscribe to a *Topic* and provide or use a *Service*.

Parameters

During runtime there are data related to the configuration and initialisation stored in the Parameter Server called *parameters*.

Topics

Topics can be perceived as data streams, asynchronous many-to-many network between nodes. It should be used for continuous data streams like sensor data or robot state.

Messages

As computer languages have supported data types, ROS supports *messages* used when subscribing or publishing to a topic.

Services

ROS nodes can exchange data not only using topics but also using *services*. Essentially, they are synchronous one-to-many network-based functions. *Services* should be used for remote procedure calls that terminate quickly. For instance, a *service* can be utilized for querying the state of a node or doing a quick calculation such as inverse kinematics (IK). Not using them for longer running processes is recommended. Furthermore, *services* cannot support processes that might be required to preempt if exceptional situations occur.

Actions

In cases where the service takes a long time to execute, the users might want the ability to cancel the request during execution or get periodic feedback about how the request is progressing. *Actions* provide the means to build specific servers that execute long-running goals that can be preempted. Moreover, it also provides client interface in order to send the related requests. Thus, if a server is executing two action goals simultaneously, for each client a separate state instance can be kept since the goal is uniquely identified by its id.

C.4 ROS development approach

In the “Simulation” Part of the dissertation I have clarified the reason why the ROS framework is going to be employed with the integration of the Gazebo simulator for the project’s simulation. On top of them, the Hector quad-rotor will form the basis for my UAV with various sensors, customised as required. Even if the users have reached this stage of decision-making, the way in which they are going to work is not predefined. This is owed mainly to the way that ROS is built, giving the users complete freedom to implement nodes as long as they can conform to the communication protocols.

My first attempt to start developing ROS system nodes for designing simple robotics algorithms was to use Matlab’s robotics system toolbox. This is new feature of Matlab (R2015a). Robotics Toolbox provides an interface between MATLAB, Simulink and ROS that enables applications testing and verification on ROS-enabled robots and robot simulators such as Gazebo. It supports C++ code generation. Thus, it enables the generation of a ROS node from a Simulink model and deployment to a ROS network. This is a great addition to the strict ROS terminal based interface. Matlab suite and scripting language is developer friendly and it can boost the research and productivity in higher levels.

The way that Matlab and ROS work is based mainly on the TCP/IP communication (fig.C.2). Although Matlab can autonomously execute ROS master and host the entire ROS ecosystem, most of the time users need to experiment with third-party ROS packages. To host these needs, it is assumed that Matlab is installed in a windows or mac host computer. To execute external ROS master, Matlab directs the users to download an Ubuntu virtual machine. After that, the users can normally execute any package in ROS master in the virtual machine and have access to topics and services from the Matlab in the host computer.

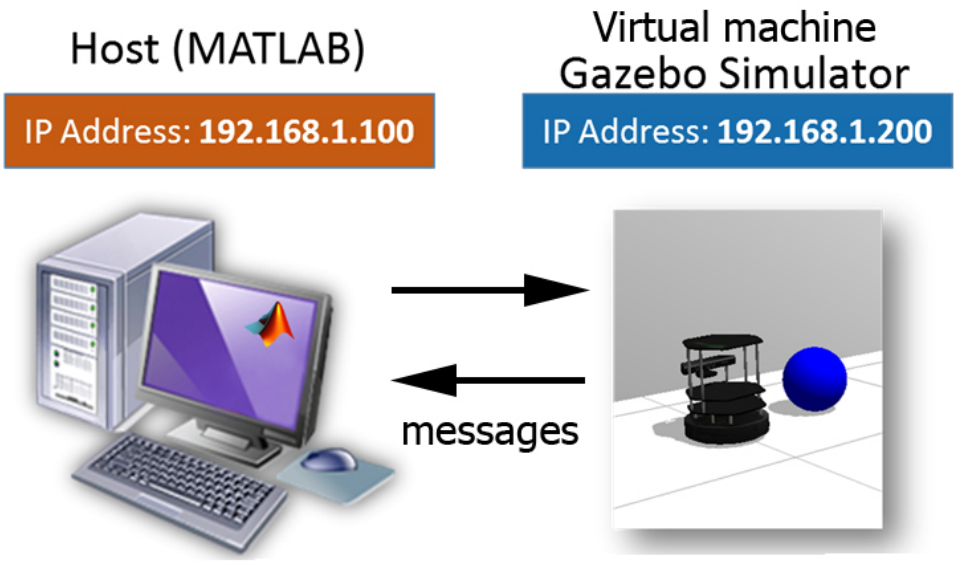


Fig.C.2 Interface between Matlab and ROS (image modified from www.mathworks.com)

The proposed way of work is convenient when there is a need to process data obtained from ROS nodes, transmitted using the topics. However, it is not straightforward to deploy more complex algorithms. From my experience in working with it, although I followed successfully all the learning material from the Matlab knowledge base, using the classic *turtlesim* simulation, when I attempted to get data from the Hector simulator I encountered problems. More precisely, Matlab couldn't receive laser scanning from hector quad-rotor. In addition, the set up of working in two different places at the same time makes the workflow more complicated. In my opinion, the interface between Matlab and ROS is a very strong combination, which can exploit the advantages of both successful platforms. Nevertheless, the implementation was just released and it needs time to mature with bug fixes and optimization.

For my project, I wanted a simpler approach. I preferred a setup implemented with fewer stages in order to eliminate potential point of failures. The solution to that was to work directly on a ROS installation in Ubuntu using the terminal and a simple text editor. In my first attempt I utilized python language, for some people the free counterpart of Matlab. Python is a high level programming language, with enhanced readability. It is an interpreted language rather than compiled like C++. This means that in ROS the users merely have to develop scripts without caring about the settings in the *CMakeList.txt* file which configures the

compilation. This is by itself a great advantage since these settings require a degree of experience to make them work correctly and efficiently.

The use of Python in ROS is officially supported and facilitates complex developments making the robotics rapid prototyping a reality. The syntax allows programmers to express concepts in fewer lines of code that it would be possible in languages such as C++. However, this level of abstraction and simplicity comes at a cost. From my experience, although the scripts were interpreted without any errors, the applications did not work. This is because there were errors in my code that were not reported from the lenient python interpreter.

The native ROS development language is C++. Using this language the users can develop everything they want in the ROS ecosystem. It requires having good programing skills in object oriented programming in C++. To compile the code the users should manually edit the settings in the *CMakeList.txt* file and *package.xml* file of the package. It is particularly hard at the beginning and even a simple task could take considerable time to accomplish. When the users acquire the needed experience, everything is simpler. On the other hand, the compiler is very strict compared to the python interpreter. If the users manage to compile the application successfully it is guaranteed that is going to work, unless it has logical errors. In addition, the compile error messages are self-explanatory and direct the users to the specific error. Finally, the syntax might not be so friendly like higher-level languages but it is concise and integrates powerful structures and tools. In short, C++ on a ROS installation was the development method that I adapted since it was the most straightforward and simpler, in terms of configuration, approach I could apply.

C.5 System configuration

The proper setup of the working environment in ROS and Gazebo is a very hard task. Factors that need to be considered when the system is configured are the following:

- Operating system

  It is recommended to use Linux. Actually, Linux is the only OS that is officially supported in ROS. The installation debian packages (unix archives) for ROS are available for Ubuntu Linux.

- ROS version

  Following a developing cycle similar to Ubuntu, ROS releases new versions every 6 months to one year. At the time of the project implementation, the latest was Jade (May 2015). However, because of compatibilities issues a hydro version was used. This version was released in September 2013. Making this decision I was restricted to install Ubuntu 12 LTS (Long Term Support) as operating system in order to gain maximum compatibility.

- Gazebo version

  Since ROS Hydro, Gazebo is considered a system package instead of a ROS package. The practical meaning is that one major version of gazebo is selected at the beginning of the ROS release cycle and remains the same during the whole life of the ROS distribution. In my case ROS Hydro is compatible with Gazebo version 1.9.

- Hardware

  Hardware is also a parameter in simulation system. In case that the ROS is combined with Gazebo, there is a need for at least 4GB RAM. Gazebo requires a working graphics card with OpenGL 3D accelerated driver to perform various rendering and image simulation tasks correctly. For the current project, an extensive use of virtual machines was engaged. Particularly, I utilize both VMware Fusion and VirtualBox software supervisors. Moreover, there are available virtual machines with pre-installed ROS distributions, which could be easier for the beginners. All in all, both virtualisation methods performed well, supporting the ROS ecosystem.

To summarize, my setup configuration based on virtual machines having Ubuntu 12.04 and 14.04 as operating system using ROS Hydro and Indigo respectively.

Appendix D

Simulation Development

In this part, technical details about the obstacle avoidance application development in ROS are presented. At the beginning, the configuration of the input devices is explained. This is followed by a detailed description of the required topics nodes and actions. Finally, having first presented the required infrastructure, the set up of the application in terms of the inputs and the outputs is defined. If the reader is not familiarised with the ROS terminology, there is a useful guide in the "Simulation Background & Configuration" appendix that explains all the related terms involved in the application development.

D.1 Input devices

Generally speaking, the control of the UAV can be implemented by designing a publisher that transmits messages of type *geometry_msgs/Twist* in the *cmd_vel* topic. A recommended way to tele-operate the UAV is to use a joystick. The hector quad-rotor comes with a node that is configured to utilize the xbox controller for tele-operation. In order to achieve continuous movement, which is disabled for safety reasons, a specific parameter, *autorepeat_rate*, should be enabled and set to the required rate. The edited launch file can be found in (goo.gl/b63Ugr). Further than that, the throttle should also be adjusted to move the UAV smoothly. The required changes in the teleoperation node to achieve this can be found in (goo.gl/K173HW).

There is also a simpler way to control the UAV using the keyboard. To use the keyboard for the UAV manipulation the users should install an appropriate package that includes the required nodes. One option is to use the ROS package (pr2_apps) developed by Willow Garage for the PR2 robot. Once the package has installed successfully, by executing the *teleop_keyboard.launch* file the users can control the translation on the x and y axis, the yaw angle and the speed.

D.2 Topics nodes

The key feature of the radiation mapping is the capability to maintain a constant distance from the floor, even if this is the ground or the terrace of a building. The sensor that was employed for this task was the sonar, which is mounted under the central hub looking downwards. This device in the simulation publishes continuously, with a rate of 10Hz, the measured distance. It has a range from 3cm to 3m.

In order to implement the required behaviour, I designed the hover node (goo.gl/vTajQO). Essentially, this node gets input from the sonar to access the actual altitude of the UAV. In addition, it takes input from a custom topic *hover_height* with messages of type *geometry_msgs* (fig.D.1). This is the interface of the node with the ROS ecosystem. Every node that wants to adjust the altitude hold of the UAV can set up a publisher for this topic and send the appropriate message to the hover node. Inside this message the z component commands the node to lock the altitude in this specific value.

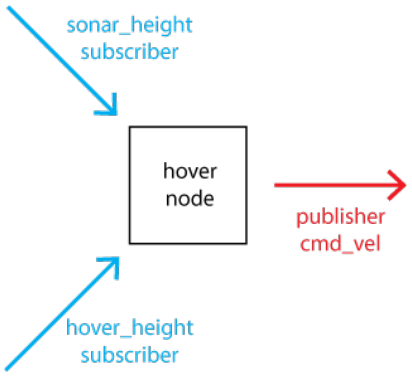


Fig.D.1 The messages' flow for the hover node

Since the node has all the data needed for its operation, it proceeds to the requested behaviour. Technically, both subscribed topics are connected with callback functions. The *heightCallBack*, related to *hover_height* receives the desired altitude hold and checks if it is inside the range of the ultrasonic sensor. In the second callback function *heightCallBack*, which corresponds to the *sonar_height* topic, the real work of this node takes place. Inside this function there is a simple P-controller (fig.D.2). This controller calculates the difference of

the desired and the current height and outputs a value adjusted according to a gain. This outcome is basically the linear velocity in the z direction. Consequently, using this value a *geometry_msgs/Twist* message is prepared and sent to the *cmd_vel* topic. As long as the altitude of the UAV is stable and equal to the desired, the node does not need to act. It is worth noting that this node is active throughout the simulation since the altitude hold is a requirement in the flight mode. To achieve this automatically, the relevant command has been included inside the simulation launch file, for example in (goo.gl/jBtQsi).

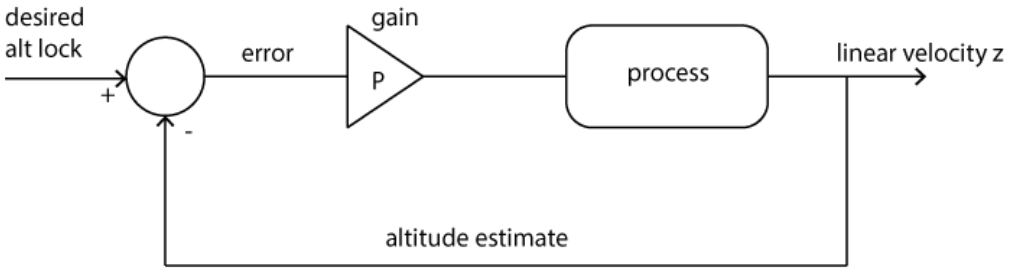


Fig.D.2 The schematic of the P-controller inside the hover node

Besides the hover node, a node responsible to change the heading (the yaw angle) is also developed called turn node. The operation principal is the identical to the hover node. The difference is that it gets the reading from the IMU sensor and publishes an angular velocity in the z axis instead of a linear velocity. A figure that summarizes the subscribed and published inputs is depicted in (fig.D.3). This node, based on topics, was replaced at the final obstacle avoidance application from a node, based on actionlib, for reasons that will be explained further on.

