Vision based navigation in a dynamic environment

Marcus Futterlieb

Université Fédérale

Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :
Marcus Futterlieb

le lundi 10 juillet 2017

Titre :

Vision Based Navigation in a Dynamic Environment

École doctorale et discipline ou spécialité :
EDSYS : Robotique 4200046

Unité de recherche :
CNRS - LAAS (UPR 8001)

Directeur/trice(s) de Thèse :

Viviane Cadenat et Thierry Sentenac

Jury :
Prof. El Mustapha Mouaddib, Université de Picardie Jules Verne
Prof. René Zapata, Université de Montpellier
Dr. Michel Devy, DR, CNRS - LAAS
Dr. Adrien Durand Petiteville, CR Université Davis
Dr. Viviane Cadenat, Maître de Conférences à l'Université P. Sabatier, HDR
Dr. Thierry Sentenac, Maître assistant à l'IMT Mines Albi, HDR

Contents

List of Figures

List of Algorithms

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
CPU	**C**entral **P**rocessing **U**nit
d	Dimensions/dimensional
DARPA	**D**efense **A**dvanced **R**esearch **P**rojects **A**gency
DWA	**D**ynamic **W**indow **A**pproach - A planner able to deal with obstacles locally
DOF	**D**egree **o**f **F**reedom
FOV	**F**ield **O**f **V**iew
Gazebo	Simulator general suggested for working on ROS
GPU	**G**raphic **P**rocessing **U**nit
IBVS	**I**mage-**b**ased **V**isual **S**ervoing
ICP	**I**terative **C**losest **P**oint
IMU	**I**nertial **M**easurement **U**nit
Laser	**L**ight **a**mplification by **S**timulated **E**mission of **R**adiation
LRF	**L**aser **R**ange **F**inder
NDT	**N**one-**D**estructive **T**esting
OA	**O**bstacle **A**voidance
PBVS	**P**osition-**b**ased **V**isual **S**ervoing
PTU	**P**an-**T**ilt-**U**nit
PTZ	**P**an-**T**ilt-**Z**oom camera
RANSAC	**RAN**dom **S**ample **C**onsensus
RANSAM	**RAN**dom **S**ample **M**atching
ROS	**R**obot **O**perating **S**ystem - Software allowing a simple integration of hardware and some off the shelf functionality for robots - For more information seeking Section 2.2.2 on Page 37

RPM	**R**evolutions **P**er **M**inute
R&R	**R**est and **R**ecuperation
SimIAm	MATLAB based mobile robot simulator useful for prototyping new controllers - Developed by the Georgia Robotics and Intelligent Systems (GRITS) Lab (see [de la Croix, 2013]
SLAM	**S**imultanious **L**oclaization **A**nd **M**apping
SNR	**S**ignal to **N**oise **R**atio
SVO	**S**emi-Direct **V**isual **O**dometry - A visual odometry proposed by members of the Robotics and Perception Group, University of Zurich, Switzerland
TEB	**T**imed **E**lastic **B**and - In order to avoid high level re-planning this approach allows for local path deformation in order to avoid obstacles
ViSP	Software package loosly based on OpenCV that allows a relatively simple and rapid visual servoing implementation - Developed by INRIA
VS	**V**isual **S**ervoing

1. Introduction

Contents

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.

2. A robot must obey the orders given it by human beings, except where such orders would conflict with the First Law.

3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

[Asimov, 1950]

1.1. Overview of general robotics context : my point of view of robotics in the 21st century

1.1.1. Motivations

In recent years, automation and the amount of robots in operation has exploded. With the contemporary advancements in autonomous driving, surgical or surveillance robots [Marquez-Gamez, 2012] we can safely assume that the amount of robots that will find their way from the research labs to our everyday life is increasing, whether we are afraid of this development or we embrace it. These robots mainly belong to the domain of mobile robotics which encompasses a vast range of terrestrial vehicles, underwater and aerial robots (**U**nmanned **A**erial **V**ehicles, UAV's) [Marquez-Gamez, 2012].

From time to time they still remind us of kids that need our help to understand their surroundings but with each passing year their algorithms become more and more mature. Just one look at the DARPA challenge for robotics shows us how far technology has come in recent years. Thinking about the Urban Challenge [Eledath et al., 2007], [Balch et al., 2007], [McBride, 2007] and looking at the maturity of self driving cars in 2017, this becomes even more obvious. It is very unlikely that this trend is going to change. If anything, the speed at which new technologies will be introduced will just increase.

As humans we are very versatile and quickly learn how even the most complex tasks can be performed. However, it still happens that we make mistakes in particularly stressful situations. These situations are encountered in aviation domain where a lot of studies on "human risk factors" have been conducted [Degani and Wiener, 1991], [Latorella and Prabhu, 2000], [Maintenance,], [Drury, 2001] and [Shappell and Wiegmann, 2012]. The risk can be reduced by designing "fool-proof" checklists, increasing the amount of R&R (Rest and Recuperation) of the staff or in some cases even double checking by a separate inspector [Rasmussen and Vicente, 1989], [Latorella and Prabhu, 2000]. A new trend is to help or to assist the staff in their daily tasks by robots. Taking a look at popular Utopian (and sometimes even Dystopia) fiction [Asimov, 1950], [Čapek, 2004], partly even [Lem, 1973] and [Dick, 1982], humans work force is completely replaced by machines. These developments then present a huge challenge to society and are meant as a warning to question technological progress. It is here, where humanity's challenge will lie in the future. How we will deal with the ever declining amount of blue collar and (more and more so) white collar jobs [Grey@youtube.com, 2014].

Another trade that we share as a species is that believe that technological progress advances in a linear fashion. If asked what kind of technologies will be introduced in the next five to ten years, we will take a look back at where we were five to ten years ago and make a linear extrapolation of what there is to come. This perception of linear progress however is flawed, because every new invention or discovery allows us to advance faster. Hence the technological change is exponential [Kurzweil, 2004], [Kurzweil,

2005], [Kurzweil, 2016]. This fact alone shows that the future is exciting, however, we should not follow technological advance blindly. Human history is full of examples where an invention's potential for bad was underestimated until it was too late. This of course does not mean that certain progress in technology can be stopped, however, it might be worth the time to consider (and anticipate) the consequence of our work. For instance, the mere prospect of a world without the need for manual labor, dangerous or dull working conditions does not make it a world worth living in. We will have to shape it so that all of human kind can profit from it, because if we continue like we have only very few people will get the luxury of living their life in a truly Utopian world, while the rest will struggle to survive. To go further into detail the next section will give a short ethical analysis about this statement.

1.1.2. Ethical analysis

In the previously mentioned science fiction novels, the narrator never fails to show us the dangers that comes with these improvements to technology. So, as we are taking further steps on a path that was so beautifully lain down in front of us, we should always keep in mind that we are not flawless and therefore, step ahead with care.

In this short paragraph, I would like to reflect on the ramifications and consequences of the replacement of human workers by machines. We want to focus this ethical analysis on the consequence of this on the employment market, since for most adults, a rather great time is spent working.

People have always been scared of new technology that might reduce the amount of available jobs (invention of printing, steam machine, etc.), but history has shown us that in fact, jobs that were destroyed have been compensated for (in some cases even exceeded) by sparking an era of growth due to the technological advance. Therefore, this mistrust and resistance towards new technology is not a recent development. However, in the past few years a discussion among scientists has started about how the robot revolution is fundamentally different to all the previous changes mankind has seen so far (Gutenberg printing press, steam machine, industrial revolution, etc.) and which in the long run have created more jobs than were destroyed [Frey and Osborne, 2013], [Brynjolfsson and McAfee, 2014], [Grey@youtube.com, 2014], [Ford, 2015]. If before the concern was mostly focused on the disappearance of low-skilled jobs and could be countered by the creation of new areas of employment, our society will be faced with robots competing for high-skilled jobs as well.

Figure 1.1 shows an extract form [Frey and Osborne, 2013] in which the likelihood of automation of specific jobs in the upcoming two decades is given. More and more scientists have come to realize that one of the hardest challenges for human kind might not be connected to satisfying basic needs, such as shelter, hunger etc. (although we are far from eradicating those needs at the moment), but might actually be associated with providing proper reason and purpose to our existence.

Bring on the personal trainers

Probability that computerisation will lead to job losses within the next two decades, 2013 (1=certain)

Job	Probability
Recreational therapists	0.003
Dentists	0.004
Athletic trainers	0.007
Clergy	0.008
Chemical engineers	0.02
Editors	0.06
Firefighters	0.17
Actors	0.37
Health technologists	0.40
Economists	0.43
Commercial pilots	0.55
Machinists	0.65
Word processors and typists	0.81
Real estate sales agents	0.86
Technical writers	0.89
Retail salespersons	0.92
Accountants and auditors	0.94
Telemarketers	0.99

Source: "The Future of Employment: How Susceptible are Jobs to Computerisation?" by C.Frey and M.Osborne (2013)

Figure 1.1.: Extract of article [Frey and Osborne, 2013]; Likelihood of computerization of several jobs

A more advanced illustration of this concept "robots replace human" was shown on the example of general practitioners and the cognitive computer system "Watson" by IBM in the popular video "Humans Need Not Apply" of CGP Grey [Grey@youtube.com, 2014]. "Watson" can be considered as an **A**rtificial **I**ntelligence, AI, developed by **I**nternational **B**usiness **M**achines Corporation, IBM, in DeepQA project. The software is capable of understanding and answering questions that are posed in natural language (as opposed to a well structured data set), which is the form most information exist today. For the purpose of demonstration "Watson" was developed to challenge human participants in the popular quiz show Jeopardy!. By having access to four terabytes of structured and unstructured information (including the full text of Wikipedia) Watson was able to defeat former winners Brad Rutter and Ken Jennings in 2011 [Wikipedia, 2015]. In recent years Watson was also tested in medical field of oncology [Malin, 2013], [Strickland and Guy, 2013]. Watson was able to show its potential in this field because of its fast pattern recognition among extremely big data sets that can not be processed by any human physician. And although the AI is still far from perfect, this example shows that it is not only the blue colored jobs that might be replaced (or reduced) by machines. In fact, Watson does not need to be perfect, it just needs to be better than its human counterpart. [Ferrucci et al., 2013] is referring to reports that diagnosis error

outnumber other medical errors by far. The main causes for those is thought to be the fast growing amount of new medication and research that no physician can keep up with during the working week. An AI with an advanced pattern recognition such as Watson completes this work much quicker. Furthermore, any miss-diagnosis by the machine will improve the next diagnosis. This might also be the case for the human counterpart, however, here it will only be the patients of that particular physician that will profit from his/her learning expertise. Patients of another doctor might suffer the same miss-diagnosis [Grey@youtube.com, 2014]. Studies have shown that certain physicians would need to read up to 627 hours [Alper et al., 2004] per month (consisting of papers and information about the latest medication, side effects and other research results) in order to keep up with the ever growing amount of available information about medication. It is unlikely that this number will decrease in the future. In fact, it is far more reasonable to assume that the amount of available medication, and therefore, the amount of possible side effects will only increase.

This leads us to the main problem. There are certain areas (automation, pattern recognition, etc.) in which a human will most likely not be able to beat its computer counterpart in the upcoming decades. And if the machine actually does provide better results, who would stop patients of seeking out its advice? And what will we do with the millions of people that will fall into unemployment because of automation? Who will provide for their financial security? This is not a problem that lies in the far future. According to [Bernstein and Raman, 2015], Brynjolfsson states that since the 1980s, the median income has begun to shrink. In Figure 1.2, we can see this trend that shows a nearly constant increase in labor productivity which had been followed by an equivalent increase of median family income until these trends began to separate in the 80s. Brynjolfsson fears that due to more and more automation this problem will get worse. Even more dangerous is the fact that the private employment seems to have stagnated in the 2000s although even though the economy has not. These two developments are what Brynjolfsson and McAfee refer to as the "Great Decoupling". He further states that a decrease of income equality has been registered throughout developed countries world-wide in the past 30 Years.

I personally believe that there is no simple answer to this problem. Keeping Moore's law [Moore, 1965] in mind even a very conservative person will have to accept that we will face more and more automation in the years to come. Of course we could try to prevent the rapid increase of automation. However, looking at technological advances throughout history, it will be hard (or even impossible) to find a case where this approach ever worked.

As we have stated before, throughout history, parts of the human population have always fought against technological advancement. Mainly, because they saw that their jobs were threatened. In retro-spec we can see that their jobs really were eradicated, however, due to the new technology (which they formerly had fought against), a completely new and at most times better (health risks, payment, etc.) line of work was

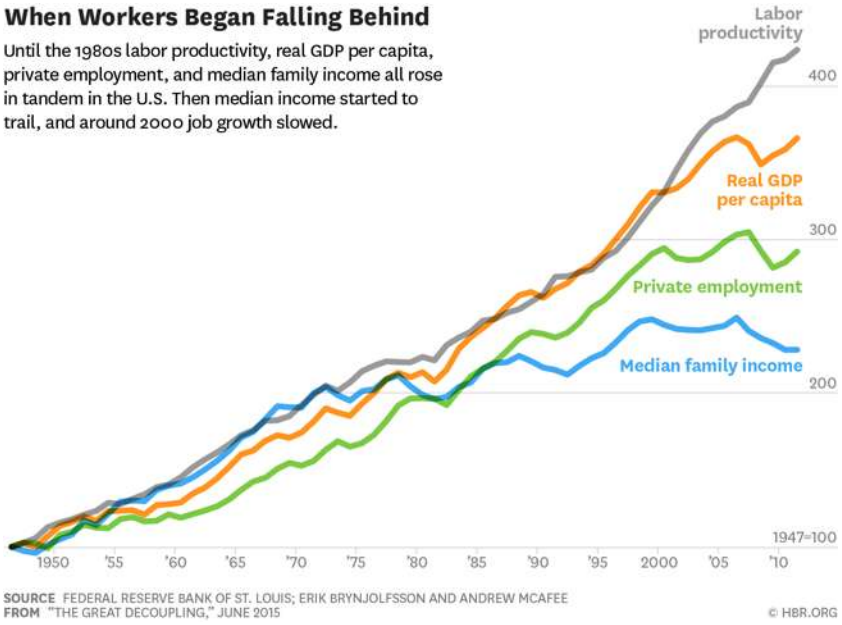


Figure 1.2.: Extract of [Bernstein and Raman, 2015]; The Great Decoupling explained

created. It is a fact that this opinion was the shared consensus among scientist for many years. Basically assuming that the problem would solve itself.

However, even in the research community more and more scientist claim that the amount of jobs being replaced by machines and software can not match the amount of new jobs that are being created at the same time. In fact, since one of the main drives for innovation in human society has always been optimization and increasing the effectiveness of processes in order to increase the profit (industrialization, green revolution, robotics) it is actually not surprising that we are succeeding in eradicating the need for human workforce more and more. In the end, was that not the idea in the first place? We forgot however, that the current system we live in is not a welfare system in which only a few people have to work. On the contrary, some of the most industrialized, first world countries become more and more anti-social. Recent polls show that more half of the American (USA) working population is in fact struggling to earn a living. [Online, 2016] states that the median US American worker received less than 30k $ net payment in the year 2014. This while many of these people are actually working more than one job. The wealth gap is increasing and more automation might just accelerate this trend.

If the system is not changed, those people will be the first ones without a job, without social security, without health care and retirement money. We should be careful not to dismiss these concerns and just hope that somehow enough paying jobs will survive the automation revolution. There is however and upside to this. If we look around in our communities, it is hard to get the impression that there is not enough work for everyone.

Sure, there might not be enough paid work, but there is enough to do. Additionally, of those people that do have a paid job, it is hard to find someone that is not forced to work more and more. The effect of such an employment is often that the employee is becoming unsatisfied that thinks beside work are not taken care of as much as he / she would like. Personally, I believe that this is exactly the upside of the problem. There is enough work, we just need to find a way to change the system we are living in to support people in doing it.

1.2. Air-Cobot project

Air-Cobot project is not an example of extreme automation of human tasks, as the "cobot" is expected to help the staff in hard working conditions rather than to replace it. It takes place in the aeronautical context to improve the safety of passengers and optimize the maintenance of aircraft.

There are nearly 100,000 commercial flights per day, which means that one thousand of aircraft inspections per hours occur across the globe. However, 70% of the inspections are still conducted visually by co-pilots and maintenance operators, which increase aircraft down-time and maintenance costs. In addition, these inspections are performed whatever the weather (rain, snow, fog, ...) and the light conditions (day, night).

The Air-Cobot is therefore a collaborative robot that automates visual inspection procedures. It helps co-pilots and maintenance operators to perform their duties faster, more reliably with repeatable precision. The robotic platform can provide a great and valuable help when the inspections conditions become tricky. Our main objective in this project is then to achieve a collaborative pre-flight inspection of an aircraft between the co-pilot or the operator and a robot. This latter must be able to navigate alone around the aircraft between the twenty different check points, while preserving the safety of the workers which are around the aircraft and the cars or trucks which are on the refueling area.

During the pre-flight inspection, the robot receives a maintenance planning from a human operator monitoring which aircraft is ready to be inspected. With this plan, the robot starts the navigation from its charging station towards the inspection area. During the approach to the test side, the robot moves along dedicated roads which are free obstacles corridors. Whenever the inspection area is reached, the aircraft can be detected and the robot moves relatively to it with respect to the maintenance planning. While performing the inspection task, the robot constantly sends progress reports to the flight personal and a trained operator. This latter will then be able to help out in case the robot encounters obstacles that block a checkpoint, malfunction in its sensors or similar unpredictable problems. When the Air-Cobot has finished its task to the operators satisfaction, it moves on to the next mission or navigates its way back to the charging station. Figure 1.3 shows the robot during an inspection task conducted in the

Figure 1.3.: Air-Cobot platform near the aircraft

early 2015 at the Airbus site Toulouse-Blagnac with an Airbus A320.

As shown in Figure 1.3, the robot is a 4MOB platform designed by STERELA. It consists of four independent wheels which are controlled in velocity. Additionally, it is outfitted with numerous navigation sensors : an IMU and a GPS; 2 stereo camera systems mounted on a pan tilt unit (PTU); a pan tilt zoom camera (PTZ); a laser range finder (LRF). More details about the robot and its sensors are provided in the upcoming chapter (see Section 2.2 on Page 36).

The robot is then endowed with all the necessary sensors allowing to implement the navigation modes. Indeed, three main navigation modes / phases can be highlighted :

- " an autonomous approach phase ", where the robot goes from the charging station to the aircraft parking area following a dedicated shipping lane. Thus, to fulfill its mission, the robot has to perform a large displacement along the road, as the initial and final points might be far. The airport environment is a highly constrained environment where the displacements obey precise rules. Thus a priori information will have to be given to the robot in order to avoid any forbidden path. As a conclusion, the approach phase from the charging station to the aircraft parking area needs an absolute navigation following a previously learnt path.

- “ an autonomous inspection phase ”, where the robot moves around the aircraft throughout a cluttered environment. The approach needs a large displacement with a highly efficient obstacle avoidance to respect strong security requirements. The navigation is mainly based on a relative navigation to the aircraft while taking into account the obstacles (aircraft, vehicles, maintenance or flight crew, etc.).
- “ a collaborative inspection phase ”, where the robot follows the co-pilot or a maintenance operator. Here, the autonomy of the robot is reduced, as the motion is defined by the person in charge of the pre-flight inspection.

The two first above mentioned navigation phases are at the core of my research work.

1.3. Aims and contributions of this thesis

This PhD work focuses on multi-sensors based navigation in a highly dynamic and cluttered environment. To answer this problem, we have tackled the following issues :

- long range navigation far and close to the aircraft,
- obstacle avoidance,
- appearance and disappearance of the aircraft in the image,
- integration on a robotic platform.

To deal with the above mentioned problems, a navigation strategy needed to be developed. This strategy makes use of the sensors (GPS/vision/laser) which are the most appropriate depending on the phase and on the situation the robot is. In order to be able to navigate far and close to the aircraft, it is necessary that our strategy features local and global properties. The navigation problem is analyzed first, from a general point of view, highlighting the different processes which are at its core: perception, modeling, planning, localization, action and decision. Our first contribution consists in the instantiation of these processes to obtain the desired global and local properties. Here is a brief overview of the instantiated processes before a more detailed explanation will follow in the subsequent chapters. Perception is both based on “exteroceptive” and “proprioceptive” sensors. Detection and tracking algorithms have also been developed to control the robot with respect to the aircraft and to avoid static and dynamic obstacles. The modeling process relies on a topological map (the sequence of checkpoints to be reached) and two metric maps. Here the first one is centered on the robot last reset point and the second one on the aircraft to be inspected. A data fusion technique is also suggested for the relative localization of the robot versus the aircraft. The action process is made of several controllers allowing to safely navigate through the airport. The decision process allows to monitor the execution of the mission and to select the best controller depending on the context.

Our second contribution is related to the sensor-based control of the robot around the aircraft. Indeed, as Air-Cobot is dedicated to inspection tasks, its positioning on the checkpoint must be accurate. To achieve this goal, visual servoing controllers are used in conjunction with metric based approaches. However, in order to achieve the high accuracy demand, the visual servoing controller will always be used as the primary means to reach the target. We will then compare different solutions to determine which of the different visual servoing controllers is the most optimal adapted to the current situation. In addition, we have also taken into account the problem of visual signal losses which might occur during the mission. We then propose a combination of both Image-based and Position-based visual servoing in a framework that: (i) reconstructs the features if they disappear with the help of a virtual image which is updated online with several sensors used simultaneously ; (ii) is able to find back these features ; and (iii) even navigate using only their reconstructed values. Finally, it is important to note that the Air-Cobot platform is equipped with five cameras, leading to a multi-visual servoing control law.

Our third contribution is related to obstacle avoidance. It is performed with the help of a laser based controller which is activated as soon as dangerous objects are detected. This reactive controller allows to avoid both static and dynamic obstacles. It relies on the definition of an avoidance spiral, which can be modified online depending on the needs (newly detected obstacles, path free anew, mobile objects motion, etc.). We will show that this approach (which is based on the flight path of insects) is extremely reactive and can be well adapted to the task at hand.

Finally, all these contributions are integrated on a robotic prototype that was designed solely for the purpose of the proof of concept which is the underlying task in the Air-Cobot project and this thesis. A framework that can communicate with the robotic platform, all its sensors and two different PC's was developed during the course of this thesis. At the heart a Linux system running a ROS-Hydro distribution will allow the navigational supervisor module to choose the best controller for every situation the robot can find itself in.

This manuscript is organized as follows. The next three chapters are dedicated to the three above mentioned contributions. All the described methods are validated by simulation and experimentation results. The experimental tests have been conducted either in LAAS or in Airbus Industries site in Blagnac. Finally, a conclusion ends this work and highlights the most promising prospects.

2. Navigation Strategy

Contents

The purpose of this chapter is to present the strategy which has been chosen to make the robot safely navigate through the airport. The main issue will be the highly dynamic characteristic of the considered environment with human operators and dedicated vehicles.

Considering the task to be performed, the entire navigation strategy for the Air-Cobot can be divided into three main modes / phases. Those three phases will be presented in the following and will be mentioned throughout this work:

- " an autonomous approach phase ", in which the robotic platform moves from charging station to aircraft parking area (inspection side) following a dedicated lane for autonomous vehicles on the airport. Hence, to fulfill its mission, the robot has to perform a large displacement along that lane, as the initial and final points might be far apart from each other. In addition, the space to be crossed is rarely cluttered with obstacles because it is dedicated for the robot alone. However, a contingency plan has to be put in place. One thing that simplifies the task at hand is that an airport is indeed a highly structured / constrained environment. This results in a large amount of a priori available informations which can be merged into metric maps or can be taken into consideration when designing topological maps. Thus a priori information will have to be given to the robot in order to avoid any forbidden path.

- " an autonomous inspection phase ", where the robot moves relative to the aircraft throughout a cluttered environment. To perform this mission, the robot will have to reach each checkpoint that was chosen specifically for each aircraft type robustly and successively. During this phase, a large displacement, due to the size of the aircraft, has to be realized. Borrowing from the concept of " Divide and Conquer " this large displacement is made of small partially independent motions. In addition, strong security requirements have to be considered, as the robot must avoid any obstacle present in its surroundings (namely: the aircraft, various transport vehicles, maintenance or flight crew, forbidden fueling zones, etc.). During this phase the robot will remain in relative close vicinity of the aircraft since it symbolizes its main means of self-localization. This particular phase also represents the main proof of concept scenario for the Air-Cobot project.

- " a collaborative inspection phase ", where the robot follows the co-pilot or a maintenance operator. Here, the autonomy of the robot is reduced, as the motion is defined by the person in charge (called the operator from here on out) of the pre-flight inspection. However, the full band of security functions is still needed. Even though the operator by him / herself represents an obstacle it is necessary for the robotic platform to still consider other objects that might cross the virtual tether. Furthermore, a tracking is essential to prevent false object following, meaning, that the half silhouette of the operator must be kept during this phase at all times.

The PhD thesis at hand is mainly focused on the two first modes / phases for which a high level of autonomy is required. To achieve the considered challenge it is neces-

sary to define an efficient navigation strategy able to perform long displacements while guaranteeing non-collision, both in a large spaces as in close vicinity to the aircraft. Additionally, the proposed strategy needs to work in very different environments, typically on aircraft tarmac as well as in hangars. To gain further insight into the challenges, the next paragraph will review the state of the art on this general subject.

In robotics literature, it is common practice to divide the navigation strategies into two main groups [Choset et al., 2005, Siegwart and Nourbakhsh, 2004, Bonin-Font et al., 2008]. A first group which bundles approaches in need of extensive a priori knowledge is usually referred to as "global approach". In other words, if the robot does not receive a map about its environment prior to the mission, it cannot start [1]. It then " suffices " to extract and follow a path connecting the initial position to the goal to perform the task. This approach is suitable to realize large displacements because it offers a global view of the problem. However, it suffers from well-studied drawbacks: mainly, a lack of flexibility when dealing with unmapped obstacles, some sensitivity in the presence of errors (especially if metric data are used to build the map or to localize the robot), etc. From this information alone the analysis for the Air-Cobot project is that global strategies could be well adapted to perform the " autonomous approach phase ". Indeed, during this phase, the environment must be precisely described to the robot to avoid forbidden paths defined on the airport. Hence, a dedicated pre-navigation step using a SLAM technique then appears mandatory. SLAM, which is the acronym for **S**imultaneous **L**ocalization **A**nd **M**apping, is a well known approach to generate metric information about the system environment while continuously performing a self localization in the metric data. As previously mentioned, the charging station might be far from the aircraft parking / inspection area, forcing the robotic platform to perform a long-range displacement, thus, increasing the interest of this approach in our context. However, let us note that it might be challenging to avoid unmapped obstacles with this kind of approach.

The second group, called "local approach", mostly covers reactive techniques which do not require a large amount of a priori knowledge about the environment. The robot then moves through the scene depending on the goal to be reached and on the sensory data gathered during the navigation. These techniques allow to handle unexpected events (such as non mapped obstacles for instance) much more efficiently than the group of approaches described before. However, as no (or insufficient) global information is available, they are not well suited to perform long displacements. Hence, this approach seems to be more appropriate to implement the " autonomous inspection " during which the robot has to move relatively about the aircraft. Indeed, during this step of the mission, it will be necessary to avoid many unpredictable obstacles[2], either static (fuel or baggage trucks parked in the aircraft vicinity, fuel pipes, etc) or mobile (moving

[1] The map can be built either by the user or automatically during an exploration step.

[2] Remember that even though the robot is developed for the well structured environment of an airport, a contingency plan needs to be put in place for all probable eventualities.

vehicles, operators or flight crew, etc.).

It is obvious that these obstacles cannot be mapped before the mission, limiting the use of a global approach for this navigation. On the other hand, accounting for the aircraft size, it appears that a long-range displacement might be necessary to perform the preflight inspection. This is one of the main challenges to overcome while designing our strategy. Therefore, it is clear that the desired navigation strategy must integrate both local and global aspects to benefit from their respective advantages. For the mission at hand both approaches are complementary and the strategy must be built to take this characteristic into account. To do so, we propose an initial analysis of the problem at a lower level and we split the navigation system into the six following processes: perception, modeling, planning, localization, action and decision. All involved processes can be instantiated depending on the needs of each navigation mode. The current chapter aims to answer the following question: "How can the processes be instantiated accordingly to the embedded sensors and the navigation modes? "

To propose a suitable response to this question, a general review of the navigation processes is first presented before showing how they have been instantiated to fulfill the requirements of the Air-Cobot project.

2.1. Definition of the navigation processes and related works

The following section is devoted to the description of the processes involved in the navigation. As mentioned earlier, a total of six processes are generally highlighted: perception, modeling, planning, localization, action and decision. A wide range of techniques are available in the literature for each of them. As these processes cooperate within a framework to perform the navigation, they cannot be designed independently and an overview of the problem is required to select the most suitable among different available methods. We present here-below such an overview.

2.1.1. Perception

In order to acquire the data needed by the navigation module, a robot can be equipped with both " proprioceptive " and " exteroceptive " sensors. While the first group of sensors (odometers, gyroscopes, ...) provide data relative to the robot internal state, sensors of the second group (camera, laser telemeters, bumpers, ...) give information about its environment. The sensory data may be used by four processes: environment modeling, localization, decision and robot control.

2.1.2. Modeling

A navigation process generally requires an environment model. This model is initially built using *a priori* data. It cannot always be complete, and might evolve with time. In this case, the model is updated with the help to data acquired during the navigation. There exists two kinds of models, namely the metric and/or topologic maps [Siegwart and Nourbakhsh, 2004].