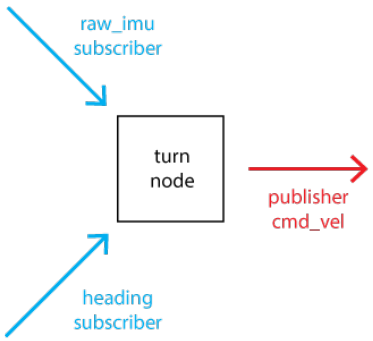


Fig.D.3 The messages' flow for the turn node

D.3 Actions nodes

In the introduction of ROS, it was described that there are many ways for the nodes to communicate with each other. Aside from the topics and the services, actionlibs is a powerful tool to design communication channels between nodes. In particular cases, when the following conditions exist, there is the only way.

- When the task is required to take time and feedback of the progress is needed, as well as a sign of completion or failure at the end. In this way, the application knows exactly the state of the ordered task and when the correct time to continue has come. This cannot be implemented using topics or services.
- When the application needs a way to cancel the ordered command.

For these reasons I decided to build action servers nodes in order to change UAV's heading and altitude lock. There is a predefined way that an actionlib server is built. In this section I will explain how the change-heading server was designed. First, in order for the client and server to communicate, there is the need to define three different types of messages on which they communicate. More precisely, the Goal, Feedback, and Result messages with which clients and servers communicate should be defined.

- Goal: To accomplish tasks using actions, the notion of a goal is introduced that can be sent to an action server by an action client. In my case the desired heading (in radians) is sent.
- Feedback: This message provides server implementers a way to tell an action client about the incremental progress of a goal. In my case, the difference between the desired heading and the current heading is published.
- Result: A result is sent from the action server to the action client upon completion of the goal. The difference with the feedback is that it is sent only once. This is also useful when the purpose of the action is to provide some sort of information at the end. In my case the result message means the successful completion of the ordered task and it is the accomplished heading.

Technically, to create new type of messages there is a need to define them in a specific action file. Then this file should be properly declared in the *CMakeList.txt* file of the package. The next important step is to execute the *catkin_make* process. Upon the completion of this, the required C++ header files containing the messages definitions have been created in a specific folder in the devel section of the ROS project. Now, the developers can use these messages merely by including the required headers.

Since the infrastructure is complete, the design of the server can be implemented. For this particular server there is a need to know the actual heading. An easy and reliable way to obtain the heading is to employ the IMU sensor. Consequently the heading-change server has to subscribe to the *raw_imu* topic. In this topic *raw_imu* transmits quaternions. Using a couple of convenient methods it is possible to convert a quaternion to roll-pitch-yaw and access the UAV heading.

Following the creation of the server, three required function callbacks should be declared (fig.D.4). The first one is responsible for setting the goal in the action server. In my case, the *goalCB* callback function sets the desired heading. The second one is the preempt callback function. This action is event driven. It only executes when the callbacks occur, therefore a preempt callback is created to ensure that the action responds to a cancel request. In my case this callback function called *preemptCB*. The third callback function is the IMU callback function. Inside this function the change of heading takes place. At the beginning the current heading is obtained, as I described before. Then, the difference between the actual and the goal is calculated. At this point the feedback message is published. After that, this difference combined with a constant gain, form the input of a P-controller (fig.D.5). The outcome of this procedure is the angular velocity in the z axis. As it is expected at this point *geometry_msgs/Twist* message is created and published to the *cmd_vel* topic. At the end of this method, a termination condition is evaluated. More precisely, when the absolute difference between the desired and the actual heading is less than 3 degrees, the procedure terminates successfully. At this point the result message is published, only once. The reason why I introduce a 3-degree angular tolerance is that it is very easy to overshoot the rotation and even a slight angular error can ruin the operation of the entire application. Empirically it was found that a tolerance of about 2.5 degrees gives acceptable results. The

complete source file of the change-heading server can be found in this location (goo.gl/qbZDVR).

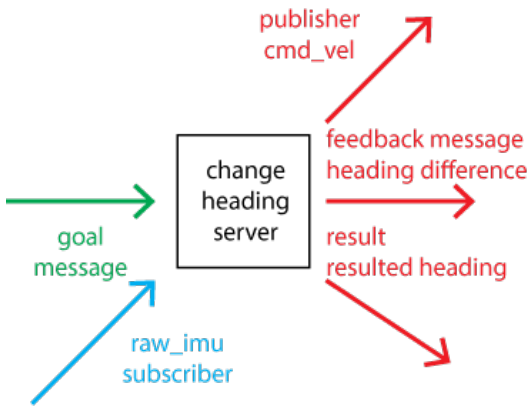


Fig.D.4 The messages flow for the change-heading action server

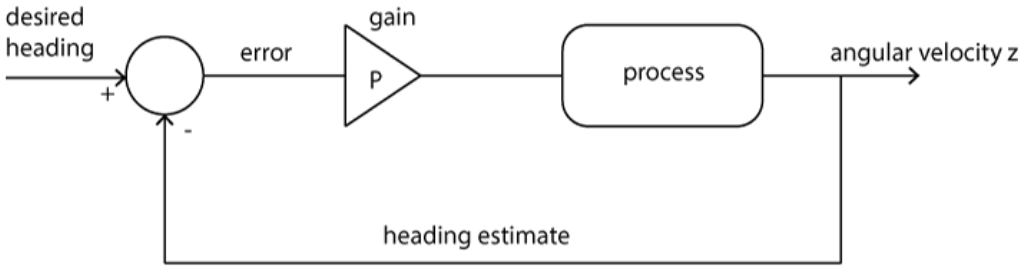


Fig.D.5 The schematic of the P-controller inside the change-heading server

Before the action server was utilized as part of the application, the debugging and testing stage could be performed using a simple action client (goo.gl/LblbEF). In this client the goal heading is sent to the server. After that, the timeout duration is defined. During this period the client receives and reports the feedback messages. If the procedure ends within the time limit, it is considered successful otherwise the action has failed and an appropriate message is printed out. Since the actions are more complex than the other approaches, for every individual action server I have developed a suitable action client.

An additional important action server of the application is the hover-height server. This server is responsible to change the altitude to the new desired value, which has been set as goal. The underlying logic and the structure of source file are similar the change-heading server that I have already described. The few differences that exist are the following as depicted in the figure (fig.D.6).

- The device that has undertaken the task to inform the current height is the sonar.
- The action server does not implement any controller. Instead of this, it takes advantage of the hover node and sends the required message to this mechanism in order to move the UAV vertically.

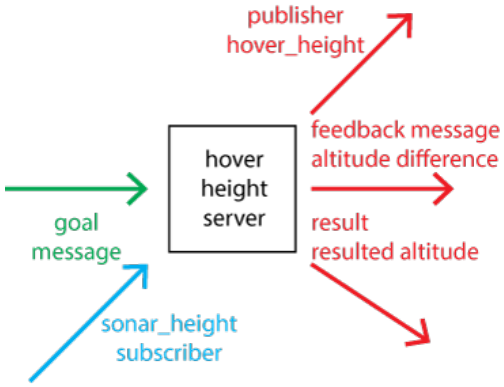


Fig.D.6 The messages' flow for the hover-height action server

The importance of both action servers in the obstacle avoidance algorithm will become apparent when the application will be explained. Nevertheless, before settling on the final version I had experimented with other implementations as well, which I included in my source code as available tools in the toolkit. The first one is a server (goo.gl/k6OjVX) which -instead of the desired altitude- accepts a vertical distance as its goal. For instance, if the goal is 1 the hover-distance action server will move the UAV 1m upwards. The second one is a server (goo.gl/KhdlHm) that performs the same action as the hover-distance but the measured height depends on the GPS altitude instead of the sonar's distance from the ground.

D.4 Obstacle Avoidance

Input devices such as the keyboard or joystick send control commands, including the linear and the angular velocity in x, y and z direction. These commands are not transmitted directly to the UAV as it traditionally happens. Besides, the obstacle avoidance application intervenes in the path between the input control and the UAV, or the flight controller, to be more precise. This application receives inputs from the proximity sensors and knows at any single moment the

measured distances around. Having this information, it can assess dangerous situations, caused by objects in close proximity, and apply the appropriate obstacle avoidance manoeuver in order to overcome it. During this procedure the control is performed by the application that sends appropriate control commands to the flight controller and not by any commands issued by the operator. Once the application has gone through the avoidance maneuver, it then corrects the heading back to the UAV's initial one and returns the control to the operator.

The aforementioned approach can be translated technically using the following techniques (fig.D.7). The control input devices send traditionally the *geometry_msgs/Twist* commands in the *cmd_vel* topic. To enable the obstacle avoidance application to alter the predefined flow of the commands, I structured it as a message getaway or a filter. To illustrate this, the input control devices have been remapped to publish the navigation commands to the *obs_cmd_vel* topic instead of the initial *cmd_vel*. The *obs_cmd_vel* is the one that the application listens to. If there is no need for intervention, the node is transparent. Whatever navigation message comes from the inputs, it is published automatically in the *cmd_vel* topic. If there is need to perform avoidance maneuver, the node overrides the control and sends the algorithm's messages as input to *cmd_vel* topic.

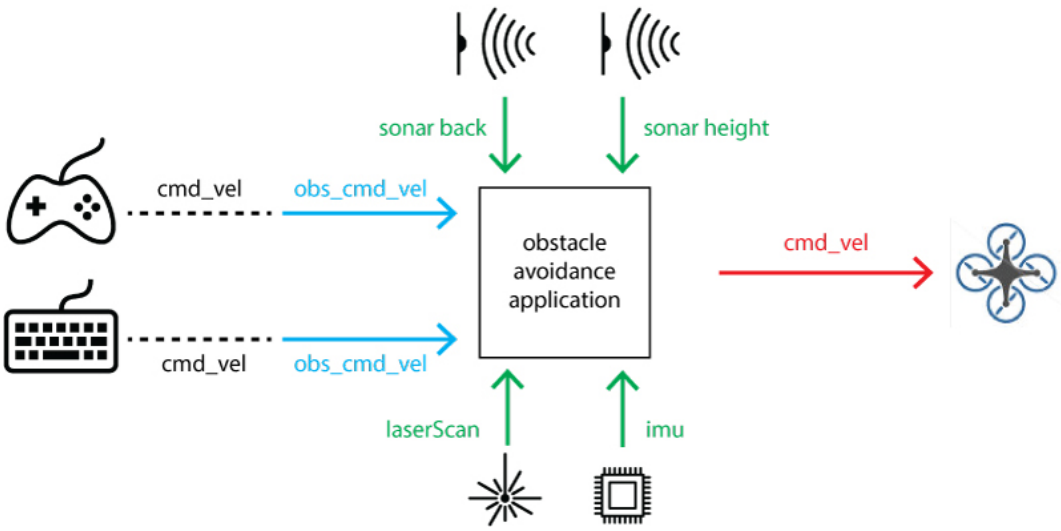


Fig.D.7 The messages flow for the obstacle avoidance application

The source file in which the obstacle avoidance application has been implemented is located in (goo.gl/eG2mXn). Details about the operation of the

finite state machine, which is the embedded mechanism inside the application, are explained in the “Simulation” part of the dissertation.

Appendix E

Mobile Applications Development

In the “Human Robotic Interface” descriptions of the concepts of two applications are presented. The first is related to the control of a ROS enabled UAV and the second to the radiation mapping. Technical details about the code development and the required configurations are illustrated in this part. Guides about the functionality of both applications are presented in the “Mobile Applications Functionality” appendix.

E.1 UAV Teleoperation application

My scenario is based on the Apple’s iOS device since I have a previous experience on coding for this ecosystem. The prototype application is visually structured for the layout of iPhone 5 (4 inches diagonal, 1136x440, 326ppi). The targeted version iOS version is 8.

The implementation of the application was based on three *UIViewControllers* embedded in navigation controllers (fig.E.1). The navigation between the screens was accomplished using segues. In the projects’ workspace, three external libraries have been imported, namely:

- SocketRocket
- RBManager
- MKMapView

As long as the control tab is enabled, the control messages are published in a continuous rate of 10Hz using an *NSTimer* object.

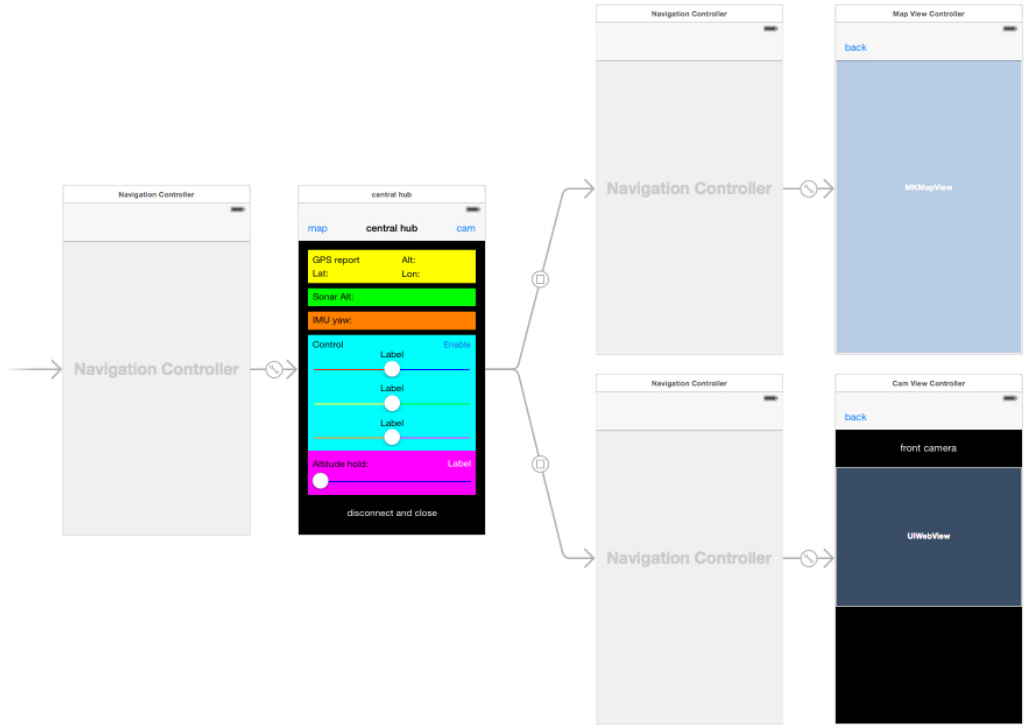


Fig.E.1 The storyboard of the teleoperation application

On the other side ROS was executed in a VMWare Virtual Machine. The ROS distribution was Indigo and the operating system Ubuntu 14.04 LTS. The simulation was based on Hector quad-rotor ROS stack. The host computer was connected with the iPhone via a Wi-Fi access point. The network topology is shown in figure (fig.E.2).

For a successful working scenario the *my_hector* package, which was developed for the assisted obstacle avoidance flight mode, had to be installed. Particularly, the hover node had to be executed to allow the altitude hold mode (*rosrun my_hector hover*). In addition, the *Rosbridge* and the *MJPEG* server also had to be turned on and to be running using the following commands:

roslaunch rosbridge_server rosbridge_websocket.launch

rosrun mjpeg_server mjpeg_server

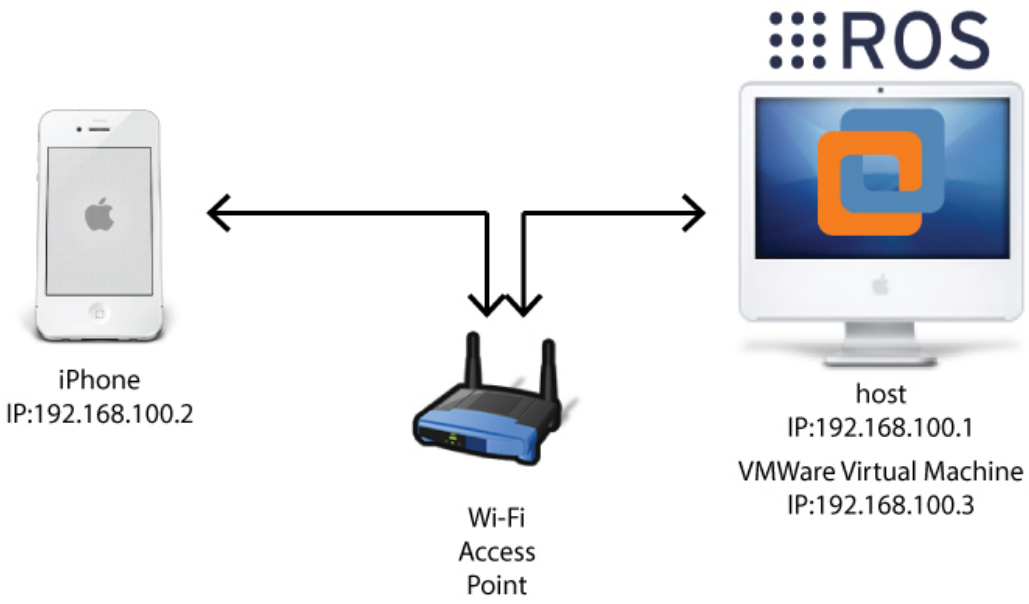


Fig.E.2 The network topology of the development environment

The iOS source files of the project are available online in the project's github repository (goo.gl/vq1U6m)

E.2 Radiation mapping application

Similarly to the previous one, the radiation mapping application was developed based on the iPhone 5 screen dimensions. Consequently, the layout and the size of the elements are adapted to this device. The minimum supported iOS version is the 7th.

The implemented application consists of three *UIViewControllers* embedded in navigation controllers (fig.E.3). The main data structure that holds the mapping data is an *NSArray.* The elements, which the mapping array stores, are objects that belong to the custom class *Measurement.* The advantages of using an array are: the potential future integration of a save and load mechanism as well as the easy handling of data as a set in the map visualization (redraw/clear).

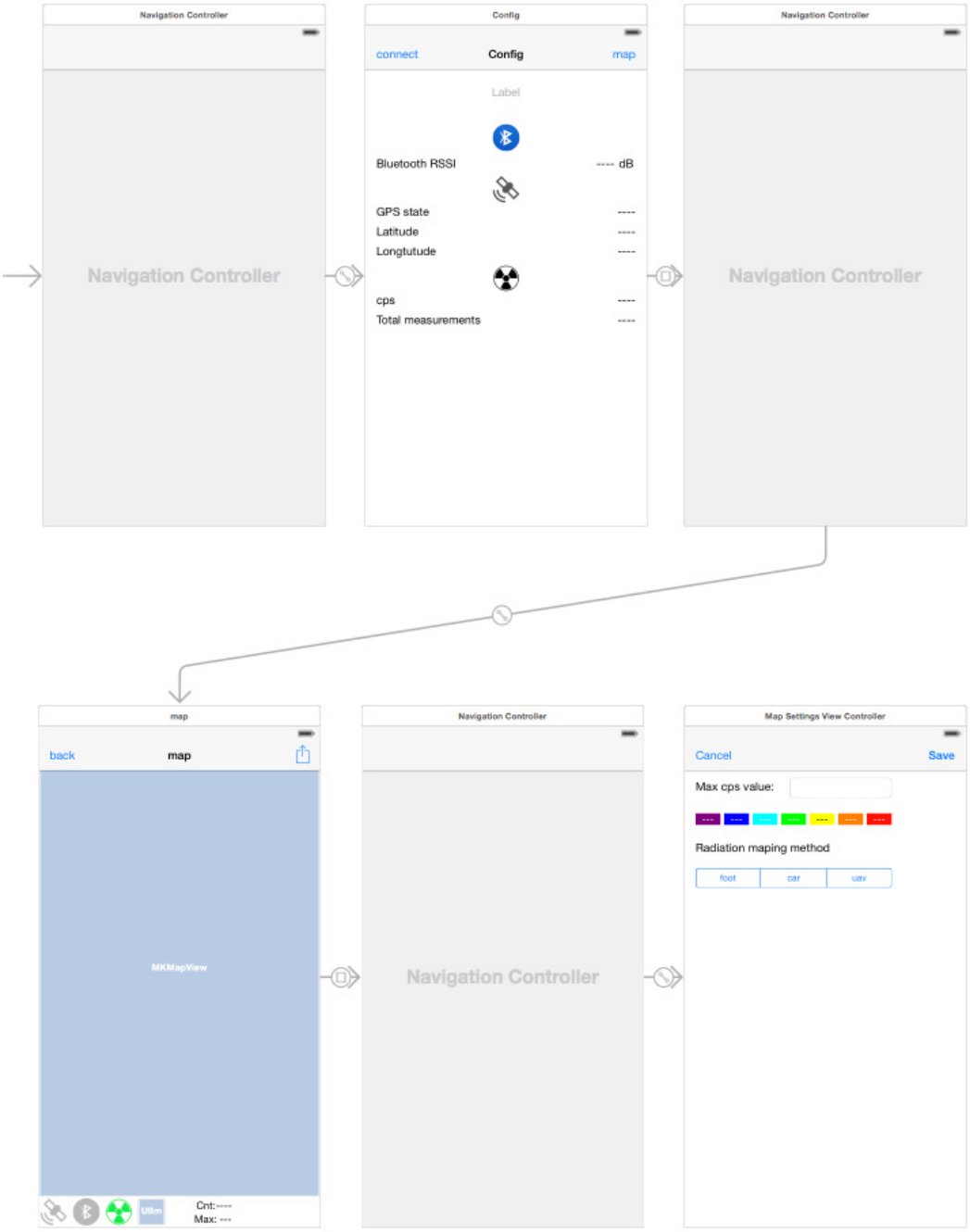


Fig.E.3 The storyboard of the radiation mapping application

The first screen is called “Configuration”. The related *UIViewController* is mainly responsible to handle the Bluetooth connection. It includes the entire set of the related methods, which are provided by the BLE library. When the user clicks the “Connect” button, using the appropriate sequence of methods’ execution, the Bluetooth connection is activated if a compatible device is enabled within the range. Afterwards, having established the connection, the application receives the messages, which are transmitted from the measurement unit. The

UIViewController has all the required tools to parse the payload of the messages. More precisely, at the beginning, two data structures are defined in order to handle binary floating-point numbers (32bit) and binary integer numbers (16bit). When the specific bit of the status byte gets enabled, the parsing of the raw bytes begins. The payload of the message contains information about the number of connected satellites, the longitude, the latitude and the CPS value. The last three elements are stored in a newly created *Measurement* object and then, in turn, this object is stored in the mapping array.

The next screen is the map. It can be switched on, on that screen, only if the GPS is locked. One noteworthy mechanism that has been implemented is the share of the same BLE manager with the previous *UIViewController*. Particularly at the transition, the same BLE manager is passed to the map screen. This is desirable because map screen needs to receive messages and status information for the summary section. Before the transition is made, the reception of new messages is disabled in the previous screen and enabled in the new screen and vise versa for the reverse switch. In case of having individual BLE managers means that in the new screen the Bluetooth connection would have to be established again, which is unacceptable. Not only do the two screens share the BLE managers but they also share the mapping array. Consequently, this array is unique in the application and is passed to each active *UIViewController.*

Another important characteristic in the implementation of the map screen is the approach that enables the individual color of each point. The standard method of using *MKCircle* overlays does not support the selection of color for each point. It is possible to select only the color for the entire overlay. To extend the provided class, I used class inheritance and I sub-classed the *MKCircle* to create *MyCircle*. This approach allowed me to use the same standard visualization mechanism and at the same time satisfy the enhanced requirements.

The last noteworthy task implemented in this screen was the adaptive zoom level on the map. According to the value that the user selects in the next screen, the zoom level and the size of the circle, which represent the measurement, change. To reflect the change, all the measurements that are contained in the mapping array need to be redrawn. This is feasible using the developed helper methods.

In the third screen the user is provided with the option to select the maximum

range of the CPS values color range and the mapping methods. The functionality is simple and when the user clicks the save button, the selected values return to the map screen and update the related features accordingly.

All in all, the navigation between the *UIViewControllers* is implemented using segues. The exchange of the data between them is accomplished using delegate methods.

On the other side the measurement unit consists of an Arduino Mega ADK as the base microcontroller and three shields (fig.E.4). The installed shields are namely:

- RedBear's lab BLE Shield
- Wireless SD Shield
- Adafruit Ultimate GPS

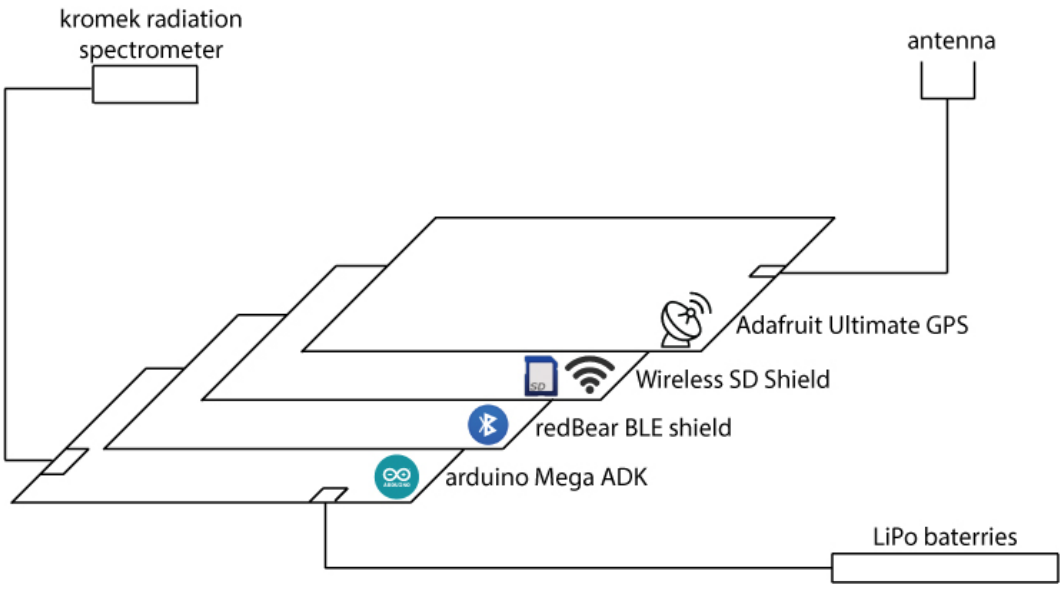


Fig.E.4 The hardware profile of the measurement unit

The Arduino sketch includes many external libraries in order to successfully support the operation of all the individual interfaces. The required libraries are:

- Adafruit GPS Library
- BLE
- RBL nRF8001
- TinyGPS
- USB Host Shield v2.0

One of the features implemented in this Sketch is the break of the native data types to bytes in order to compose the payload of the message. The approach is very similar to what is implemented in the iOS side. Likewise, two data structures are defined in the beginning of the script. The first supports binary 32bits floats and the second binary 16bit integers. The data, using this mechanism, are broken to bytes and sent sequentially by employing the BLE shield.