The metric map is a continuous or discrete representation of the free and occupied spaces. A global frame is defined and robot with respect to obstacles poses are known with more or less precision in this frame. The data must then be expressed in this frame to update the model [Choset et al., 2005].

The topologic map on the other hand is a discrete representation of the environment based on graphs [Choset et al., 2005]. Each node represents a continuous area of the scene defined by a characteristic property. All areas are naturally connected and can be limited to a unique scene point. Characteristic properties, chosen by the user, may be the feature visibility or its belonging to a same physical space. Moreover, if a couple of nodes verifies an adjacency condition, then they are connected. The adjacency condition is chosen prior to the mission execution by the user and may correspond for example to the existence of a path allowing to connect two sets, each of them represented by a node. A main advantage of topologic map is their reduced sensitivity to scene evolution: a topologic map has to be updated solely if there are modifications concerning the area represented by the nodes or the adjacency between two nodes. This is extremely helpful for the project at hand since the aircraft will not always be parked exactly at the same spot, its features however will rest unmoved relative to the aircraft frame.

Metric and topologic maps can be enhanced by adding sensory data, actions or control inputs to them. This information may be required to localize or control the robot. It is worth to mention here that hybrid representations of the environment exist based on both metric and topologic maps [Segvic et al., 2009], [Royer et al., 2007], [Victorino and Rives, 2004].

2.1.3. Localization

For navigation, two kinds of localizations are identified: the metric one and the topologic one [Siegwart and Nourbakhsh, 2004]. As mentioned a metric localization consists in calculating the robot pose with respect to a global or a local frame. To do so, a first solution can be to solely utilize proprioceptive data. However, such a solution can lead to significant errors [Cobzas and Zhang, 2001], [Wolf et al., 2002] for at the least three reasons. A first problem is due to the pose computation process which consists in successively integrating the acquired data. Put plainly, this process is likely to drift. The second issue is inherent to the model which is used to determine the pose: an error which occurs during the modeling step is automatically transferred to the pose computation.

Finally, the last one is related to phenomenons such as sliding (the Air-Cobot platform uses a skid steering drive control) which are not taken into account and due to its physical nature might never be. It is then necessary to consider additional exteroceptive information to be able to localize the robot properly. Visual odometry [Nister et al., 2004], [Comport et al., 2010], [Cheng et al., 2006] is an example of such a fusion.

Topological localization [Segvic et al., 2009], [Sim and Dudek, 1999], [Paletta et al., 2001], [Krösea et al., 2001] consists in relating the data provided by the sensors with the ones associated to the graph nodes which model the environment. Instead of determining a situation with respect to a frame, the goal is to evaluate the robot's bearings according to a graph [Siegwart and Nourbakhsh, 2004]. Topological localization is not as highly sensitive to measurement errors compared to its metric alter-ego. Its precision only depends on the accuracy with which the environment has been described and, naturally, the noise of the perceiving sensor.

2.1.4. Planning

The planning step consists in computing (taking a preliminary environment model into account) an itinerary allowing the robot to reach its final pose. For all purposes the itinerary may be a path, a trajectory, a set of poses to reach successively, ... There exists a large variety of planning methods depending on the environment modeling. An overview is presented hereafter.

One of the most straight forward approaches consists in computing the path or the trajectory using a metric map. To do so, it is recommended to transpose the geometric space into the configuration space. The configuration corresponds to a parametrization of the static robot state. The advantage is obvious. A robot with a complex geometry in the workspace configuration space [Siegwart and Nourbakhsh, 2004]. Especially for obstacle avoidance techniques this transformation reduces the computational cost of testing whether the robotic platform is in collision with objects in its world. After that, planning consists in finding a path or a trajectory in the configuration space allowing to reach the final configuration from the initial one [Choset et al., 2005]. Examples of this process for continuous representations of the environment, can be gained by applying methods such as visibility graphs or Voronoi diagrams [Okabe et al., 2000]. With a discrete scene model on the other hand, planning is performed thanks to methods from graph theory such as A^* or Dijkstra algorithms [Hart et al., 1968] [Dijkstra, 1971]. For the two kinds of maps, planning may be time consuming. A solution consists in using probabilistic planning techniques such as *probabilistic road-maps* [Kavraki et al., 1996] [Geraerts and Overmars, 2002] or *rapidly exploring random trees* [LaValle, 1998].

For situations in which the environment model is incomplete at the beginning of the navigation, unexpected obstacles may lie on the robot itinerary. A first solution to overcome this problem consists in adding the obstacles to the model and then to plan a new path or a new trajectory [Choset et al., 2005]. In [Khatib et al., 1997a, Lami-

raux et al., 2004a], authors propose to consider the trajectory as an elastic band which can be deformed if necessary. More examples of the same idea where further explored in [Rösmann et al., 2012], [Rosmann et al., 2013] with the **T**imed **E**lastic **B** (TEB) methods and in [Fox et al., 1997] [Gerkey and Konolige, 2008] with the **D**ynamic **W**indow **A**pproach. A global re-planning step can then be required for a major environment modification.

Using a topological map without any metric data leads to generate by an itinerary generally composed of a set of poses to reach successively. These poses can be expressed in a frame associated to the scene or to a sensor. The overlying itinerary is computed using general methods from the graph theory which have been mentioned in the prior paragraph. Depending on the precision degree used to describe the environment, it is conceivable to generate an itinerary which does not take into account all the obstacles. This issue then has to be considered during the motion realization.

2.1.5. Action

For the Air-Cobot to reach its designated inspection pose the mission has to be executed as planned. In order to execute the tasks required by the navigation, two kinds of controllers can be imagined: state feedback or output feedback [Levine, 1996]. For the first case, the task is generally defined by an error between the current robot state[3] and the desired one. To reduce this error over time, a state feedback is designed. Accordingly, the control law implementation requires to know the current state value and therefore a metric localization is needed.

The second proposed option (output / measurement feedback) consists in making an error between the current measure and the desired one vanish. This measure depends on the relative pose with respect to an element of the environment, called feature or landmark. Using vision as a means of generating a relevant analysis of the situation, the measures correspond to image data (points, lines, moments, etc. [Chaumette, 2004]). Proximity sensors on the other hand can provide distances to the closest obstacles or other objects of interest in the environment through sensors such as: ultrasound [Folio and Cadenat, 2005] and laser range finder [Victorino and Rives, 2004], [Thrun et al., 2000] telemeters. The measures are directly used in the control law computation, which means that no metric localization is required. For this approach to work the landmark must be perceptible during the entire navigation to compute the control inputs.

2.1.6. Decision

To perform a navigation, it may be necessary to take decisions at different process states. Especially projects with aspirations such as Air-Cobot - designing a robot to be able to

[3]Here the state may correspond to the robot pose with respect to a global frame or to a given landmark [Hutchinson et al., 1996a], [Bellot and Danes, 2001].

operate organically in a normal day to day airport environment - it is obvious that in such complex environments decision have to be taken from time to time. It is worth to mention here that for this thesis, only the deterministic decision making process will be explained. Machine learning approaches have found there way into the project, but mainly through image processing and not in the decision modules.

Decisions may concern high level, e.g. a re-planning step [Lamiraux et al., 2004a], or low level, e.g. the applied control law [Folio and Cadenat, 2005]. They are usually taken by supervision algorithms based on exteroceptive data.

2.2. Description of navigation processes in the Air-Cobot project

In this part, we briefly review the methods which have to be implemented for the navigation strategy to work efficiently. Furthermore, a presentation of the robot the software the main part of the software that was needed to achieve our contributions will be presented.

2.2.1. Air-Cobot platform

The following section will give a brief introduction of the hardware of the Air-Cobot robotic platform. All electronic equipment (sensors, computing units, communication hardware, etc.) is carried by the 4MOB platform provided by Sterela, an engineering constructor which usually works for the military sector. Figure 2.1 presents this 230 kg platform and the current state of sensor integration. The name of the platform is given by its four-wheel skid steering drive. With a maximum speed of 5 m/s[4], the platform can run for about 8 hours thanks to its lithium-ion batteries. With all the sensory however, this autonomy is reduced to 5 hours of operation. For safety reasons (the platform weights about 230 kg) the platform itself is equipped with two bumpers that are located in front and in the rear of the robot. If for some reason one of them is compressed the platform will stop all electric motors and engage its brakes until the "emergency stop" state is canceled manually by the operator.

Further sensors that where added for the Air-Cobot mission are:

- four Point Grey cameras;
- two Hokuyo laser range finders;
- one **G**lobal **P**ositioning **S**ystem (GPS)
- a single **I**nitial **M**easurement **U**nit (IMU)
- one **P**an-**T**ilt-**Z**oom (PTZ) camera

[4]However, to increase torque the Air-Cobot platform has a reduced speed of 2 m/s.

(a) Robot platform, 4MOB, designed by Sterela

(b) Robot platform fully equipped during an experimental test run next to an A320

Figure 2.1.: 4MOB platform and fully integrated Air-Cobot

- a highly precise Eva 3D scanner manufactured by Artec

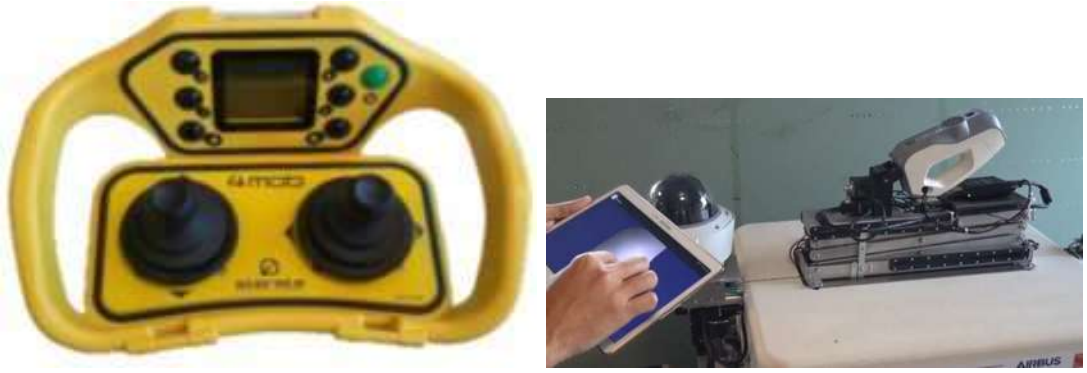
The sensor interfacing framework that is used throughout the project is the **R**obot **O**perating **System**, ROS (Hydro), on a Linux (12.04) machine. A second industrial computer that runs Microsoft Windows is mainly used for the interface with all the **N**one-**D**estructive **T**esting algortihms while the Linux machine hosts all supervisor and navigation functionalities.

Figure 2.2 shows the remote for the 4MOB Sterela platform and the possible tablet interaction with Air-Cobot. Currently this interaction is still quite basic (Go to, Switch to). However, in the future, the pilot is supposed to monitor the status of the pre-flight inspection through such an interface.

2.2.2. The Robot Operating System

The following paragraph will give a brief overview about the **R**obot **O**perating **S**ystem (or short ROS), the middleware that was used in the Air-Cobot project. One can basically refer to this software as the foundation of the Air-Cobot projects framework. This middleware allowed for fast integration of new features. Of course a new controller is much faster developed ('rapid prototyping') in MATLAB / MATLAB Simulink, which is why a MATLAB based simulation environment was used for exactly this reason. In Sections 2.2.7 starting on Page 56 our main prototyping environment will be presented.

Since testing these controllers and features on the robot is essential to achieve the



(a) Remote used by operator when close to the platform

(b) Tablet usage

Figure 2.2.: Remote of the 4MOB platform and tablet operation of Air-Cobot

goal of providing a prototype robot for autonomous aircraft inspection from scratch in the period of three years, the middleware had to be chose very early on during the project. Due to its widely accepted use in the scientific community, its portability and the amount of already available, open source software features, the team decided to utilize the **R**obot **O**perating **S**ystem. Several comprehensive introductions to ROS are already available [Quigley et al., 2009], [Martinez and Fernández, 2013] and [O'Kane, 2014]. Furthermore, in [Crick et al., 2011] researchers try to make the framework available to a broader audience by introducing Rosbridge. It is considered to be the most popular middleware in the community which does not mean that is the simplest, most robust, most generic, real time capable, etc., but it does mean that the support from the community is beyond example.

For this work and the Air-Cobot project ROS was chosen for the broad availability of already working hardware drivers (for cameras, robot CAN compatibility, interfaces for navigation modules) and reusable software stacks. Part of the navigation stack [Marder-Eppstein et al., 2010] of ROS was "recycled" to provide tracking of the robots pose in the environment with dead reckoning, IMU inputs, occasionally visual odometry and occasionally highly precise localization information that was taken as ground truth. All of this implemented in a extended Kalman filter to provide sufficiently accurate pose tracking even throughout lengthy time periods. The navigation stack is just one example why ROS can be extremely helpful.

A major downside of ROS however is that providing real time computation can be difficult. Since the software is running on top of a Linux desktop system it naturally inherits all of its short comings. These are surely minor for our prototype, since sampling control inputs for the robot can still be ensured around frequencies up to 30 Hz. Even if we consider dynamic obstacles at higher speeds the robot is able to break early enough, because equipped laser range finders can sense up to 50 meters ahead of the robot. Still, especially in areas where this 'viewing distance' is artificially shortened (corners

that keep obstacles in the sensor shadow, etc.) the robots speed needs to be adjusted accordingly.

2.2.3. Perception " exteroceptive" (cameras and lasers) and "proprioceptive" (odometer)

As mentioned in the previous subsection, different sensors have been mounted on the Air-Cobot platform. We review them briefly, showing how the data have been treated to be useful for the other processes. We first consider the exteroceptive sensors (vision and laser range finder) before treating the proprioceptive ones.

Detection and Tracking Features of interest by vision

Several cameras have been embedded on our robot. They are positioned on top of stereo camera systems (stereo basis of 0.3 meter) at the front and at the rear of the robot. Furthermore, a Pan-Tilt-Zoom camera system which usually is used for surveillance has been integrated on the platform. We first detail their specifications before explaining the image features processing implemented on the robot.

Specifications of cameras The four cameras that are used on the stereo camera systems (front and rear) are provided by PointGrey (now FLIR). All of them are of the type $FL3-FW-14S3C-C$ with 1.4 mega pixels, a global shutter, a maximum resolution of 1384 by 1032 [5]. and a 4.65 μm pixel size. Furthermore, the focal length is 6 mm and with four cameras the maximum refresh rate that the platform can handle is 25 Hz [6]. These cameras are all connected via $1394b$ firewire.

The second vision sensor in use is a PTZ camera by the producer Axis. Mainly used for the None-destructive testing part of the mission this camera does not have a global shutter. A global shutter is very important for the robustness of navigation algorithms. This camera is equipped with a photo sensor that features a much higher resolution 1920 by 1080. Furthermore, the camera system itself is able to rotate about completely about its vertical axis and 180 degrees about its horizontal one.

Target detection and tracking algorithm An algorithm "Track Feature Behavior" allows for tracking features while the robot remains fixed. We simply compute the center of all available features currently tracked and keep this point in the center of the image plane. This algorithm is mostly used for moving the cameras into a more favorable position (e.g., for visual servoing reinitialization) during the execution of missions. In addition, the robot does not need to be kept stationary while executing this algorithm, if the module is provided with localization information from odometry, GPS or visual

[5] The maximum resolution used in the project is 800 by 600 to allow for faster refresh rates

[6] Here the platform is the bottleneck since the theoretical limit for the cameras at $640x480$ would be 75 Hz

odometry (see the following sections). During the later presented experiments the presented technique is used for turns with high angular to linear velocity (sharp turns). Due to the shaking that these turns can cause, a robust feature tracking algorithm is necessary.

Detection and tracking Obstacles by laser range finders

Now we consider the second exteroceptive type of sensor mounted on our robot, the laser range finder (or short LRF). As previously mentioned, two laser range finders have been installed: one in the front and the other one in the rear. We first detail the specifications of the chosen sensor before focusing on the algorithm developed to detect static and moving obstacles.

Specifications of the Hokuyo laser range finder The obstacle detection is performed with the help of two Hokuyo UTM−30LX laser range finders (operating at wavelength of $870nm$). One Hokuyo LRF is located in the front of the robot while the other one is located at the rear. These laser range finders each have a range of 270° with a resolution of 0.25° meaning that there are 1080 measurements sampled on the full range of the laser. The two LRFs are mounted on pan tilt units, therefore, enabling a $3D$ range acquisition of data. If the aspect is not very important for obstacle detection, it appears to be very valuable for the relative localization of the robot towards the aircraft as discussed in Section 2.2.5 (Page 45). Additionally, in the future the robot will equipped with a sonar belt, thus, eradicating blind spots due to laser placement and robot geometry. Contrary to current sensor sets for semi-autonomous vehicles such as the Tesla Model S, recent models of the Mercedes-Benz S-Class, etc, the blind spot on the Air-Cobot is extremely small.

Currently however, the two laser range finder solely provide the raw obstacle detection. The raw data is sent at a much higher interval ($50Hz$) than it is actually processed ($20Hz$). Processing this data is done in several steps. First, a cleaning operation allows to remove outliers. Then, a dedicated method provides additional informations about the obstacles such as an estimate about geometry and center of gravity or furthermore linear and angular speeds. The following section will give a more detailed insight into this process.

Obstacles detection and tracking Algorithm The following section details how obstacles are detected and tracked. Figure 2.3 and pseudo algorithm 2.1 show a brief overview about the data acquisition and the successive processing which are applied. First, the raw LRF measurements are read and filtered to remove outliers. Then, the remaining points are clustered into separate sets. Finally, a RANSAC method allows to classify the obstacles between circles and lines. Each step helps the system to refine / reduce the amount of raw data coming from the Hokuyo LRF. Finally it is possible to achieve a tracking of the most relevant objects in the scene which in turn allows object movement prediction, even if its only valid for a very short time period. This feature

can be extremely helpful to perform a smooth and robust obstacle avoidance, especially in case of dynamic obstacles occlusions or when the object lies in a robot blind spot.

Algorithm 2.1 Obstacle Detection Algorithm

1: Receive Hokuyo raw laser measurements
2: **procedure** LRF DATA PROCESSING ▷ Callback function of laser
3: Clean the raw data by removing isolated points or accumulations of points that are below a predefined threshold
4: Cluster the remaining points into separate obstacles
5: Perform a RANSAC on these separate obstacles to determine whether any of these obstacles are either lines or circles
6: **end procedure**
7: Distribute information to the supervisor and obstacle avoidance

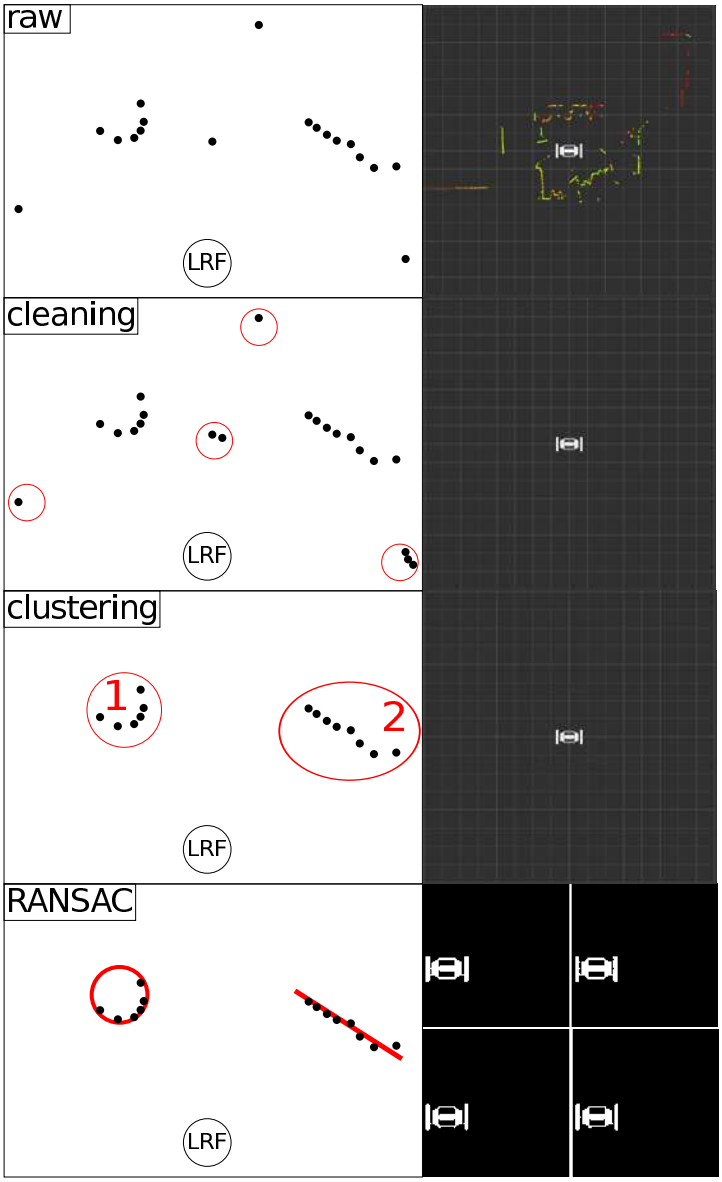


Figure 2.3.: Obstacle detection step by step

Figure 2.3 gives a step by step visualization of the concept explained in the pseudo algorithm 2.1. Each step helps the system to refine / reduce the amount of raw data coming from the Hokuyo laser range finder. Finally it is possible to achieve a tracking of the most relevant objects in the scene which in turn allows object movement prediction even if its only valid for a very short time period.

Odometry

The following section will review the different platform odometry systems. The basic odometry at hand has been progressively improved by adding more and more advanced techniques whose results have been fused (see the localization process for more details).

Wheel Odometry Now, we focus on the first proprioceptive sensor: the wheel odometry. It has the advantage to be very simple and fast. The Air-Cobot platform is equipped with wheel tick sensors. Through simple geometric relationships (wheel diameter, axle distance, etc.), the wheel movements can be translated into a full body motion of the robot in the world-coordinate-system.

The platform linear and angular velocities ν_{robot} and ω_{robot} can be easily deduced from the four left and right wheels velocities, as shown by equations (2.1) and (2.2):

$$\nu_{robot} = \frac{v_{lin_r} + v_{lin_l}}{2} \tag{2.1}$$

$$\omega_{robot} = \frac{v_{lin_r} - v_{lin_l}}{2 \times d_{track-width}} \tag{2.2}$$

where v_{lin_r} represents the average between front right and rear right wheels and v_{lin_l} accordingly for the left side. These velocities (ν_{robot} and ω_{robot}) are expressed in the points defined by an imaginary axle exactly in the middle of front and rear driving axle.

Inertial Measurement Unit (IMU) Odometry Our robot is also equipped with an **I**nertial **M**easurement **U**nit (IMU). It consists of three accelerometers that register the linear acceleration around x, y, z and three gyroscopes which measure the angular velocities around these last axes. This sensor can help out in areas where the wheel odometry fails (see section 2.2.5).

2.2.4. Modeling: both topological and metric maps

Now, we present the modeling process. As previously mentioned, three maps are available, a topological one and two metric ones. The two metric maps differ in their coordinate origin. The first one is centered in the robot last reset point and the second in the aircraft to be inspected. Let us first consider the topological map:

Topological map

As explained earlier, the topological map is made of the sequence of checkpoints to be reached. This list is known before the navigation begins and is derived from an existent checking schedule for airport staff or aircraft crew. The map has been optimized and the number of checkpoints reduced, as the robot is enhanced with sensors which allow to inspect several items at a time[7]. Figure 2.4 shows an example of such a map. As stated in the introduction, the main advantage of the topological map that it is in general more robust than a metric map, when it comes to localization.

The checkpoints are expressed both in terms of visual features and of relative positions with respect to the aircraft plane. This is possible because a 3D model of the aircraft is

[7]This is possible if the checkpoint is carefully chosen.

also available. This "double" description of the checkpoints in a visual or a metric manner is very useful to increase the overall robustness of the navigation strategy. Indeed, it allows to deal with the problem of temporary visual features losses whenever they occur due to an occlusion or an image processing failure.

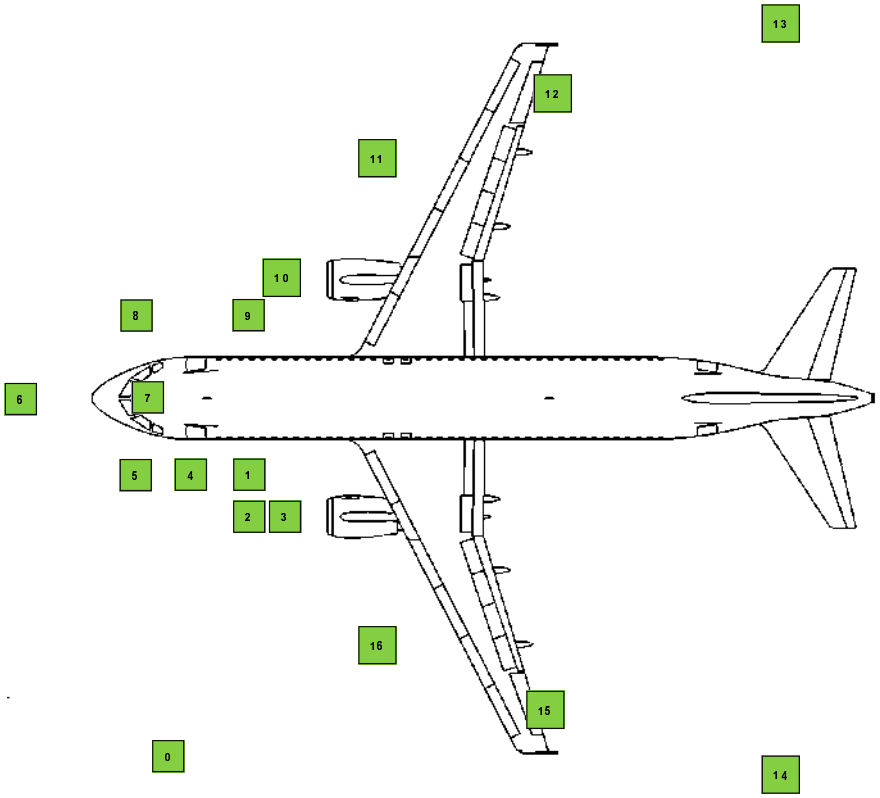


Figure 2.4.: Example of a navigation plan around the aircraft

Metric map

As mentioned earlier, in addition to the topological map, two metric maps are available. The first one is centered on the robot last reset point and the second one on the aircraft to be inspected. They are generated from the data provided by the laser range finder located at the front and at the rear of the robot (e.g.: SLAM). Through knowledge about the detected objects position, it is possible to fuse the incoming data onto a map. Map building is performed with a software from the ROS-Nav-Stack. Figure 2.5 shows an example of such a map (top view). It has been generated by the embedded Hokuyo LRF. In the map center, one can see the robot represented with a simplified geometric form. The squares in this example have the size of $1m^2$ and all dots are instances in which the laser rays have been reflected by the environment. Each color provides an indication of the materials emission coefficient. This map can help the user to put the data into perspective.

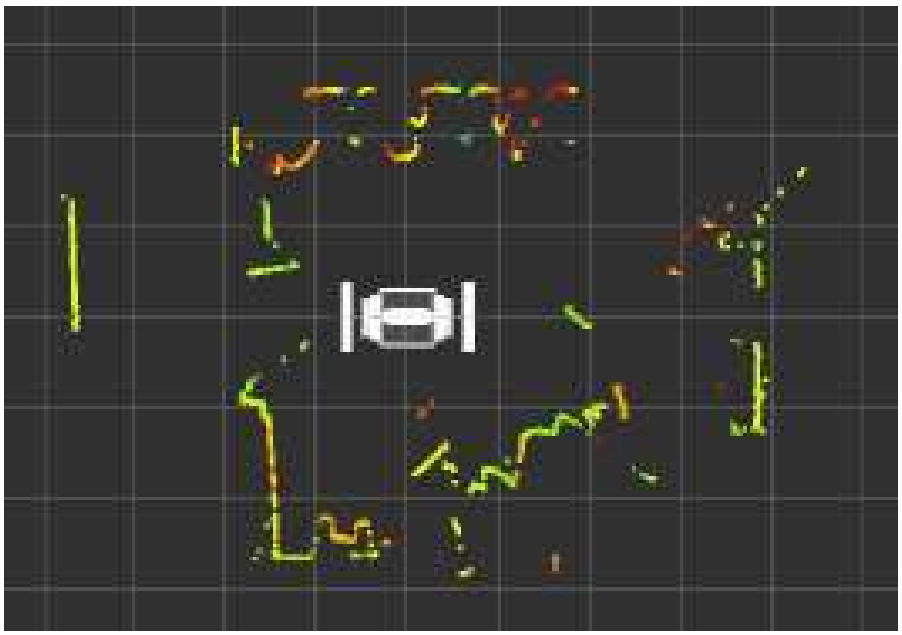


Figure 2.5.: Example of a metric map.

Figure 2.5 shows an example of what a metric map generated by the Hokuyo laser ranger finder can look like. In the center of the map a simplified outline of the robot's geometry (top down view) is given. This can help the user to put the data into perspective. The squares in this example have the size of $1m^2$ and all dots are instances in which the laser rays have been reflected by the environment. Each color provides an indication of the materials emission coefficient.