It is important to cite that the initial IAC sketch was compatible with the Arduino 1.0.x IDE and not with the most recent 1.5.x and 1.6.x. versions. In order to avoid the required time to build the sketch from scratch, since it was not working with the newer versions, I adapted the 1.0.x version by downgrading the BLE and the RBL nRF8001 libraries. This way I maintained the compatibility, in order to have a working set of code.

The iOS and Arduino source files of the project are available online in the project's github repository (goo.gl/vrfU9L).

Appendix F
Mobile Applications Functionality

A guide about the functionality of each application is presented in this part. An explanation of every element presented in the applications' screens as well as a description of the available configuration settings are included. A detailed presentation regarding the technical details of the applications' implementation is presented in the "Mobile Applications Development" appendix.

F.1 UAV Teleoperation application

The teleoperation application is a prototype of a fully working application yet has not optimised in areas such as functionality and appearance to be commercially available. The application consists of three main screens. The first screen appears when the application is loaded as the central hub (fig.F.1). From this point, the operator has a complete overview of the sensors' feedback. It shows data collected from the GPS unit, the IMU unit and the sonar, which measures the distance from the ground. In addition, in this screen the operator can enable the control tab. This tab provides the ability to control the linear velocity in the plane, x and y direction, as well as change the heading by manipulating the angular velocity in the z direction. This tab includes also the altitude lock mode. More precisely, the user can adjust the constant distance from the ground in which the UAV flies. Whenever the operators need to freeze the UAV (zero the velocities components of the control), they just have to toggle the enable/disable button. In this procedure the altitude remains constant. Finally, it contains a button to disconnect from the UAV and to terminate the application.

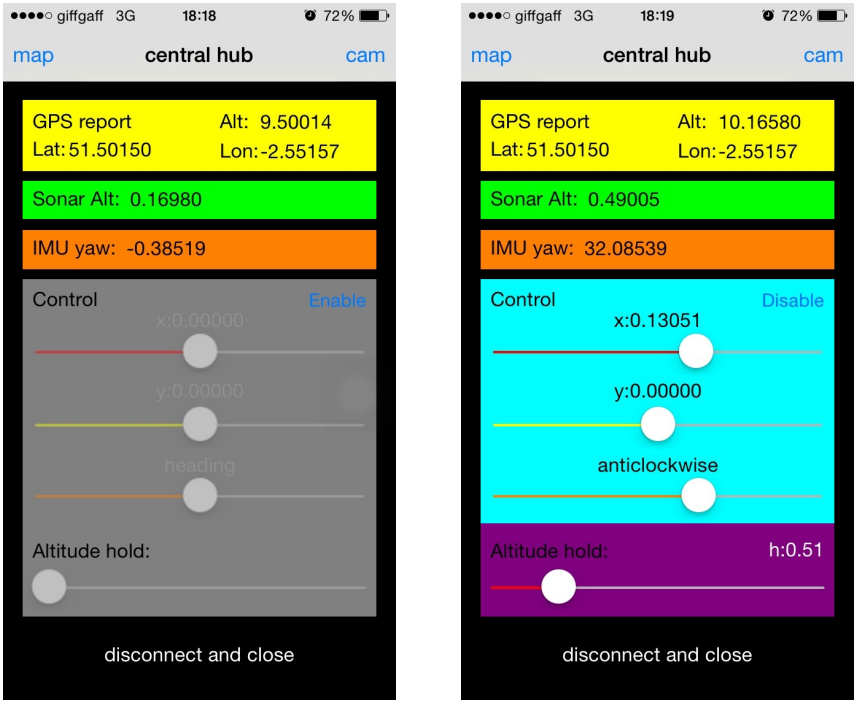


Fig.F.1 The central hub screen with control tab enabled and disabled

The second screen shows the exact location of the UAV in a map (fig.F.2). It takes the longitude and the latitude from the on-board GPS unit. The third screen shows the streaming video of the on-board front camera (fig.F.2). The user having locked the altitude could also drive the UAV from this screen.

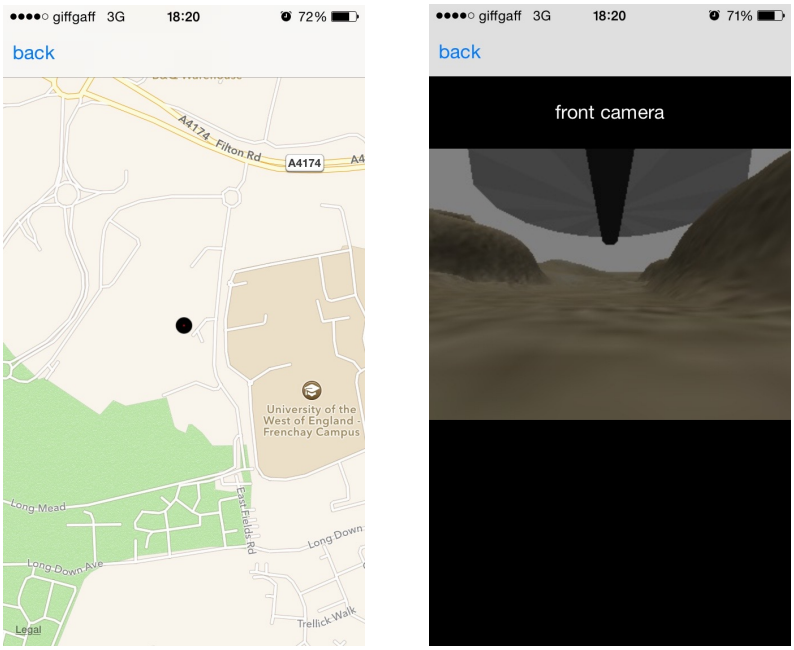


Fig.F.2 The map and the camera screen

F.2 Radiation mapping application

The radiation mapping application has been developed using the directions of the experienced IAC researchers. It is a very specific application and in order to be useful and a real tool in the hands of the user, it has to be specifically defined and designed according to their needs.

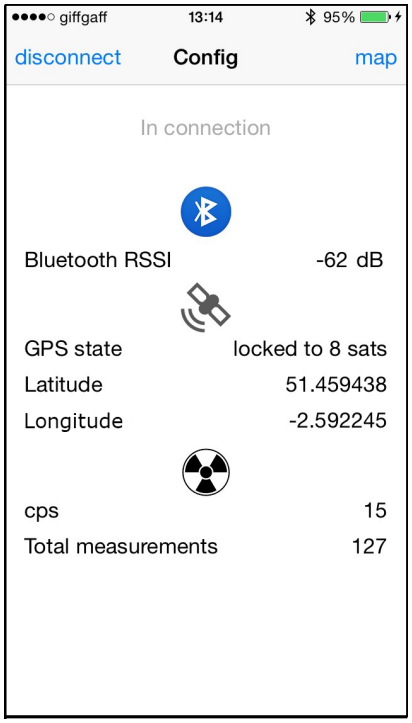


Fig.F.3 The configuration screen of the radiation mapping application

The configuration screen displays all the useful information to the user (fig.F.3). In the fist part of the screen there is a message board that displays informative messages regarding the Bluetooth connection. In this display, the user can be informed about the availability of other BLE devices, the name of the device that they are eventually connected to and other relevant information. After a successful connection the RSSI value is continuously updated to reflect the signal strength of the Bluetooth connection between the measurement unit and the iPhone. At this point, with an active Bluetooth connection, the user can read the status of the measurement device in the iPhone screen. Consequently, when the GPS unit locks, both the relevant message and the number of the connected satellites will be displayed to the screen as a sign of the signal strength. Finally, there is a section related to data which come from the radiation spectrometer. In

this part, the actual CPS value is reported together with total number of measurements that has been performed.

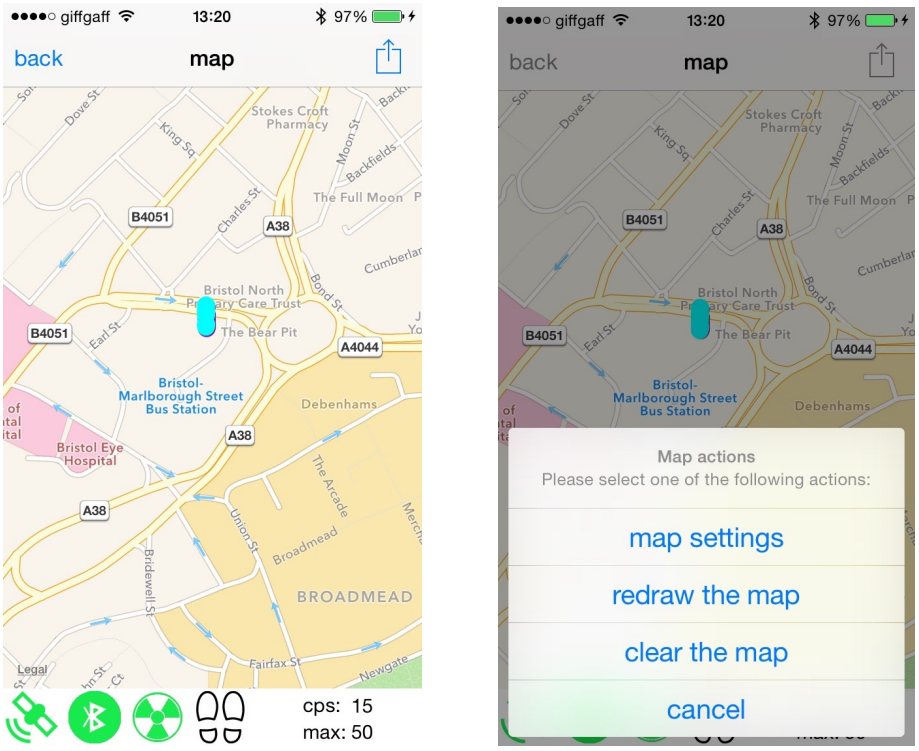


Fig.F.4 The map screen (left) and the map screen with the actions button enabled (right)

The next screen displays the visual representation of the measurements in a map (fig.F.4). Every time a new measurement is received, the application stores this value in an internal data structure and at the same time it displays a colour-encoded point at the exact longitude and latitude in the map. Colour depends on the CPS value. At the lower part of the application there is an information tab. Since the users spend most of the time on this screen, they need a way to know the overall status of the system. The summary part, which is available, contains the following:

- Three status icons corresponding to GPS, Bluetooth and radiation device connectivity. The first two are green when the connection is active, otherwise they are grey. The third one is animated like a heart pulse to indicate that the CPS storing and visualisation proceeds flawlessly.
- There is an additional icon in the tab that represents the method of the radiation mapping. Furthermore, there are two character strings

displaying information about the current CPS value and the visual colour-encoded CPS range.

There is a button in the navigation part of the screen, which contains a set of map related tasks. Using this button the user can clear and redraw the map. An additionally important task is the display of the map settings screen.

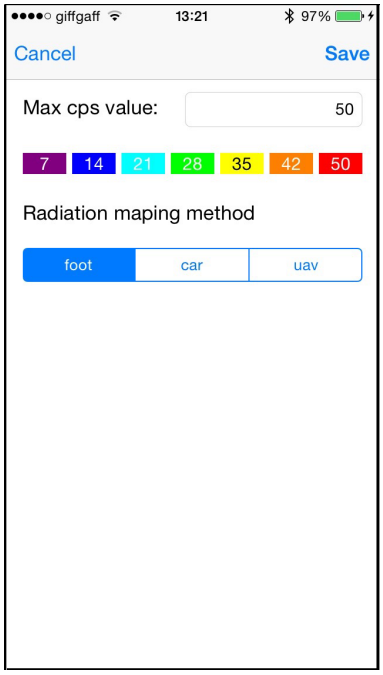


Fig.F.5 The map settings screen of the radiation mapping application

In the map settings screen the user can adjust various visualisation parameters (fig.F.5). Firstly, there is the option to select the radiation mapping which is used to perform the measurements. There are three methods available, measurements taken on foot, by UAV or using a car. According to the selection, the zoom level of the map is changed to allow a reasonable display detail level. Moreover, in this point an attenuation factor can be added since there is a different sensitivity level if the user takes measurements inside of a car or on foot. Secondly, the user can adjust the colour encoding of the measurements. More precisely, the red colour represents measurements close to 500 CPS and purple close 0. In cases where the area is considered safe the colour will remain purple without the ability to represent minor fluctuations of 10 to 20 CPS difference. Thus, in this screen the user is able to define the maximum CPS value and based on this value the CPS range is rescaled accordingly. As a result, with a maximum of 50 CPS the range of each colour will be approximately 7 instead of

the initially 71 for the 500 CPS. All in all, when the user returns to the map the measurements are redrawn reflecting the new colour range.

Appendix G

UAV configuration

One of the most important parts of the project was to build a state of the art UAV. There were many different requirements that the configuration should satisfy. The primary demand was to use commercial off the shelf (COTS) products in order to save time from re-implementing successful existing parts and simplifying the replication due to the practical design. Further than that, open hardware products were chosen so that it would be easy to get support from the related online communities and it would also be able to choose from different manufacturers with lower prices. In general, COTS components tend to be inexpensive, upgradable and reconfigurable. Moreover, it is easy to gain experience even before starting to experiment, since many researchers have already used them and published relevant papers. Aside from the general requirements, specific research has been conducted for each individual part in order to use the most appropriate component and how this item can be combined with the others in the most efficient way.