Costmaps A costmap is a special group of metric maps. Usually, a sensor is used to build an occupancy grid around the robot. This grid determines where in the vicinity of the robot an area can be classified as 'free-space' and where an area is 'occupied' by obstacles. the costmap takes this approach one step further by assigning cost values to each atomic part of this grid. During later navigation planning steps it is now possible to assign a total cost to a proposed path for the robot. The main advantage of this technique is that an evaluation of a huge amount of path choices is extremely fast, since it is only necessary to sort them by their respective total cost and then choose the 'cheapest' solution.

2.2.5. Absolute and relative localization

Similarly to the modeling section (see Pages 43 and following), the localization of the Air-Cobot platform is performed first in an absolute frame and secondly in the coordinate frames chosen in relation to the task. These relative frames are either entangled directly with the last checkpoint visited by the robot or attached to the aircraft that needs to be inspected. In the following section, we present these different localization coordinate

systems.

Absolute localization

The absolute coordinate frame is chosen by the Air-Cobot navigational base module. Our implementation of ROS navigation stack provides a localization of the chosen origin at each initialization. In its future field of application, this point will most likely be defined by the robot charging station, should the robot be kept indoors where the GPS reception is quite poor. Figure 2.6 shows such a situation. The robot started just at the origin of the coordinate system which is now taken as the absolute reference frame.

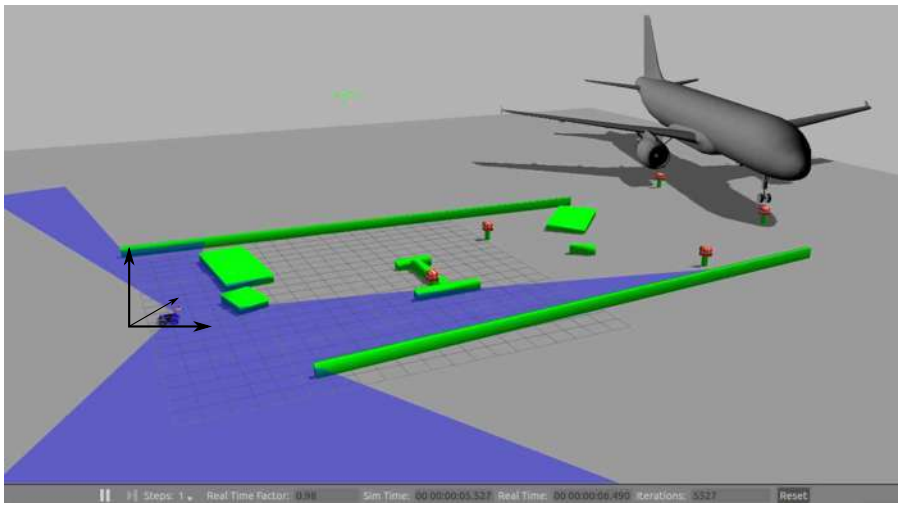


Figure 2.6.: Example of the absolute robot localization

Furthermore, the absolute localization can be performed by a GPS sensor which has been integrated on the platform towards the end of the project. Whenever a signal is available the absolute reference frame will be provided by the GPS signal. The scenario that is the basis of Figure 2.6 will be provided, in full, in the result section of Chapter 4 (see Page 143. Here, the robotic platform was not equipped with a GPS receiver, therefore, the localization is provided through a module that fuses a simulated IMU and wheel tick sensors.

Thus Figure 2.6, although suggesting a simulation in the outside environment, shows an example of a proper indoor experiment(e.g.: hangar), or a situation in which the GPS signal reception is not sufficient for other reasons..

A sensor, such as the GPS signal receiver installed on the Air-Cobot platform towards the end of the project, allows for subsequent updates on the robot absolute pose in a global frame. These positions will be used to guide the robot during the autonomous approach phase. In the aircraft neighborhood, relative localizations techniques will be preferred because of bad signal receptions close and under the plane. The GPS informations allow to enhance the previously mentioned metric map to take into account

forbidden airport areas (ground hatches for cables, parking areas for other ramps or vehicles, hazardous and dangerous zones such as fueling areas, etc.). Indeed, thanks to this data, it is possible to map these areas as obstacles. This approach is known as geo-fencing.

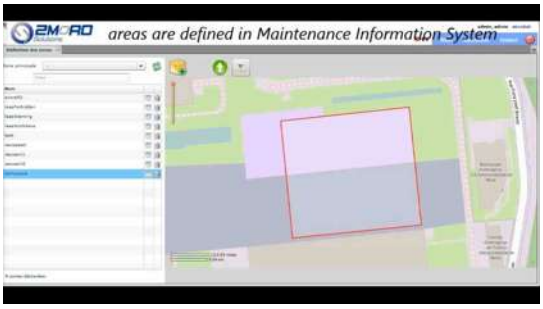
Figure 2.7 displays the integration / testing of the geo-fencing module during early 2016 on the airfield of Airbus close to Blagnac, Toulouse, France. To evaluate this module, the robot is guided manually into a restricted zone that was previously marked out visually for the operator, but also provided to the module as GPS coordinates. As a security feature, this module runs outside of the Air-Cobot navigational module and is therefore able to overwrite all its control commands. The test was a success: whenever the robot entered the forbidden zones, it was stopped immediately.

Relative localization

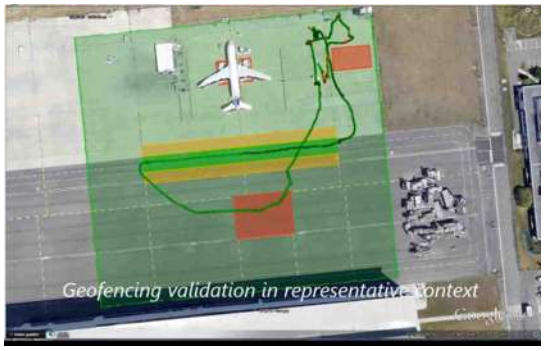
As explained above, relative localization techniques have also been implemented and are always available in addition to the absolute pose provided by the GPS (Figure 2.8). The figure shows an image taken from the video stream of the robot (Point Grey camera sensor) that has been augmented with additional information about the pose of the detected feature relative to the position of the robot. In the figure, several numbers are presented next to features that were detected and recognized by the robot. Each number represents a distance estimation of that feature towards the robot center of gravity. Should the feature be lost, due occlusions or sensor failure, the robot is still able to keep the sensor pointed in the correct orientation due to the self (relative) localization with the help of other features. Indeed, this localization appears to be extremely useful, especially in the second phase of the mission where the robot is expected to move around the aircraft. It allows to deal with sudden losses of the visual features and on top of that, provides a reliable back up technique whenever GPS is not available. The GPS signal will deteriorate fast the closer we get to the aircraft and of course, as soon as the robot enters a hangar.

However more importantly, it also allows to take into account the fact that the aircraft can never been parked at the exact same spot. There is indeed an uncertainty that must be taken into account. In this context, a more flexible localization, relative to the aircraft nose cone, the aircraft wings, etc. is recommended.

For the Air-Cobot project, two reference frames are set for the relative localization. The first one, which is also the most important, has been fixed on the aircraft front nose cone. This frame is orientated so that its x-axis is moving straight to its tail and that the z-axis is aimed at the ground. All checkpoints are expressed in this coordinate frame. The secondary coordinate system is only a back-up localization aid whenever absolute and relative (to aircraft) localization is unavailable. Its origin is always set into the latest checkpoint that was visited by the robot. This way the platform is able to move back and restart from a point that is already known with a great accuracy.



(a) Geo-fencing software interface provided by 2MORO



(b) Overlay of the robots trajectory in a map

(c) Picture taking during the acquisition

Figure 2.7.: Example of the VS environment experiments were conducted in

Figure 2.8.: Example of the relative robot localization

To perform this relative localization, in addition to the classical odometry solutions detailed in the perception process, several complementary methods have been implemented.

Point-Cloud-Matching During the mission, it is possible that Air-Cobot runs into situations where an on-line localization appears to be difficult. Whenever the confidence for the current localization goes below a certain threshold, a cloud matching technique is invoked. The confidence can drop due to various reasons: a loss of visual features, an increasing drift in the odometry, or a software problem, etc.

When the cloud-matching technique is active, it will force the robot into a stop. In this perspective the point-clod-matching behavior is much like the emergency stop behavior, except that the PTU of the front laser range finder is used to get a 3D point cloud of the surroundings. This cloud is obtained by first registering the distances recorded by the planar laser range finder. A second step uses the position and orientation of the PTU to transform these distances into 3D points. This process is detailed in Algorithm 2.2. Once the cloud is available, it is necessary to classify the points, because only some of them really belong to the aircraft. To do so, two major matching algorithms (**I**terative **C**losest **P**oint, ICP [Chen and Medioni, 1991], [Besl and McKay, 1992], [Zhang, 1994] ; and **RAN**dom **S**ample **C**onsensus, RANSAC [Fischler and Bolles, 1981]) are used. The way this latter method has been applied on our robot is described in algorithm 2.3. The later used ICP algorithm is much more effective if the preliminary solution is already close to the best solution possible.

Algorithm 2.2 3D Point Cloud Matching

1: **procedure** ACQUIRE POSE ESTIMATION USING HOKUYO NAD 3D MODEL OF THE AIRCRAFT ▷ Called by Supervisor
2: Stop robot
3: Start moving the pan tilt unit of the laser range finder form lowest to highest pan angle while simultaneously saving the acquired data to build an unfiltered 3d point cloud of the environment
4: Clean the acquired 3D point cloud (sampling, clustering, cylindrical segmentation)
5: Perform a modified RANSAC algorithm until either the global matching error is below a certain threshold or a time limit is reached
6: Perform an ICP matching
7: Send result to supervisor
8: **end procedure**
9: Supervisor updates current robot pose in aircraft frame and continues with the given pre-flight check

Algorithm 2.3 Modified RANSAC

1: **procedure** MODIFIED RANSAC ▷ Called by 3d point matching
2: Calculate characteristic of each point in laser range finder scan of the scene and the 3d model of the aircraft
3: Take three points of the model and compare these characteristics and geometry
4: Apply transformation of the most likely point set onto the 3d point cloud
5: Calculate the global error
6: **end procedure**

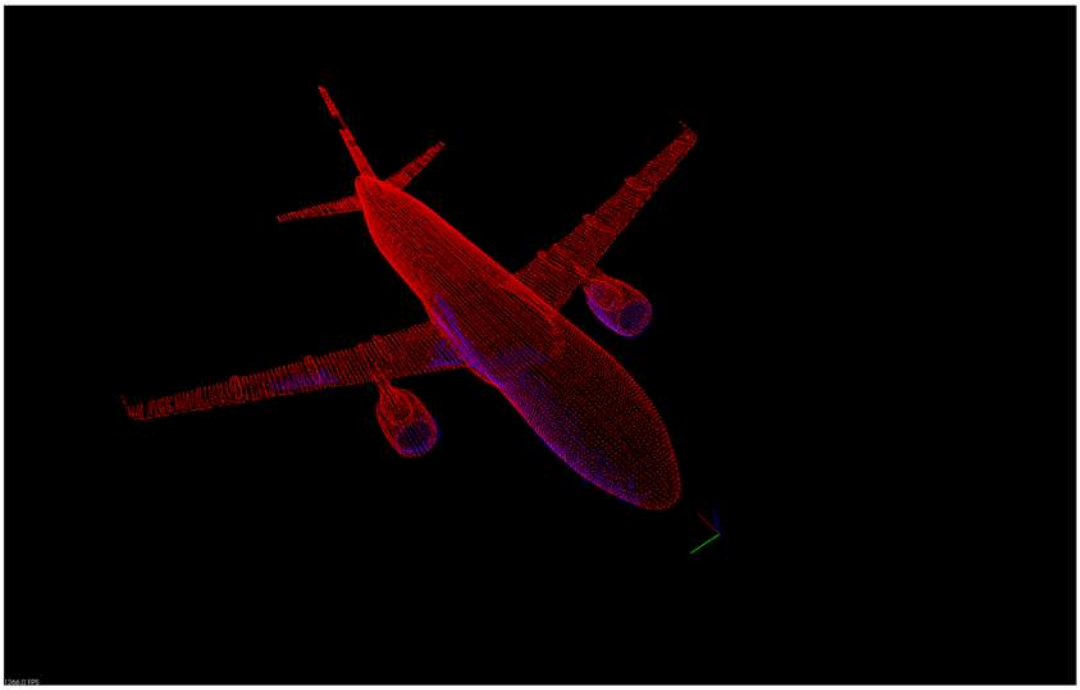


Figure 2.9.: 3d model of an Airbus A320 that was used for matching scans acquired for pose estimation in the aircraft reference frame (red = model points / blue = scanned points)

Figure 2.9 shows this process while the scans can be seen in Figures 2.10(a) and 2.10(b)

Semi Direct Visual (SVO) Odometry and ORB SLAM We have also implemented and tested two other relative localization techniques: the **S**emi-Direct **V**isual **O**dometry (SVO) [Forster et al., 2014] and ORB SLAM [Mur-Artal et al., 2015]. While SVO aims to generate movement information when analyzing camera sensor inputs, ORB-SLAM creates a metric map during its operation and locates the camera sensor in this global map. These methods can be seen as vision-based backup solutions when classical odometry fails. They have been evaluated on our system and the experiments have demonstrated that ORB SLAM in general fits better to the constraints of our project. Although SVO provides remarkable results for **M**icro **A**erial **V**ehicles (MAVs) for the robot, it suffers in our case from an initialization problem for a non-holonomic robot, when the cameras are aligned with the main motion axis. It is possible to obtain a better initialization when the cameras are turned $\pm 90°$, which requires to perform it prior to the inspection mission. This will of course leads to issues to reinitialize the system if it crashes during the mission. This is the reason why ORB-SLAM has been preferred. However, SLAM techniques often need a certain time and movement of the sensor (mono-cam SLAM) to be properly initialized. This is due to the fact that a key-frame can be constructed only if a sufficient amount of valid features in the image is provided. In our case, this sufficient amount of information can be hardly reached and

(a) 3d scan acquired with Hokuyo during experimentation on Airbus 320 - robot pose behind the wing

(b) 3d scan acquired with Hokuyo during experimentation on Airbus 320 - robot position under right wing

Figure 2.10.: Example of two scans that were acquired with an Airbus A320 in a hangar environment

no reliable informations can be deduced by this approach. We handle this problem by recording the movement with the IMU and odometry sensors. Later we use the resulting pose estimate as an offset whenever the SLAM is finally initialized.