G.1 Chassis, brushless motors and Electronic Speed Controllers (ESC)

G.1.1 General description

The most basic and influencing component in a UAV is its chassis. There is a great range of available options according to the number of motors and their topology. For the current project, a quad copter has been chosen. Some of the advantages of selecting quad copters are the low price, the great manoeuvrability and the enough available power to add accessories since they have only four motors. The option of using a tri-copter has been discarded due to its limited trust and power capabilities. On the other hand, hex or octo-copters can fly in higher heights and greater flight speeds, are safer if one motor dies, since the rest can pick up the slack and also can handle higher overall payloads. These

are considerable advantages but for the specific project these do not affect the objectives and add only unnecessary complexity.

Regarding the motors of the UAV, the most popular type is the brushless motors. Brushless motors are much more efficient than the conventional brushed ones. They have a greater lifetime and they do not require maintenance due to the absence of brushes and commutators. In addition to that, the absence of brushes eliminates the friction and thus they have very high power density (in terms of size and weight), which means longer battery life. In general, brushless motors are very easy to control their speed accurately and adjust it quickly. Nowadays, the prices of the brushless motors are similar to brushed so they are not expensive as they were in the past. Finally, it generates less noise that causes weaker electrical magnetic interference (EMI), which is very important for the interference of the GPS.

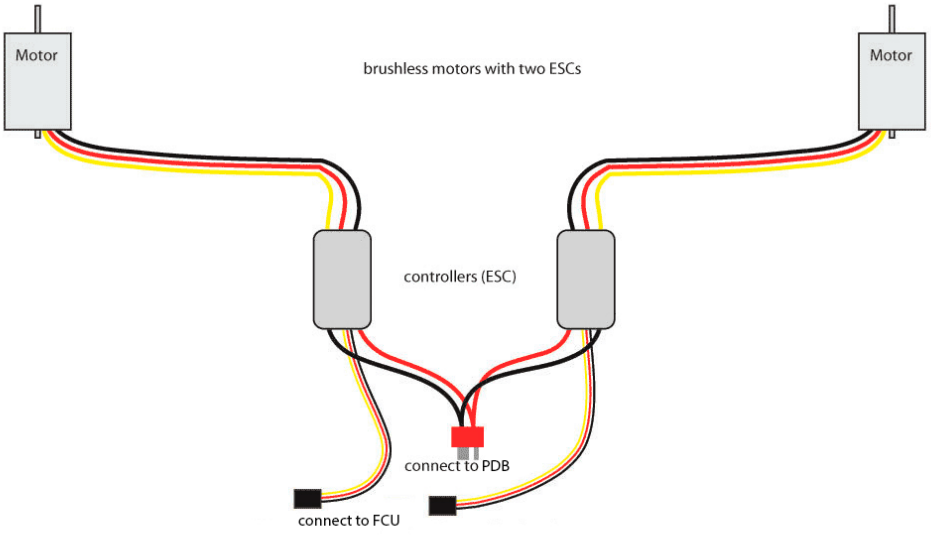


Fig.G.1 Brushless motors and Electronic Speed Controllers (ESCs) connection to flight management unit and the power distribution board (image modified from one in www.pteroworks.com)

The use of brushless motors requires the presence of Electronic Speed controllers (ESC). Electronic speed controllers are responsible for spinning the motors at the speed requested by the flight management unit (FMU). Each brushless motor is connected with three wires (fig.G.1) to the corresponding ESC. This is because brushless motors are three phase motors having three

coils inside. Consequently ESCs activate the coils in sequence and depending on the synchronization they can accelerate or decelerate the motors to the desired speed. The ESCS comprises of the following parts:

- Microcontrollers that activate or deactivate the coils and calculate timing by measuring the feedback in the coils caused by the movement of the magnets.
- 5V voltage regulators to produce a stable supply and a low battery cut-off feature to stop spinning the motor when the lithium polymer battery is drained. This feature protects the battery from suffering permanent damages that occur if it drained below a threshold.

A general schematic of a UAV wiring is shown in the (fig.G.2)

G.1.2 Vibration damping

The UAV's flight management unit (FMU) sits in the upper part of the central hub. It is equipped with accelerometers in the inertial measurement unit (IMU), which are sensitive to vibrations. In general, in most of the flight modes it is crucial to have a correct position estimate. The calculation of the position estimate is carried out using data from the IMU in combination with the barometer and GPS. It is clear that vibrations can interfere the operation of these units and lead to unacceptable performance. Generally speaking, the motors and propellers produce a high frequency and low amplitude vibration. The vibration needs to be less that 0.3 g in the X and Y axis and less that 0.5G in the Z axis (Copter.ardupilot.com, 2015).

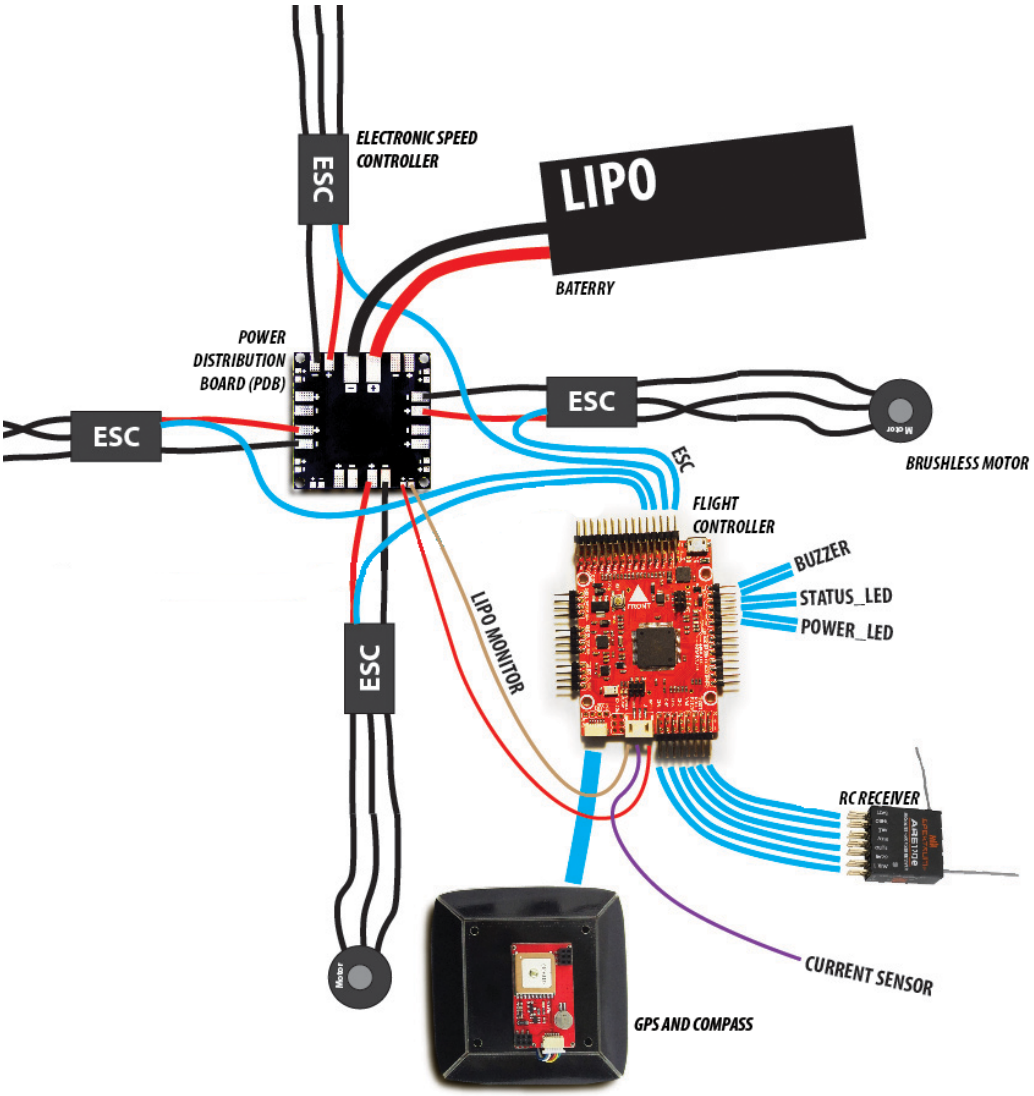


Fig.G.2 G general schematic of a UAV wiring (image modified from one in www.diymulticopter.org)

The most important factor that affects the vibration is the frame. Frame arms should be as rigid as possible. Carbon fiber armed copters and copters with injection-molded exoskeletons perform much better than cheaper frames, which tend to be flex. It is also very important to assemble secure and flex-free all the conjunctions like motors to frame arms, frame arms to central hub and accessories to frame. Another factor that plays a role in the vibration is the propellers. They should be fully balanced. The size and the slope are important. Large slow propellers will induce more vibration than small fast ones. In addition, the material from which they are made of affects the vibrations. Carbon fiber-made have the best performance.

Since the UAV has been assembled following the guides for minimizing the sources of vibration, a further optimization step can be implemented. More precisely, the flight controller mounting plate can be installed on the central hub using anti-vibration absorber rubber balls.

G.1.3 Magnetic interference

An additional factor that should be minimized since it degrades the performance and can cause serious problems in various flight modes is the magnetic interference. The DC currents of the UAV cause this interference. The places where DC current exists is in the power distribution board (PDB) and the wires between the ESC and PDB. To combat this effect the following techniques can be adapted:

- Keeping the length of DC current wires as short as possible.
- Placing the external GPS and compass module above the FMU and the PDB in the lowest part of the central hub to ensure the greatest distance between them.
- In cases where it is feasible, twisting the wires between the PDB, ESC and battery and using grounded shielding.
- Finally, using 4-in-1 PDB and ESCs instead of 4 separate ESCs and a PDB, so as to produce less interference since all the circuitry is enclosed in an aluminium case without any free wires running along (fig.G.3).

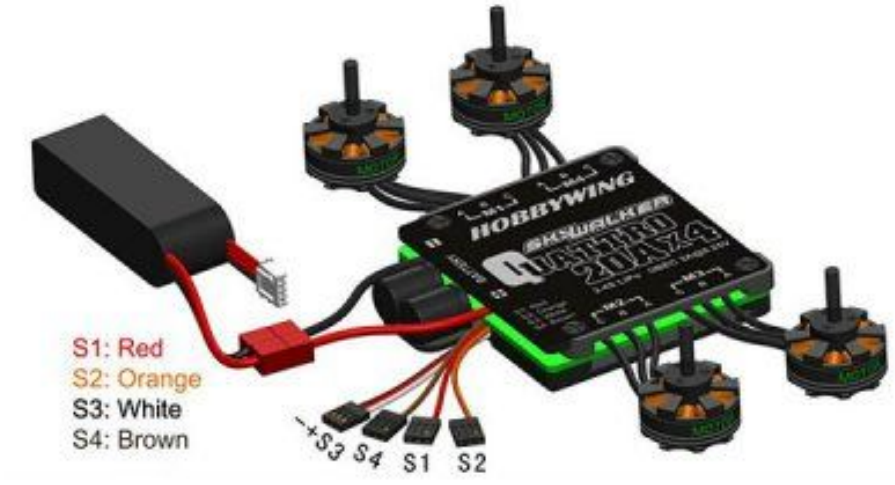


Fig.G.3 4-in 1 ESC and PDB module for less magnetic interference (image from copter.ardupilot.com)

G.1.4 Power module

The project's UAV is powered by Lithium – Polymer (LiPo) batteries. This type of batteries is the preferred power source for the most electric models. The advantages of employing them are the high discharge rates and the high-energy storage/weight ratio. However, they require special handling and charging. The most important matter to consider is the safety. Specific chargers should be used specifically designed to charge LiPo cells with the correct parameter set such as correct voltage and cell count.

In addition to the batteries, in order to provide filtered clean power to the FMU without glitches, a power module has been installed (fig.G.4). Through the 6P cable, it provides stable 5.3V voltage as well as current consumption and battery voltage measurements to the FMU. This module does not provide power to the ESCs, which are fed directly from the LiPo batteries through the power distribution board (PDB).

Fig.G.4 Flight control board power module T plug (image from www.littleblupigs.com)

G.2 Flight controller

The flight controller or flight management unit (FMU) is the nerve center of the UAV. This component is a microcontroller and gathers all the connections from the various accessories. It sends the appropriate commands to the ESCs and

receives data from the sensors, the GPS and the compass as well as the telemetry link. The flight controller provides a built-in inertial measurement unit (IMU) and a barometer. The IMU, in general, is a component that integrates accelerometers and gyroscopes. It receives data from both and using sensor fusion algorithms it can estimate the current position and attitude of the UAV. The main usage of the barometer is to inform an estimation of height for altitudes above 10m.

There are two dominant flight controllers in the market (fig.G.5), the AMP 2.6 and the Pixhawk. The APM 2.6 is a complete open source autopilot system. It is equipped with a microcontroller based on the Atmel's 8bit-ATMEGA2560 chip. It includes a 3-axis gyro, accelerometer (6-DOF MPU-6000) and a barometer (MS5611-01BA03). It is compatible up to the APM:Copter 3.2 version firmware. Generally speaking, it is the most popular and bestselling flight controller with a large supporting community and plenty of online tutorials ranging from simple use to code developing.

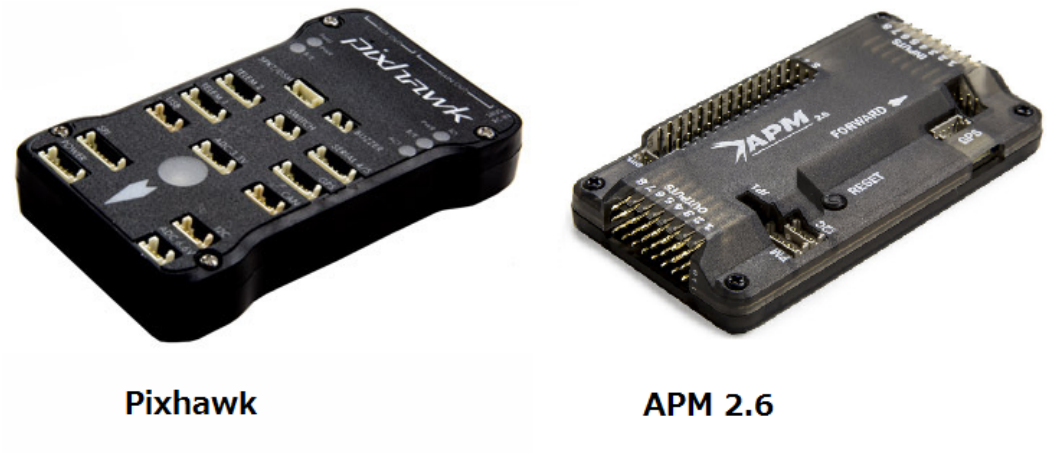


Fig.G.5 The most popular flight controllers: Pixhawk and APM 2.6 (image from copter.ardupilot.com)

On the other hand, the descendant of the APM is the Pixhawk board. The need of having a board that can offer more processing power and more available memory led the ETH's PX4 team to design it. Pixhawk is an advanced autopilot system designed by the PX4 open-hardware project. A 32-bit ARM Cortex M4 processor running NuttX real time operating system (RTOS) powers it. It

provides abundant connectivity options for additional peripherals (UART, I2C, CAN). It also has a MicroSD card slot for data logs.

Essentially, the difference between the two flight controllers is that the APM is old, well tested, and fully featured, but at the end of it's development cycle. The Pixhawk board is new, is more powerful, but it has not been as fully tested, and it has a lot of promise for the future. For the current project although the time frame of developing an application was very restricted, the need of flight controller that can support various advanced sensors and provide capabilities to implement obstacle avoidance techniques and ROS compatibility restrict the available choices. APM has been satisfying the need for a tried and tested configuration, with a large support and availability of many tutorials since there was no prior experience, yet the Pixhawk has been chosen in order to accommodate the project's enhanced requirements. The Pixhawk is more complex with a restricted knowledge base and a single point of support. It is a new project and quite immature compared to APM but it is the only tool if one wants to carry out professional projects.

G.3 Telemetry

Telemetry is a popular accessory added in UAVs that provides an air-to-ground data link between the autopilot and your ground station laptop or tablet. Although it is possible to control the UAV via the telemetry stream, it is safe to use an additional remote control (RC) for that, due to the superior reliability of that dedicated radio link and controller.

The telemetry equipment supports two transmission frequencies bands, the 433MHz and 915MHz radio links. The former is selected (fig.G.6) for the current project, mainly for the following reasons:

- The 433MHz is legal in Europe
- The 915MHz interference with the GSM bands, so in regions where mobile antennas exist (more in urban areas) it is more likely to have worse communication.

Fig.G.6 433MHz radio telemetry kit (image from www.rctimer.com)

The project's quad copter is equipped with an open source implementation of the Xbee replacement radio set. It is fully compatible with the 3DR radio system which is the market standard yet significantly cheaper. It has a range of approximately 1 mile using transmit power up to 20dBm. It uses the MAVLink communication protocol (Micro Air Vehicle), which is a very lightweight, header-only message marshaling library. It supports both frequency hopping spread spectrum (FHSS) and adaptive time division multiplexing (TDM).

G.4 GPS and compass module

This component includes a GPS and a compass (fig.G.7). The GPS component is the u-blox 6-position engine, a low power consumption high performance GPS adapted to the miniature form factor (NEO-6 series, 2011). It uses the NMEA protocol with baud rate equal to 9600 and 5Hz update rate. The other component is the HMC5883L compass. It is a popular magnetometer with 1-2 degree compass heading accuracy. It is considerably fast reaching 160Hz maximum output rate (3-Axis Digital Compass IC HMC5883L, 2010).

Fig.G.7 Neo 6M GPS and MAG v1.2 module (image from www.rctimer.com)

G.5 On-board computer

On-board or companion computers, are single board computers that add the real intelligence to the UAV or generally in a robotic system. Admittedly, the addition of on-board computer adds to the restricted payload of a UAV but there is no other way to develop a sense of autonomy to the vehicle. It is noteworthy that the flight controllers (APM or Pixhawk) are not capable of executing complex computer vision or SLAM (Simultaneous Localization and Mapping) algorithms.

From a technical point of view, the actual operation of companion computer is that it can be used to interface and communicate with the flight controller using the MAVLink protocol. Aside from the interconnection of the vehicle's subsystems, this protocol is also utilized for the communication between the ground control station and unmanned vehicle. In my case, the companion computer gets the operator's control commands and the sensors' readings, and use them to make intelligent decisions during a flight.

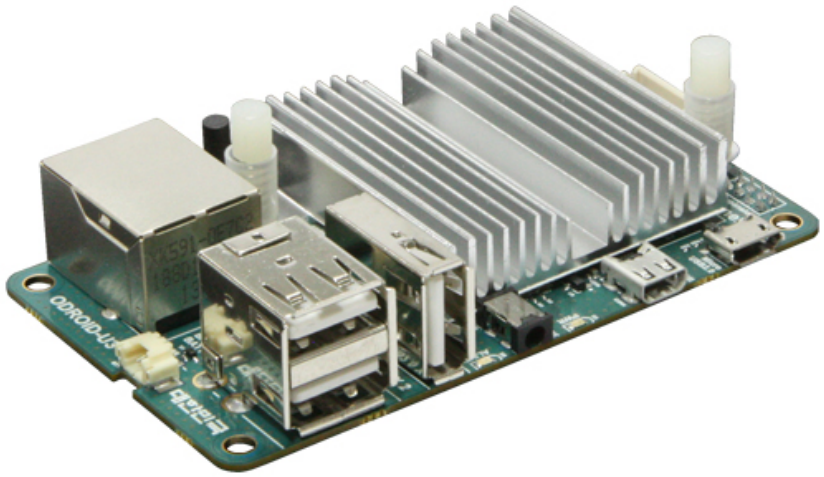


Fig.G.8 Odroid-U3 companion computer for robotics applications (image from www.hardkernel.com)

For the project's UAV, the Odroid U3 mini-computer was employed (fig.G.8). The selection of the specific model was based on the recommendation of the PX4 development community. More precisely, the requirement was to use a mini-computer having the computational power to support a slim Linux version running ROS. Odroid U3 is powered by a 1.7GHz Quad-Core processor and a 2GB RAM. It supports MicroSD and eMMC storage. The latter is 4 times faster than the Micro SD, which allows executing computer vision and robotics algorithms. It hosts 3 high-speed USB2.0 ports and 100Mbps LAN. Odroid U3 on-board computer has also a serial UART port in order to provide system console interface for platform development and debugging. All in all, it is a card-size (83 x 48 mm), light (48g including the heat sink) yet powerful minicomputer, which ideally complies with the robotics control requirements.

Appendix H

Real UAV Development

A description of the development of the real UAV applications is presented in this part. More precisely, for the real UAV experiments two approaches have been adapted. The technical details and the required configuration for both of them are illustrated throughout this part.

H.1 Basic Approach

In the context of the basic approach, a custom measurements device has been developed. This device is based on the circuit presented as the proximity application in the "Sensors" part. Consequently, an Arduino Uno controls the LIDAR-Lite range finder and the Hitec HS-422 servomotor. The circuit connections are illustrated in the following figure (fig.H.1) and in the table (tbl.H.1).

On top of the Arduino, the RedBear's lab Bluethooth Low Energy (BLE) shield was installed which is responsible for the communication with the client application. Finally, a breadboard shield sits on top of them, which facilitates the soldering of the custom circuit connections in order to deploy a compact structure at the end.

Table H.1

(Connection between LIDAR-Lite, HS-422 and Arduino Uno)

LIDAR-Lite	Arduino Uno
5V V_{cc}	5V
PWR EN	(Unused)
MODE	(Unused)
SCL	A5
SDA	A4
Ground	GND

HS-422	Arduino Uno
Control	A7
Ground	GND
Power supply	5V

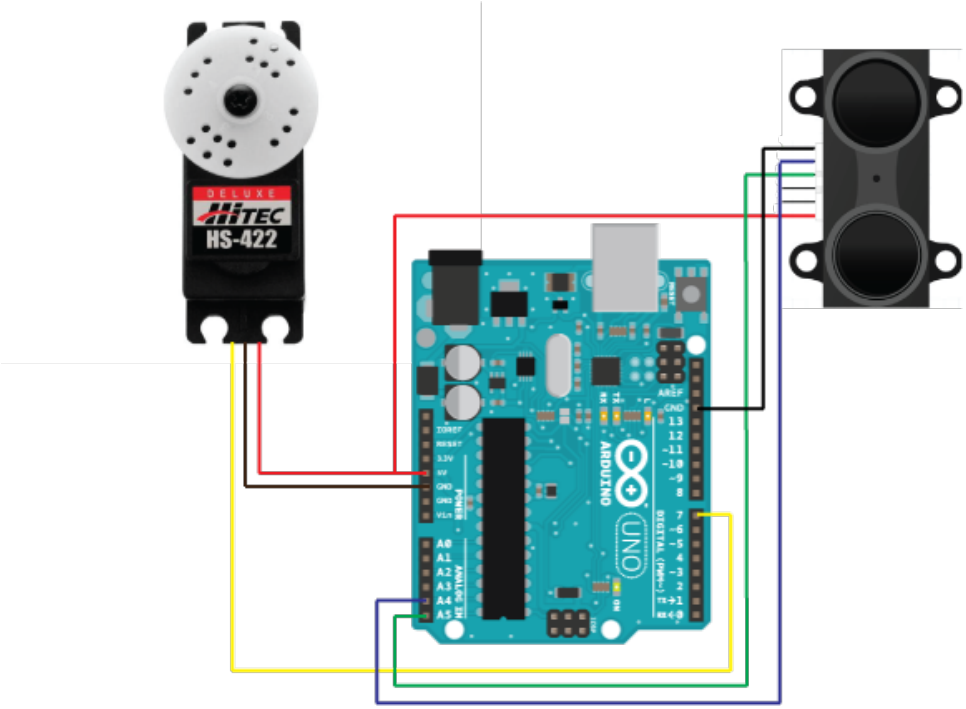


Fig.H.1 The basic circuit of the custom measurements device consist of the LIDAR-Lite, ant the servo HS-422 deluxe

The servomotor is configured to rotate and stop in the three main directions, 90°, 0° and -90°. In order to achieve higher precision, the servo was calibrated at the beginning by identifying the exact durations in microseconds that correspond to the predefined directions. After that, the servomotor was able to rotate continuously to these directions using the *writeMicrosecond* command. The delay, which was set between the rotations, was selected in order to have the shortest value without having problems with drifts and synchronisation.

The communication between the laser sensor and the microcontroller is implemented using the I2C protocol. The DSS circuit has developed the library, which was utilised for this connection, as recommended from the LIDAR-Lite manufacturer. In every predefined direction, the laser sensor performs two measurements and calculates the mean value. After that, using two union structures, the distance is converted to 4 bytes of a 32-bit integer and the

direction in 2 bytes of a 16-bit integer. These 6 bytes form the payload of the message that is sent in every predefined direction.
The related Arduino sketch is available on the online git repository in the following address (goo.gl/0S717x)

The second component of the basic approach is the client mobile application. The prototype application is visually structured for the layout of iPhone 5 (4 inches diagonal, 1136x440, 326ppi). The targeted version iOS version is 8. The implemented application consists of two *UIViewControllers* embedded in navigation controllers (fig.H.2).