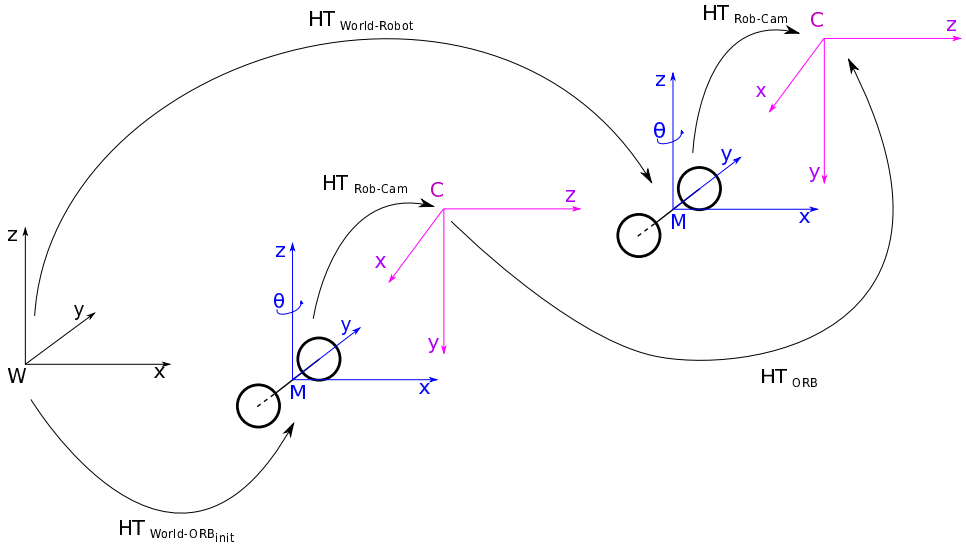


Figure 2.11.: Transformations during ORB-SLAM initialization

Let us consider Figure 2.11. The different frames used in our project are displayed: the world frame F_w, the robot frame F_m, the camera frame F_c. The figure also presents the homogeneous transformations which are involved in order to add the initialization movement offset ($HT_{World-ORB_{init}}$). This motion was tracked using the odometry and IMU data to find the current position of the robot ($HT_{World-Robot}$). Experiments have shown that the transformation during the initialization ($HT_{World-ORB_{init}}$) is much smaller than the actual pose estimation later calculated by the ORB-SLAM (HT_{ORB}).

$$HT_{World-Robot} = HT_{World-ORB_{init}} \times HT_{Robot-Cam} \times HT_{ORB} \times HT_{Cam-Robot} \tag{2.3}$$

At this step, we have several relative localization techniques at hand. The following section provides some more details on how these techniques were fused to generate one conclusive input for the navigation module.

Fusion principle In order to integrate all available robot self localization inputs, a fusion for all the raw data is necessary. The ROS navigation stack (robot_pose_ekf) provides a first solution to this problem: an extended Kalman filter dedicated to localization data fusion. By default the concerning package provides three interfaces for localization inputs:

- the first one is fed by the fusion of odometric data concerning wheel displacement. The method is explained here below.

- the second one is filled by the output of the IMU (**I**nertial **M**easurement **U**nit) that is located in the center of gravity of the robot.
- the last one is either provided by an SVO or an ORB-SLAM [Mur-Artal et al., 2015] implementation. For the moment, a simple switch between these techniques is realized. However, should the computational performance allow running both, an additional fusion will be built.

We address here below the principle of the fusion between wheel and IMU odometry. These data are combined to reduce a major downside of wheel odometry: its sensitivity to drift. The further the robot has traveled, the less reliable the information recorded through wheel encoders gets. However, this unwanted behavior gets even worse on a skid steering robot such as Air-Cobot. Since it does not have a steering axle, it is impossible to change its orientation without skidding on the ground. Hence, a localization based on solely on wheel odometry will fail over time. While driving in a straight line still allows an acceptable movement tracking precision, the confidence in the measure accuracy drops very fast as soon as tight curves need to be taken during a inspection mission.

It is then necessary to overcome this problem and try to cancel out the drift. Skidding or slippage of the wheels for example is not registered and therefore does not enter as false movement if a filtering process is used to combine IMU and wheel odometry.

Therefore, the straight forward solution commonly applied in the robotic community is to fuse wheel odometry and IMU to improve localization precision. A first solution is of course to use the navigation stack as explained before, but this would mean using two inputs out of three, only for odometry. In addition, the IMU must be calibrated after each system restart. Finally, on a robot the size and weight of Air-Cobot (it is possible that the system starts bouncing when turning in place) it can be tedious to find the right parameters for the fusion and a changing friction coefficient can have very negative results on the localization.

Therefore, we have made further improvements to guarantee that the wheel odometry will be relevant. The idea is to use visual data to cancel out the drift. To do so, the robot is allowed to constantly scan its environment for familiar landmarks with respect to which it will be possible to move in the sequel. This constant scan appears to be a quite challenging task in our airfield environment for several reasons. First, in such an environment, landmarks are scarce. Second, as it is dynamic, all the detected objects are not necessarily relevant. For example, transportation vehicles like buses or baggage trucks might not be mapped before hand and aircraft types and positions might change throughout the airfield. Thus, during the approach phase, the robot is constantly searching for interest landmarks, but can only act on them when the confidence level exceeds a high threshold. During the autonomous inspection phase and especially once it gets in close proximity to the checkpoints, visual landmarks are well mapped and more simple to detect and track. It is then possible to use vision-based pose estimation methods to generate a drift free estimated pose of the robot towards the aircraft (let us

recall that each checkpoint is defined in the aircraft main frame). In this way, we use the last localization that we trust the most.

2.2.6. Planning

The following section gives more insight towards the situations during which planning is needed in the Air-Cobot Project. During the autonomous inspection phase, no real planning is needed, as the robot is expected to reach all the checkpoints given in the topological map. As previously explained, some points have been removed from the map because the robot is able to verify several items simultaneously. Hence the check-list (see Figure 2.4 on Page 44) of the human operator has been modified to play to the advantages of Air-Cobot.

However, during the approach phase, some planning is needed. Whether it is the execution of a trajectory re-play, or the simple path planning towards the operation starting point from the charging station. The first of these options can be executed either on GPS coordinates that have been recorded in an earlier instance or with the way points taken from the robot localization module.

2.2.7. The action process

This subsection gives a broad overview of the controllers which are available in the action process. The list is shown below:

1. Multi Visual Servoing (MVS) (see Chapter 3).
2. Obstacle Avoidance (OA) (see Chapter 4).
3. Go to Goal
4. Emergency Stop
5. Correct Camera Angle
6. None-Destructive-Testing
7. Laser servoing
8. Follow Operator
9. Wait Time

We now give a short description of the main controllers.

Multi Visual Servoing (MVS)

This controller is used in the second navigation mode where the inspection is performed. It allows to reach each checkpoint around the aircraft if the corresponding visual features are available. Different visual servoing techniques have been implemented: **I**mage **b**ased **V**isual **S**ervoing (IBVS), **P**osition-**b**ased **V**isual **S**ervoing (PBVS) and a mix of both. In this way, it is possible to benefit from their corresponding advantages [Chaumette and Hutchinson, 2006],[Kermorgant and Chaumette, 2011]. More detailed information is given in Chapter 3 on Page 75.

Obstacle Avoidance

The obstacle avoidance controller can be invoked during navigation modes (approach, autonomous and collaborative inspection). It is able to handle both static and dynamic obstacles thanks to the definition of a suitable avoidance spiral. Several approaches have been developed and will be discussed in Chapter 4 on Page 121.

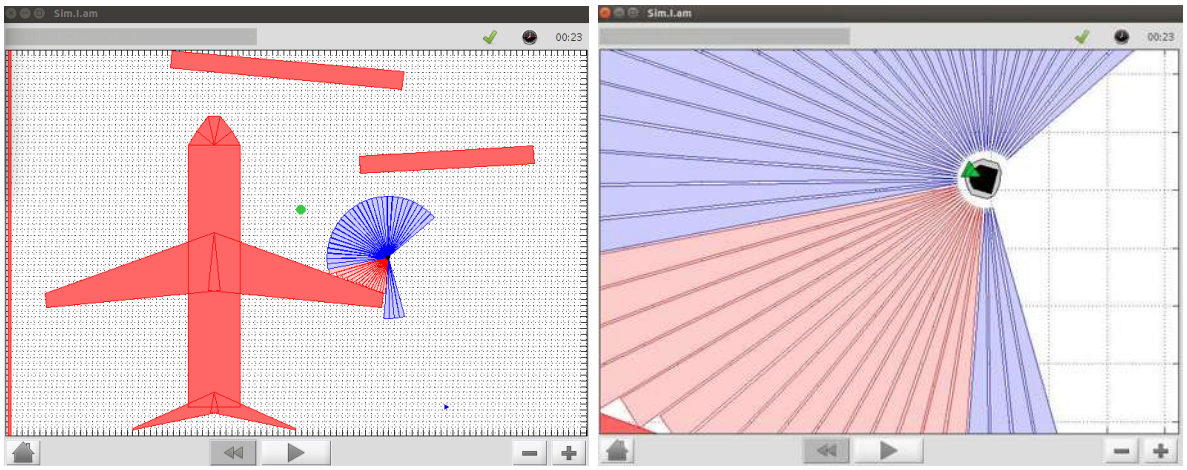
Go to Goal Controller

This controller is mainly used in the first phase of the mission. It will be denoted in the sequel by the acronym GTG.

SimIam robot simulator The controller proposed in this work relies on the control law that can be found in the SimIAM robot simulator[8] [de la Croix, 2013]. As stated in Section 2.2.2, the software was used for rapid prototyping of new features and control ideas. Figure 2.12 shows the same instance taken as a screen-shot for one of the maps implemented for the project. The simulator gives a 2d (bird view) of the scene and in Figure 2.12(a) the outline of the aircraft is clearly visible. Furthermore, we have provided a close up view in Figure 2.12(b) that allows to present the visualization of the robot. The blue and red triangles that originate from the robot represent the simple visualization the physical sensor range (here red means that an object has been detected and blue determines the opposite). Throughout the project, this simulator was used to check new ideas, improve existing controllers and tune parameters of the control law. Finally, it is worth while to mention that during our work with the controller, we have improved the simulator software with a feature that: allows to safe several environments and choose them at the start of the simulation; the possibility to include moving obstacles (which can move in straight lines, circles, etc.); added a new sensor type; etc..

Controller In the following section we will present the metric based controller that needed to be used if the visual navigation was sending inconclusive steering commands or was re-initializing due to some feature detection problem. Its objective is to make a robot reach a given position in an absolute frame (here, the airport frame). The idea is

[8]This simulator, which is based around a free MATLAB environment, has been used at the beginning of the project to rapidly prototype, design and test our solutions.



(a) General overview of the simulator execution interface

(b) Close up of the visualization of the robot

Figure 2.12.: Simulator that helped in the evaluation of new, rapidly prototyped ideas throughout the Air-Cobot project

first to align the direction of the robot with the one of the goal and then to move towards it. This "align first" behavior is desirable when navigating through a hangar environment or on the tarmac where the motions are reasonably constrained. In this case, turning the robot before moving and thus generating trajectories very close to straight lines is much safer. Note nonetheless that, if the environment is very highly cluttered, such a behavior is less interesting. This is the reason why the operator can choose at the beginning of the mission if the linear and angular motion are done successively or not. To do so, the idea is to reduce the angular error $\Delta\theta = \theta_d - \theta_c$ where θ_d and θ_c are respectively the target angular position and the current robot heading expressed in the airport frame (see figure 2.13).

Assuming that a sufficiently accurate localization (see Section 2.2.5) of the robot is provided, ω can be computed as shown below:

$$\begin{aligned} \omega &= atan2(dy, dx) \\ dx &= \cos\Delta\theta \\ dy &= \sin\Delta\theta \\ \Delta\theta &= \theta_d - \theta_c \end{aligned} \tag{2.4}$$

Note that we have decided to use the $atan2$ function to keep the robot control angular values within the interval $[-\pi, \pi]$. The controller then allows to align the direction of the robot to the one of the target. This is done as shown on Figure 2.14.

A target heading too far off the current robot heading (red part of the angle) will reduce the speed of the robot until complete halt and start turning it towards the target until the difference has been reduced down to an acceptable angle (green part of the angle). This acceptable angle is dependent on the distance towards the target. The closer the

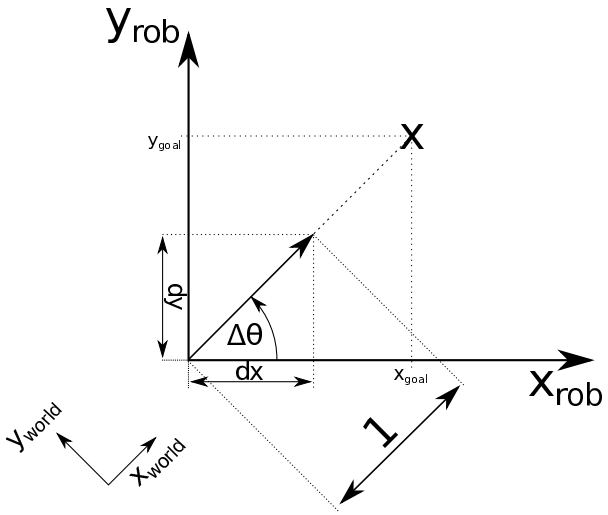


Figure 2.13.: Basics of the mobile base angular velocity ω calculation.

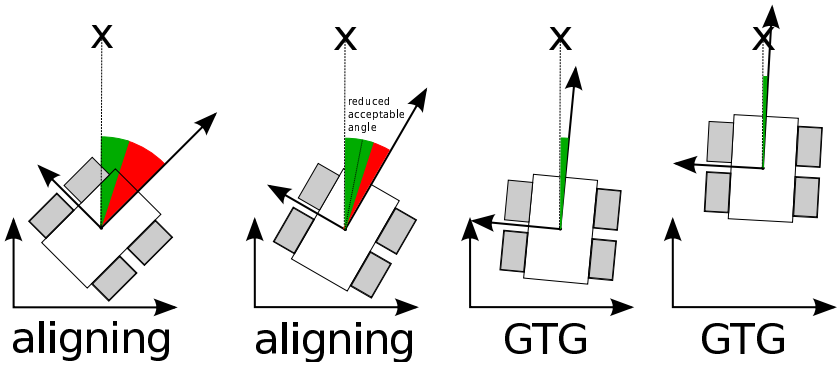


Figure 2.14.: Adjustment to the standard go-to-goal behavior

target, the more narrow that range is set. Furthermore, once the controller is aligning the angle, the acceptable range is decreased (see second sub-figure in Figure 2.14) much like in a hysteresis to ensure a smoother motion of the robot when it will start moving towards the target. To do so, an easy solution is to impose a trapeze profile to define the linear velocity.