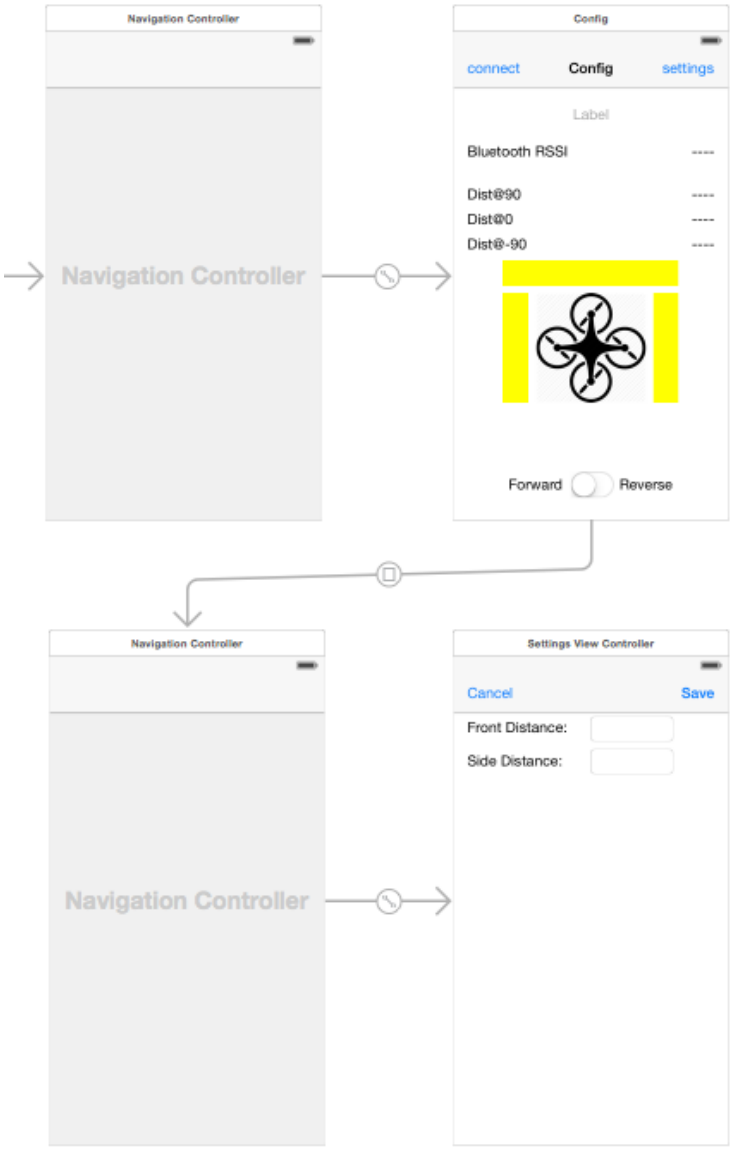


Fig.H.2 The storyboard of the client application

In the main screen, information related to the obstacle detection procedure is depicted. On the upper part of the screen, the distances in the three directions are shown. In order to make it easy for the operator to quickly identify the presence of an obstacle, a graphical representation of the surrounding space has been added. The input data in this *UIViewController* are delivered via Bluetooth. In order to add the Bluetooth functionality I utilised the BLE Library, which is supported by the BLE Shield that I have installed on the other side. By using the appropriate delegate methods, the client application can search for compatible Bluetooth devices and handle the connection and disconnection events. In addition, there is a method that delivers the message that is transmitted from the measurement device. This message consists of 6 bytes (fig.H.3). In this part, the opposite procedure is implemented and the 6 bytes are assembled to form the 32bit integer distance and the 16bit integer direction. Once the data are available, the screen of the *UIViewController* is updated.

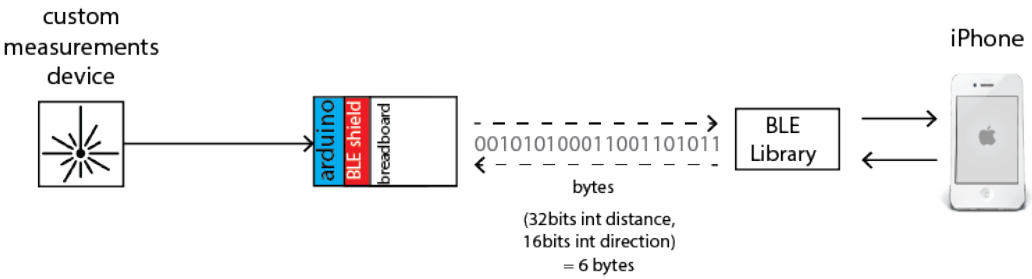


Fig.H.3 The schematic representation of the basic approach

The second *UIViewController* is used only for data entry. The operator can type the thresholds for the distances in front and on both sides. These values are used in the first screen in order to paint the related region with the appropriate colour (red: danger, green: no danger).

All in all, the navigation between the *UIViewControllers* was implemented using segues. The exchange of the data between them was accomplished using delegate methods. The source code for the application is available online on the following address (goo.gl/7rRSkP).

H.2 Advanced Approach

In the advanced approach the on-board computer acts as a central hub in the system. Around the on-board computer a number of entities interact by sending and receiving data (fig.H.4). Having described the functional overview of the system in the "Real UAV" part, a detail technical description regarding the configuration and the code development of every part will be presented.

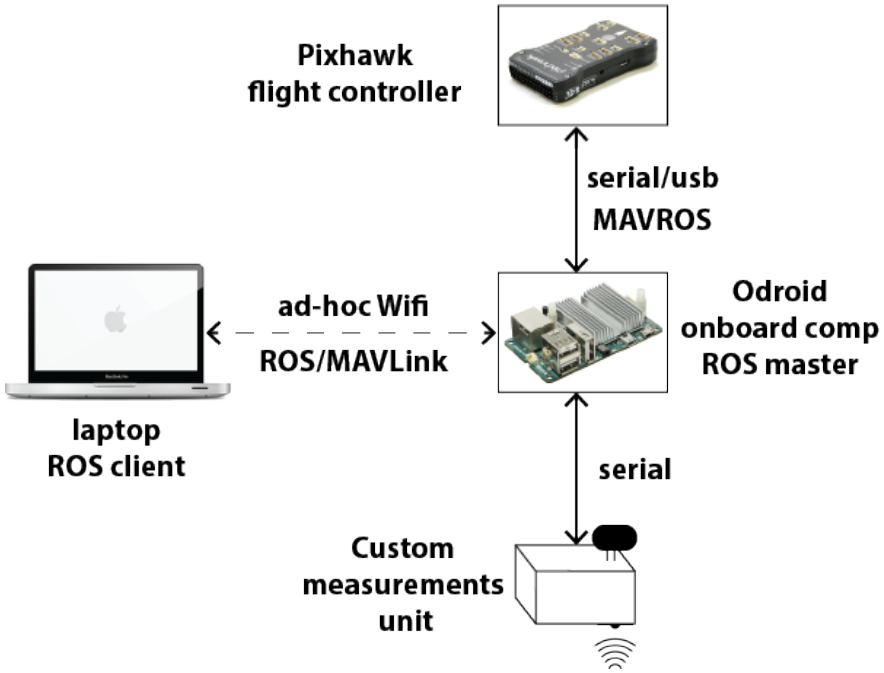


Fig.H.4 The schematic representation of the advanced approach

H.2.1 On-board computer

The on-board computer is an Odroid U3 community edition. The specific Odroid configuration that was used in the project has an eMMC storage card and a cooling fan as extras compared to the basic edition. In addition, in order to keep the system clock consistent a backup battery was installed. The Odroid as it is manufactured does not have an operating system installed. For the project's purposes there is a need for a lightweight yet recent and powerful operating system. The recommended choice is to install an Ubuntu 14.04 LTS server and particularly a manufacturer's version that has a very small size since it includes only the essential modules. Instructions of how to format and flash the Linux image are described in this online guide (goo.gl/bl9KCc). In order to have access

at the beginning in the Odroid, there is a need for a serial console cable. In addition, when the Linux image was flashed the disk partition was created having a size of 2Gb without taking the entire available space. It is recommended to use the Odroid utility and resize the partition to employ all the available space, for which instructions can be found in this guide (goo.gl/NVpmVU). One last task that needs to be performed is the set up of the network connections. The Odroid has a 100Mbps Ethernet port. Additionally, in order to establish an ad-hoc wireless network with ground computer a Wi-Fi module is also installed. It is convenient to have the Ethernet connection set as a DHCP client to use Internet during the setup and the Wi-Fi locked to peer-to-peer communication. The configuration of this scenario is implemented in the */etc/network/interfaces* file which is available in the online repository (goo.gl/2ads0x).

At this point the Odroid computer should be fully workable and the user should be able to login wirelessly by using a ssh command like:

ssh odroidip –l username (e.g. ssh 192.168.100.110 –l root)

The next step is to install ROS. In this guide (goo.gl/9Y74Ve), there are instructions on how to install a specific ARM build of ROS Indigo to Ubuntu 14.04 LTS. In order to save storage space and avoid unwanted complexity, only the necessary packages are installed such as ROS base, MAVLink and MAVROS.

There are three available ways to connect Odroid and Pixhawk:

- Using the USB port. This is the easiest way and it works. However, it is not recommended during flights since it is an unstable interface as it is cited in the manufacturer's site. In this this project, for simplicity reasons during the tests, it was used without problems.
- Using hardware UART. In Odroid, there is available a UART socket for system console which can be directly connected with the Telemetry port of the Pixhawk. It requires a cable with the appropriate terminals and a level shifter to convert to 3.3V. This way was not used in the project because this type of hardware was not available.
- Using UART Adapter. In this type of connection, a FTDI USB adapter is utilised to connect the 2nd telemetry port of the Pixhawk with the USB port of the Ordoid. This connection was built during the project and it worked as expected.

The obstacle avoidance application, which was developed in the simulation, needs to be modified in order to enable the communication with MAVROS and the custom measurement device. The necessary changes, which were accomplished, in order to convert the code in the compatible version that is available on git (goo.gl/awpnbG) are the following:

- Change the topic in which the joystick transmit the control commands from */mavros/setpoint_velocity/cmd_vel* to */obs_cmd_vel* in mavteleop script. This is required to enable the obstacle avoidance application to act as getaway (goo.gl/J1DRvs).
- Change the sensor topics from the ones that were used in the simulation to the real ones. Therefore, the active proximity sensor topics in the real UAV are:
    - */dist0*: The distance in front (heading: 0^o) measured using the laser sensor.
    - */dist90*: The distance on the left (heading: 90^o) measured using the laser sensor.
    - */distm90*: The distance on the right (heading: -90^o) measured using the laser sensor.
    - /sonar: The distance from the ground measured using the ultrasonic sensor.

  In addition to them, for the IMU unit the correct topic is the */mavros/imu/data_raw*.
- Change the topic in which the obstacle avoidance application sends the control commands to */mavros/setpoint_velocity/cmd_vel* instead of the */cmd_vel.* Additionally, the code should also change since the type of the message in the new topic is *TwistStamped* and not *Twist* as before.
- Simplify the mechanism which was used to obtain the front and the sides' measurements. In the simulation this was implemented by partitioning the 180^o range of the laser scanner to three domains. In the real UAV there are three different topics that transmit the related distances.
- Adjust the distance thresholds to safer levels for real UAV experiments.
- Adapt the sensors limits to the real ones. Additionally, adjust the laser specific behaviour for the case that returns 0 when it measures distances greater than its max range.

The application files have been bundled in a ROS package named *my_px4*. Inside the package there are the required files to execute the obstacle avoidance application. Similarly with the simulation, at the beginning the *realHover* node needs to be executed:

rosrun my_px4 realHover

And afterwards the application's executable file:

rosrun my_px4 ObstacleAvoidance3

However, it is desirable to increase gradually the complexity in order to fix possible bugs. To achieve this, simpler versions of the algorithm have been designed. These scripts can be applied to the system and explore basic behaviours such as:

rosrun my_package realHoverTest (to test the UAV's altitude hold mode)

rosrun my_px4 ObstacleAvoidance0 (to test the simple avoidance algorithm in which the UAV lands when it identifies an obstacle)

After the initial setup, the Odroid was installed on the UAV, between the custom measurements device and the central hub. The power input is connected to a Li-Po battery through an Ultimate Battery Eliminator Circuit (UBEC) to convert the battery voltage to 5V.

H.2.2 Custom Measurements Device

The custom measurement device is an enhanced version of the implementation that I used in the previous approach. More precisely, supplementary to the LIDAR-Lite laser sensor, a Devantech's SRF02 Ultrasonic sensor has been installed at the bottom of the plastic box. The overview of the system is illustrated in the figure (fig.H.5)

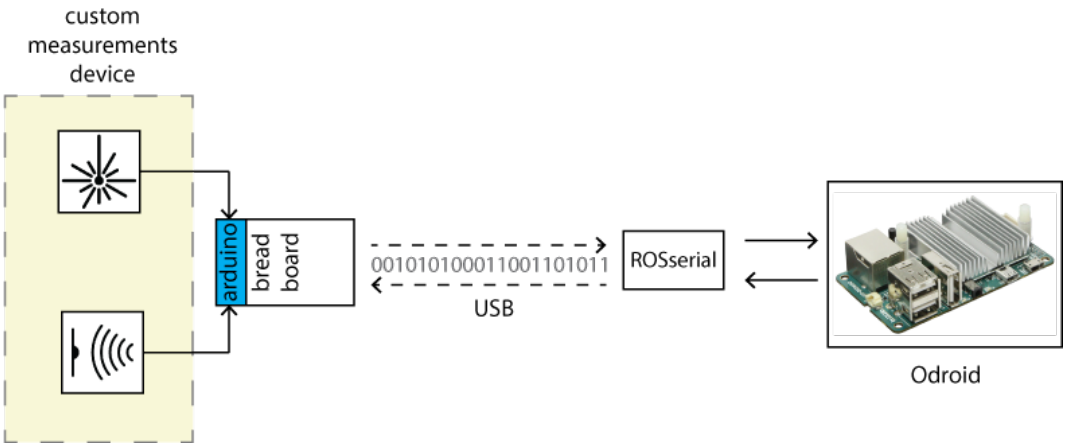


Fig.H.5 The overview of the custom measurements device

The circuit, as it is shown in the diagram (fig.H.6), consists of the LIDAR-Lite laser sensor, the SRF02 ultrasonic sensor, the HS-422 deluxe servomotor connected to an Arduino Uno microcontroller with a breadboard shield on top. The exact circuit connections can be found in the table (tbl.H.2).