We have then also taken into account two characteristics of the Air-Cobot platform. First, its form demonstrates some symmetry and, in addition, the same sensors are mounted at its front and its rear. Thus, moving in reverse is not a problem and can even save valuable time, especially when the target is located behind the robot, that is when target heading θ_d is greater than $\frac{\Pi}{2}$ or below to $-\frac{\Pi}{2}$. In such cases, the actual heading of Air-Cobot will be changed as shown in Figure 2.15 and in Equation 2.5:

$$\theta \leftarrow \quad (-1) \times (\Pi - \theta) \tag{2.5}$$

The above control is well adapted to unicycle robots such as Khepera, Turtlebot or

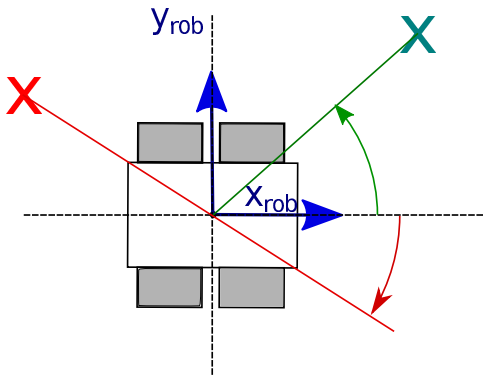


Figure 2.15.: Aligning first with a target "in front" and "behind" the robot

IRobot platforms. However, it appears to be limited with a skid-steering system such as Air-Cobot because of tire wear. Indeed, our platform has a lot of troubles when turning in place (shaky, wobbly behavior the higher the demanded angular velocity). We have then come up with an approach that always allows for a minimal linear velocity while still ensuring a minimal turning area. The idea is taken from maneuvering automobiles on narrow roads. When trying to achieve a u-turn in a small street, we have to perform maneuvers, moving the vehicle forward and backward. This can be done with the robot as well. It presents the advantage to increase its " in place "-turning speed, while reducing tire wear substantially and keeping unwanted shaking of the platform and its sensors to a minimum. Figure 2.16 visualizes the idea.

The red "X" marks the goal which was set for the robot. A zone is created (green area) in which the movement is still considered to be "turn in place". Air-Cobot will now try to keep its reference point (robot center of gravity) in this area. Since the angle between the current robot and target headings is bigger than 90°, the robot will start its aligning first behavior by moving backward while trying to minimize the angle difference. Once the reference point is close to the boundary, the linear movement will be reversed. Depending on the angle difference, this can take up to three steps (see Figure 2.16 (a) to (d)). If by then Air-Cobot has not been able to reduce the angle to a neglectable amount, the old align first behavior will be triggered. As we demonstrate in Algorithm 2.4 we keep the robot in a certain motion as long as it does not leave the zone we envisioned for the turning.

Correct Camera Angle Controller

This controller allows to correct the camera orientation when it is not suitable for the inspection task. It appears to be necessary when the visual servoing has led to a high pan angle for one of the stereo cameras systems. To fix the problem, this controller checks the difference between heading of the robot (θ_{robot}) and current angle of the active PTU camera systems. It then regulates this error to zero using the task function approach [Samson et al., 1991]. No linear velocity (ν) is applied. Only angular velocities ω are sent to the robot. Figure 2.17 shows how the controller is used when the front

Algorithm 2.4 Modified Align First Behavior

1: **procedure** DETERMINE NECESSITY OF ALIGN FIRST ▷ Called during go too goal execution
2: Estimate angle to target γ
3: **if** $|\gamma| \geq threshold$ **then**
4: $flag_{align} \leftarrow$ `TRUE`
5: **if** $|\gamma| \leq 90°$ **then**
6: **repeat**
7: `turn in direction of target (shortest angle difference)`
8: `move forward`
9: **until** boundary (Figure 2.16) reached OR ($|\gamma| \leq threshold - safety$)
10: **else**
11: **repeat**
12: `turn in direction of target (shortest angle difference)`
13: `move backward`
14: **until** boundary (Figure 2.16) reached OR ($|\gamma| \leq threshold - safety$)
15: **end if**
16: **else**
17: $flag_{align} \leftarrow$ `FALSE` State `execute: default go to goal controller`
18: **end if**
19: Send ω and υ to CAN-BUS
20: **end procedure**

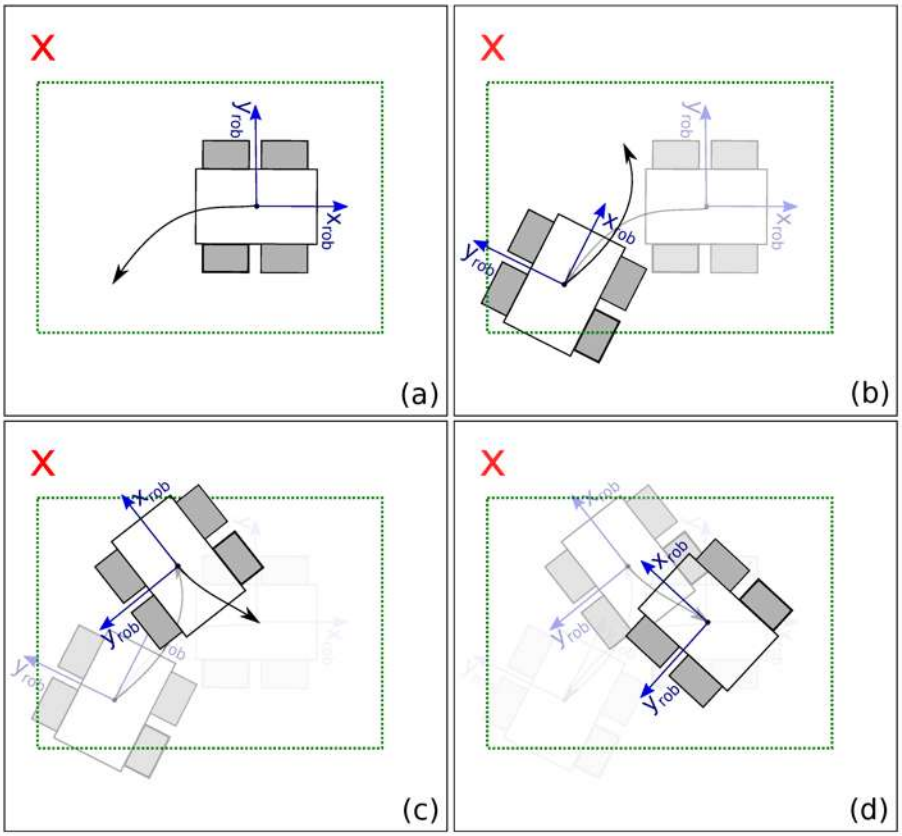


Figure 2.16.: Modification of the align first behavior to decrease tire wear.

PTU (**P**an-**T**ilt-**U**nit - see Figure A.6 on Page 183 in the appendix) camera systems are not aligned with the robot heading.

Emergency Stop

This controller is invoked when technical problems have been encountered during the mission. It can then be used at any time and during any phase of the navigation. Some problems are not very serious and it is possible to recover from them :

- Waiting for a node to initialize/wake up.
- Major threat of collision (the collision cannot be avoided with airport staff or any object). Here, the robot waits until a solution for avoidance is found.
- Impossible to perform the non-destructive testing because of an object lying in one checkpoint for example. In this case, the robot waits until the checkpoint is freed.
- A failure in the feature tracking (direct sunlight, airport staff passing in front of the active camera and thus occluding the target features, etc.) resulting in a crash of one of the tracking nodes. The nodes are then automatically re-spawned and the features tracked anew.

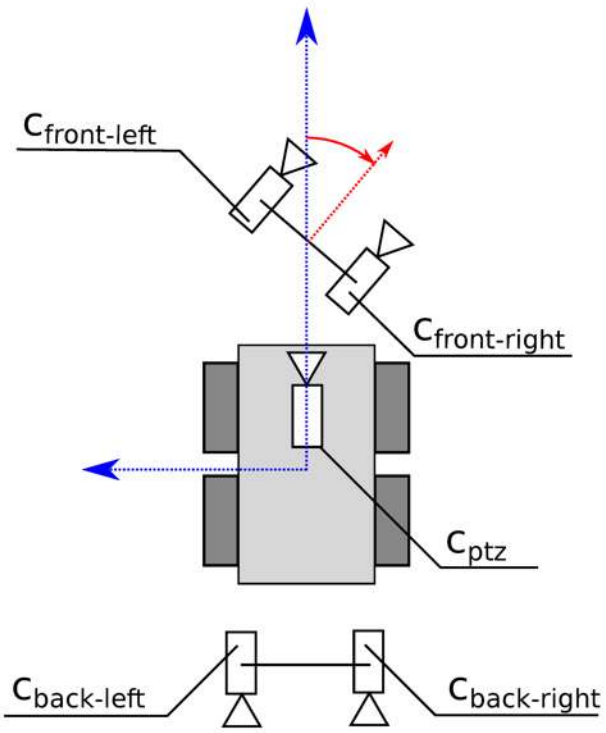


Figure 2.17.: Correct-Camera-Angle controller

In this case, once the problem is solved, a switch onto the last used controller before emergency stop occurs. Other more serious problems requires a definitive stop of the robot and have to be solved by the operator:

- A node has crashed and it cannot be re-spawned ;
- Too inaccurate localization ;
- Several missed attempts of finding back the features necessary for navigation (together with a too inaccurate localization).

In this case, the robot is definitively stopped until the problem is fixed by the operator.

Follow Operator controller

The Follow-Operator controller is used only during the third navigation phase. In this case, the robot is expected to follow a maintenance operator or the co-pilot to perform the inspection with him. Our platform then becomes a true cooperative robot, that is a true "cobot".

The Follow-Operator controller is generally launched when a problem has been detected during the mission: the localization is too inaccurate and the cloud-matching does not suffice to fix it, or the features have been lost and cannot be recovered, or an

object prevents the robot from reaching the checkpoint and no avoidance motion can be found, etc. This controller will then be usually invoked after the emergency-stop one.

In our case, it is an elegant way to overcome blocking situations which require a human intervention. Here, a message is sent to the operator asking him / her to help the robot. This way, this latter has enough time to position himself in front of the robot and manually enables the follower-controller. Such a procedure is necessary to ensure that the vehicle is not following the wrong staff member or even a mobile obstacle.

To determine the object to follow, the proposed method is based on the laser range information. A search through the planar informations determines the closest, most circular object available. This object will then be tracked while the robot tries to keep a constant distance towards its center point. The controller has been designed by AKKA Technologies, the partner in charge of the integration in the project. It relies on the existing control proposed in the ROS Nav-Stack [Marder-Eppstein et al., 2010], [O'Kane, 2014]. Once the problem has been overcome, the supervision algorithm invokes the next controller depending on the mission needs. This is possible since the localization module is running in the background throughout every phase. Hence, the navigation module can be re-engaged at any time.

None-Destructive-Testing (NDT) controller

This controller is intended to perform the None-Destructive-Testing whenever the Air-Cobot has reached a checkpoint. In these poses, the robot base remains fixed and will therefore, be sent the same control inputs as in the controller Emergency-Stop. The payload however, with all its sensors (cameras, laser range finders, 3D scanner, etc.) will be controlled according to the demands of the robot second PC which is dedicated to the inspection tasks alone. Hence, the two stereo camera systems, the two laser PTU, the PTZ camera and the 3D scanner mounted on the pantograph will be the only moving parts of the robot. Figure 2.18 displays the two different configurations of the robot (navigation/inspection).

Furthermore, figures 2.19(c), 2.19(b), 2.19(e) represent a view of the items that are visually inspected during each mission.

2.2.8. Decision

A supervision in form of a finite state machine defines the way the proposed strategy is executed. Figure 2.20 visualizes the different states and the majority of the guarding conditions schematically.

(a) Robot in navigation configuration (b) Robot in NDT configuration

Figure 2.18.: Picture of Air-Cobot in default configuration and with the pantograph-3d-scanner unit extended for inspection

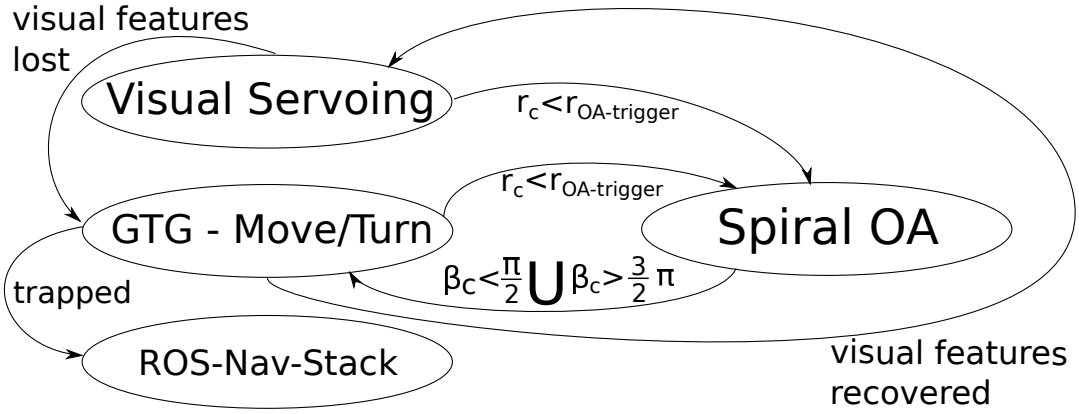
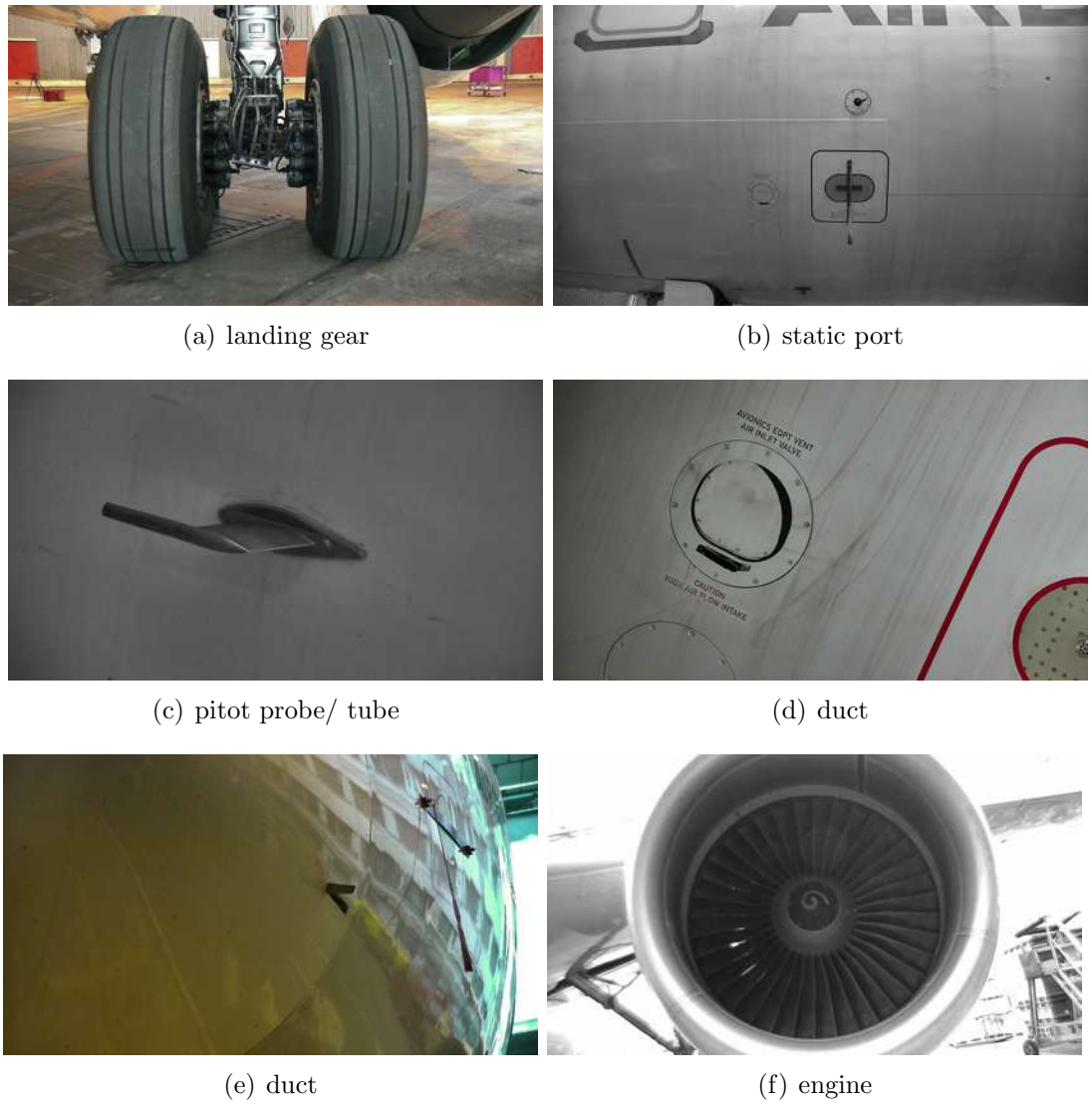


Figure 2.20.: Schematic of the state machine which defines the decision process

Here, a schematic is provided that explains how the supervisor is using guard conditions to evaluate which controller is to be chosen. The ellipses in this figure represent these different controllers and the arrows visualize the possible switching directions. So far we have only talked about the OA controller. However, to induce a motion towards a specific target, several controllers can be chosen.