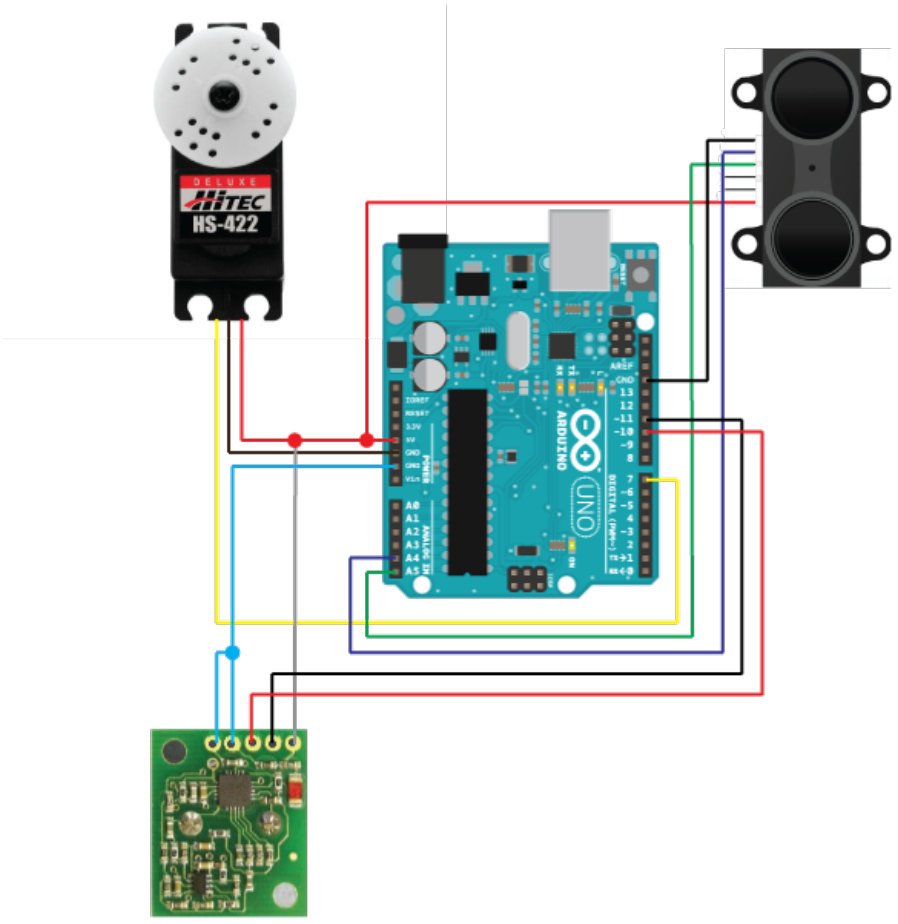


Fig.H.6 The implemented circuit of the custom measurements device

Table H.2

(Connection between LIDAR-Lite, HS-422, SRF02 and Arduino Uno)

LIDAR-Lite	**Arduino Uno**
5V V_{cc}	5V
PWR EN	(Unused)
MODE	(Unused)
SCL	A5
SDA	A4
Ground	GND
HS-422	**Arduino Uno**
Control	7 digital
Ground	GND
Power supply	5V
SRF02	**Arduino Uno**
5V V_{cc}	5V
Rx	11 digital (Tx)
Tx	10 digital (Rx)
Mode	GND
Ground	GND

The custom measurements device is connected with the Odroid via a USB cable. For the power supply of the device a LiPo battery is used. A UBEC 5V/5A is utilised in this case as well. The Arduino sketch (goo.gl/T9SsgZ) for the circuit is based on the previous version that have been already analysed. ROS library is included in order to provide the required connectivity using *rosserial* to transmit the data to the on-board computer. Finally, in order to enable the on-board computer to receive the data, which are transmitted contentiously in the appropriate topics, the following command should be executed:

roslaunch my_package iron2.launch

It is assumed that the Linux assigned port is the *(/dev/ttyACM1)* otherwise the *iron2.launch* file should be changed accordingly.

H.2.3 Flight Controller

The Pixhawk was used as the flight controller for the project's UAV. In order to be fully compatible with ROS and enable the communication with the on-board computer, the PX4 flight stack firmware was flashed in the controller. The firmware upgrade and the initial calibration were performed using the qGroundControl station software. During the calibration, the accelerometer, the

gyroscope and the ESC units were tuned. Additionally, a Taranis radio control system was calibrated in order to manipulate the UAV. An important task during the radio control calibration is to assign a switch using an individual radio channel for the off-board flying mode. This is the failsafe option of the system, since in the case that something goes wrong in the autonomous flight the operator is able to take the control back and resolve the emergency situation.

In the project's experiments, the Pixhawk was connected using a USB cable with the on-board computer. In order to arm the quad-copter and fly, the following steps have to be accomplished.

1. The MAVROS px4 communication should start:

roslaunch mavros px4.launch &

2. The joystick control should be enabled:

roslaunch mavros_extras teleop.launch &

3. The Pixhawk is read to arm (the safety switch should be pressed before)

rosrun mavros mavsafety arm &

The ampersand symbol at the end of every command enables us to execute them in the background in order to work from a single terminal. Additionally, it is assumed that the Linux assigned port is the *(/dev/ttyACM0)* otherwise the *px4.launch* file should be changed accordingly.

H.2.4 Ground computer

The ground computer has also Ubuntu Linux as the operating system. It is connected to the ad-hoc network of the on-board computer. Since both computers are in the same sub-network, they can operate using the ROS distributed topology or simply as master and slave. To enable this scheme, in the terminal, on the ground computer, the following commands should be executed:

export ROS_IP:ipOfGroundStation (e.g. export ROS_IP=192.168.100.9)

export ROS_MASTERURI=http://ipOfonboardComp:11311
(e.g. export ROS_MASTER_URI=http://192.168.100.110:11311)

And in the Odroid terminal:

export ROS_IP:ipOfonboardComp (e.g. export ROS_IP=192.168.100.110)

To control manually the UAV, a joystick is utilised. This device is connected via USB (*/dev/input/js0*) to the ground computer and sends the control commands to the UAV using the underlying infrastructure. To enable the joystick as an input device for the ROS-enabled UAV the following commands should be executed in the ground computer terminal:

sudo jstest /dev/input/js0
sudo chmod a+rw /dev/input/js0
rosparam set joy_node /dev/input/js0
rosrun joy joy_node

These commands have to be executed between steps 2 and 3 as they have been described in the flight controller section.

Appendix I

References

3-Axis Digital Compass IC HMC5883L. (2010). 1st ed. [ebook] Available at: http://www.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf [Accessed 9 Jul. 2015].

A.R. Al-Omari, M., A. Jaradat, M. and Jarrah, M. (2013). Integrated Simulation Platform for Indoor Quadrotor Applications. In: *Proceedings of the 9th International Symposium on Mechatronics and its Applications (ISMA13)*. Amman, Jordan.

Bohren, J. (2015). *ROS Crash-Course, Part I Introduction to ROS distribution, build system and infrastructure*.

Bristol.ac.uk, (2015). *Bristol University | School of Physics | About*. [online] Available at: http://www.bristol.ac.uk/physics/research/iac/about/ [Accessed 16 Jul. 2015].

Chen, M., Edwards, D., Boehmer, E., Eller, N., Slack, J., Speck, C., Brown, S., Williams, H., Wilson, S., Gillum, C., Lewin, G., Sherriff, M. and Garner, G. (2013). Designing a spatially aware and autonomous quadcopter. pp.213-218.

Conley, K. and Foote, T. (2014). *ROS community metrics*. [online] ROS. Available at: http://download.ros.org/downloads/metrics/metrics-report-2014-07.pdf [Accessed 30 Jul. 2015].

Copter.ardupilot.com, (2015). *Vibration Damping | Copter*. [online] Available at: http://copter.ardupilot.com/wiki/initial-setup/assembly-instructions/vibration-damping/ [Accessed 8 Jul. 2015].

Crick, C., Jay, G., Osentoski, S., Pitzer, B. and Jenkins, O. (2011). Rosbridge: ROS for non-ros users. In: *International Symposium on Robotics Research (ISRR 2011)*

Cyberbotics.com, (2015). *Webots: robot simulator - Overview*. [online] Available at:

https://www.cyberbotics.com/overview [Accessed 30 Jul. 2015].

Developer.apple.com, (2015). *UIWebView Class Reference*. [online] Available at: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebView_Class/ [Accessed 10 Aug. 2015].

Fischler, M. and Bolles, R. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), pp.381-395.

Goebel, R. (2014). *ROS by example*. [Raleigh, NC]: Lulu.com.

Goodhoofd, W. (2014). *wesgood/RBManager*. [online] GitHub. Available at: https://github.com/wesgood/RBManager [Accessed 10 Aug. 2015].

GP2Y0A02YK0F. (2006). [online] Sharp. Available at: https://www.sparkfun.com/datasheets/Sensors/Infrared/gp2y0a02yk_e.pdf [Accessed 18 Jul. 2015].

Hartman, D., Landis, K., Mehrer, M., Moreno, S. and Kim, J. (2014). *dch33/Quad-Sim*. [online] GitHub. Available at: https://github.com/dch33/Quad-Sim [Accessed 18 Jun. 2015].

Honegger, D., Meier, L., Tanskanen, P. and Pollefeys, M. (2013). An open source and open hardware embedded metric optical flow CMOS camera for indoor and outdoor applications. pp.1736-1741.

Hrabar, S. (2011). Reactive obstacle avoidance for Rotorcraft UAVs. pp.4967-4974.

Imitec Limited, (2014). *Advanced Airborne Radiation Monitoring System*.

Imitec, (2014). *Sellafiend Trial*. [video] Available at: https://vimeo.com/97758715 [Accessed 16 Jul. 2015].

Lewis, M. (2015). *Introducing SocketRocket: A WebSocket library for Objective-C*. [online] Corner.squareup.com. Available at: https://corner.squareup.com/2012/02/socketrocket-websockets.html [Accessed 10 Aug. 2015].

Martinez, A. (2013). *Learning ROS for Robotics Programming*. Packt Publishing.

Mellander, H. (1995). The role of mobile gamma spectrometry in the Swedish emergency response programme for nuclear accidents,Äì experience and future plans. In: *Application of Uranium Exploration Data and Techniques in Environmental Studies,*.

Mendes, J. and Ventura, R. (2012). Safe teleoperation of a quadrotor using FastSLAM. pp.1-6.

MEXT, (2015). *Results of the Fourth Airborne Monitoring Survey by Ministry of Education, Culture, Sports & Technology (MEXT)*. [online] Available at: http://radioactivity.nsr.go.jp/en/contents/4000/3179/24/1270_1216.pdf [Accessed 13 Jul. 2015].

Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U. and Stryk, O. (2012). Comprehensive Simulation of Quadrotor UAVs using ROS and Gazebo. p.

NEO-6 series. (2011). 1st ed. [ebook] p.Versatile u-blox 6 GPS models. Available at: https://www.u-blox.com/images/downloads/Product_Docs/NEO-6_ProductSummary_(GPS.G6-HW-09003).pdf [Accessed 9 Jul. 2015].

Pitzer, B. (2011). *mjpeg server*. 1st ed. [ebook] Bosch Research Std. Available at: http://www.ros.org/wiki/mjpeg server [Accessed 10 Aug. 2015].

Pixhawk.org, (2015). *PX4 ROS SITL Setup - PX4 Autopilot Project*. [online] Available at: https://pixhawk.org/dev/ros/sitl [Accessed 31 Jul. 2015].

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. (2009). ROS: an open-source Robot Operating System. p.5.

Richards, A. (2015). *State Space Modelling and Simulation*. 1st ed. Robotics Systems notes.

Robot-electronics.co.uk, (2015). *Ultrasonic Rangers SRF04*. [online] Available at: http://www.robot-electronics.co.uk/htm/sonar_faq.htm [Accessed 17 Jul. 2015].

Ros.org, (2015). *ROS.org | About ROS*. [online] Available at: http://www.ros.org/about-ros/ [Accessed 17 Jun. 2015].

Sanderson, D., Allyson, J., Tyler, A. and Scott, E. (1995). Environmental applications of

airborne gamma spectrometry.

Schwarz, G., Rybach, L. and Klingele, E. (1995). Data processing and mapping in airborne radioactivity surveys. In: *Application of uranium exploration data and techniques in environmental studies, IAEA-TECDOC-827, IAEA, 61-70*.

Scott, T. (2014). *UAV radiation mapping survey of a Japanese school*. [video] Available at: https://vimeo.com/95921788 [Accessed 16 Jul. 2015].

Technology Brief. (2015). LIDAR-Lite Documentation. [online] PulsedLight. Available at: http://kb.pulsedlight3d.com/support/solutions/articles/5000552320-lidar-lite-technology-brief [Accessed 19 Jun. 2015].

Truchsess, W. (2015). *Arduino I2C Master Library*. [online] Dsscircuits.com. Available at: http://www.dsscircuits.com/index.php/articles/66-arduino-i2c-master-library [Accessed 20 Jul. 2015].

Uk.mathworks.com, (2015). *Robotics System Toolbox Documentation*. [online] Available at: http://uk.mathworks.com/help/robotics/index.html?refresh=true [Accessed 18 Jun. 2015].

Wang, S., Zhen, Z., Zheng, F. and Wang, X. (2014). Design of autonomous flight control system for small-scale UAV. pp.1885-1888.