Visual servoing (whenever visual cues can be detected), a simple move/turn - controller (if VS is not available) and finally the map-/trajectory-based controller presented in [Marder-Eppstein et al., 2010] which is implemented in the Navigation Stack of the Robot Operating System (ROS). The last mentioned controller is used as a backup solution for cases when the reactive controllers are stuck in a local minima.

Guarding conditions, or simply guards, are methods which tell the supervisor to switch behavior. They can be found along the arrows in Figure 2.20. Our guard for switching to OA is very basic. Whenever the LRF detects points (r_c) which are closer than the trigger distance,$r_{OA-trigger}$, the supervisor will switch to OA behavior. $r_{OA-trigger}$ is chosen so



(a) landing gear (b) static port

(c) pitot probe/ tube (d) duct

(e) duct (f) engine

Figure 2.19.: Collection of elements that are inspected in a pre-flight check

that a maneuver around the obstacle is always possible. For the experiments presented in Section 4.3, this condition is sufficient since no obstacle exceeds the robot maneuvering capabilities In an airport-scenario, a secondary condition (velocity of obstacle below threshold) must be met, otherwise an approach close to [Fiorini and Shiller, 1998] is used for OA. To disengage OA a technique is used that constantly monitors β_c and releases the robot whenever its value goes either below $\frac{\pi}{2}$ or above $\frac{3}{2}\pi$. By using an approach that helps to dynamically choose the center reference point for the obstacle avoidance this technique has proven to be very robust and safe. A more detailed explanation on this approach will be given in Chapter 4 starting from Page 121. Figure 2.20 also shows that during regular missions the visual servoing controller is the preferred controller and is only abandoned when all visual cues are lost. Towards finalization of the navigation

upon a checkpoint the NDT (**N**one-**D**estructive **T**esting) will be invoked if the camera pan angle is below a reasonable set threshold. Otherwise the robots orientation will be changed to allow lower panning angles. Finally, we should point out that each state can be exited onto the "Emergency Stop" state whenever the lowest current distance towards an obstacle (r_c) drops below the emergency stop trigger distance (r_{ES}). This distance is lower as the trigger distance of obstacle avoidance ($r_{OA-trigger}$). If for what ever conceivable reason an object would enter this security zone of the robot, all actions will be halted until either the operator provides a manually overwrite permission or the object is leaving the zone.

2.3. Instantiation of the navigation processes in Air-Cobot project

As previously mentioned, three navigation modes / phases have been defined for the Air-Cobot platform:

- The " autonomous approach",
- The " autonomous inspection ",
- The " collaborative inspection ".

As explained before, the first phase has a dominant global feature but some local skills have to be provided to guarantee a collision-free passage. On the other hand, the envisioned second phase is characterized by an important local aspect but the global information must not be forgotten for the mission success. As a consequence, the process instantiation will depend on the selected navigation phase. We now consider this problem.

2.3.1. Autonomous approach mode

Let us remind ourselves that first navigation phase relies on global localization data and on the airport metric map. When this mode is activated, a trajectory allowing to bring the robot from its initial position (generally, the charge station) to a close neighborhood of the aircraft is defined with respect to the airport frame. The previously mentioned "close neighborhood" is defined (always in the airport frame) by a particular point P which is chosen so that the initial checkpoint can be easily reached, making easier the transition between the two navigation modes. Then this trajectory is split into a sequence of several positions which can be successively reached by applying the go to goal (GTG) controller. If one or several obstacles are detected by the laser sensor, a dedicated controller is applied in order to guarantee non-collision and safety. A very detailed explanation of this controller is given in Chapter 4 starting from Page 121. This decision is taken by a supervision algorithm which is implemented as a state machine. To implement this reasoning, the processes have been instantiated as follows:

Processes	Autonomous approach mode.
Perception	laser range finder Wheel Odometry GPS IMU
Modeling	Absolute metric map
Localization	Absolute localization using GPS and IMU
Planning	Trajectory given in terms of successive absolute positions to be reached
Action	Obstacle avoidance Go to goal (GTG) controller
Decision	Supervision algorithm (state machine)

Table 2.1.: Description of the navigation strategy during the approach phase

2.3.2. Autonomous inspection mode

In this section let us consider the second navigation phase. This mode is activated once the robot reaches the final point P of the previously mentioned trajectory. From that time, only relative localization data are used [9]. This means that we will focus on information provided by vision, laser range finder and relative positions expressed with respect to the aircraft. The latter are given by the above mentioned relative localization techniques. The modeling process provides the topological map, that is the graph containing a checkpoints list. This list must be built prior to the mission execution and will differ for each aircraft type that is inspected. From this, the robot is controlled using multi-camera visual servoing controller to reach each checkpoint. This allows to obtain a high accuracy and practically no drift can be retained. Once a checkpoint is reached, an inspection process is launched and, when it is completed, the robot continues to the next one. If, during the motion, the visual features happens to be lost, then a procedure allowing to deal with the visual signal loss is launched, allowing to avoid the mission failure. A more detailed description of this process will be given in the Chapter 3 starting from Page 75. As mentioned, if one (or several) obstacle(s) is (are) detected, the avoidance controller is activated to guarantee collision-free robot guidance. Once the last checkpoint is reached, the robot will return to point P and then to the charging station thanks to the go to goal controller.

These decisions are taken by another dedicated supervision algorithm. To implement this reasoning, the processes have been instantiated as follows:

[9] The aircraft to be inspected is not always positioned exactly at the same place. As the automated inspection requires a high positioning accuracy, the mobile base motion cannot be performed using absolute data given by the GPS for instance. Even with high precise GPS and an aircraft parked correctly it is doubtful that a navigation on these absolute values should be consider at all. Especially close and under the aircraft such a system will fail. Relative localization information is mandatory for the task to be successful.

Processes	Autonomous inspection phase.
Perception	Odometry Vision Laser range finder
Modeling	Topological map
Localization	Relative localization with odometry, matching cloud and the localization part from ORB-SLAM algorithm.
Planning	No method is necessary (the path is directly given by the topological map)
Action	Multi-camera visual servoing Obstacle avoidance
Decision	Supervision algorithm(state machine)

Table 2.2.: Description of the navigation strategy during the autonomous inspection phase

2.3.3. Collaborative inspection mode

The third navigation mode corresponds to the collaborative inspection. In this case, the robot is expected to follow a human operator using the laser data. The idea is that he/she remains in a dedicated position in front of the robot before launching this mode. This way the robot is able to locate and evaluate the operator geometric shape (planar 2D shape) and start the tracking process. Obviously, in this case, the autonomy is reduced, leaving the processes uninstantiated, except the perception and the action ones.

Processes	collaborative inspection phase
Perception	laser range finder
Modeling	-
Localization	-
Planning	-
Action	Follower controller Obstacle avoidance
Decision	-

Table 2.3.: Description of the navigation strategy during the collaborative phase (Follower mode)

2.4. General overview about the navigational mode management and modularity of the framework

In the following section we will describe the concept on how our framework decides how to switch among the different phases. Additionally we will present the incorporation of

a metric-map-based exploration, map-building, navigation approach that will provide a suitable example for the versatility of our framework. The foundation of this approach was introduced in Section 2.2.4 on Page 45.

2.4.1. Overview of the management of the navigation modes

The decision module switches between the different navigation modes / phases. In the following section we will explain this high level state machine. An abstracted overview is presented in Figure 2.21. Mored in−depth information about each sub state was presented in this chapter. For the autonomous inspection phase an flow chart was given in Figure 2.20 on Page 64

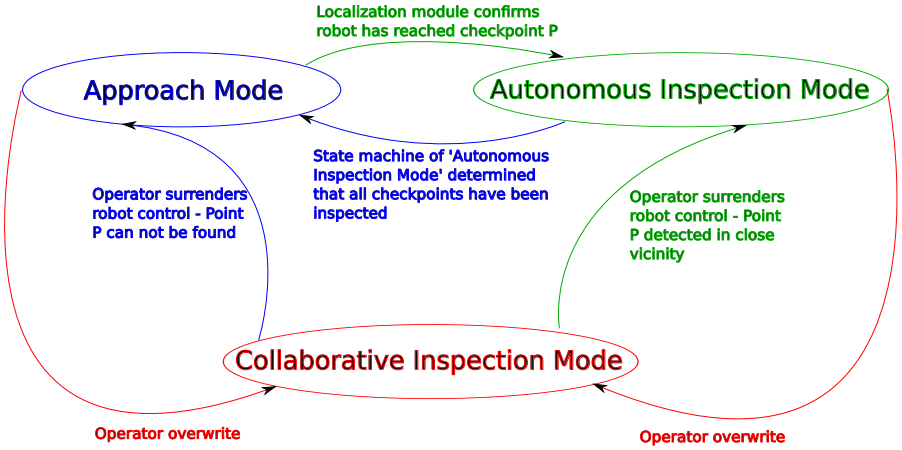


Figure 2.21.: A general overview of the high level state machine of Air-Cobot

Figure 2.21 highlights the three main navigational states of the robot that were introduced at the beginning of this chapter. Furthermore, transitions (among them) and their most basic and simple guarding conditions are shown. As one can see, the human operator can impose a switch towards the third navigation mode at any time for security reasons.

Allowing smooth transition among these controllers is not necessary. The reason being that these three states can be considered extremely high level (concerning their respective abstraction to machine language) Even if we implement a hard switch among these phases a lower level control module will apply jerk and acceleration limitation. This will not only prevent abrupt motion of the robot which might be bad for any module tracking visual cues in the camera image, but it will also extend overall lifespan of the tires and the electric motors.

Whenever the operator wants to take over control the robot should react instantaneous. The same reasoning applies for the switch between approach and inspection phase. Whenever the robot is entering the inspection zone a hard switch to the corresponding controller is much better than a merge of both outputs. It might make sense to

envision a " return control zone " before which all sensors are either booted or turned off. For this project we have gone the way of keeping all sensors constantly. Furthermore, the all software modules concerned with feature detection and tracking are constantly running as well. This return control zone might be a way to optimize the robots battery usage, which will be a concern in future developments for industrial usage. However, by keeping the sensors constantly searching for cues the switch between approach and inspection phase might can happen much earlier. If a sufficient amount of cues is found before the GPS sensor determines that the inspection zone is reached, the supervisor is able to engage all modules for the inception phase which will lead to a smooth transition and also increases the robustness of the system.

Our experiments on the airport have shown that it is possible that the control with GPS might not be accurate enough to allow the robot to find all visual targets once the supervisor switches from approach to inspection mode. If however, we allow the robot to detect and track features during the approach phase, the chances of success increase dramatically.

2.4.2. Versatility of the navigation framework

During the course of the Air-Cobot project we have realized that there are constantly new ideas and techniques concerning the field of autonomous navigation. The latest release of ORB-SLAM is now able to deal with the scaling issue while still being constrained to just one camera. In order to prepare the framework to be operational in the future, it needed to be envisioned extremely modular and generic. Apart from techniques such as velocity maps, which help the robot deal with more than just one moving obstacle we have implemented approaches that help to build a metric costmap of the environment. These costmaps can then later be used for self localization or optimization of the mission path.

Incorporation and testing of a metric-based navigation approach to build a map of the environment During the experiments at the lab we investigated and perfected the versatility of our framework by making the software as modular and generic as possible. With the help of the ROS navigation stack, we started to incorporate techniques to build costmaps of the surrounding environment. A costmap is a metric representation of the environment which assigns costs to each of its atomic sections. These costs can then be used in finding a 'lowest price' for the navigation mission at hand.

(a) About 1 second into the experiment

(b) About 20 seconds into the experiment

(c) About 50 seconds into the experiment

(d) About 100 seconds into the experiment

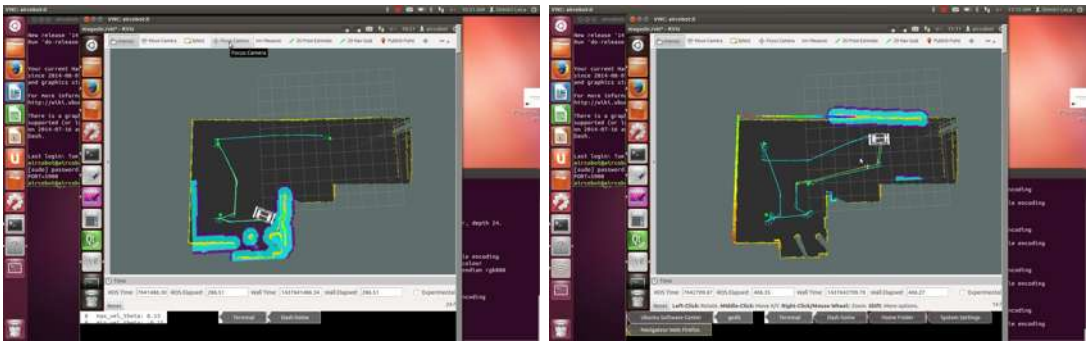
(e) About 210 seconds into the experiment

(f) About 270 seconds into the experiment

(g) About 310 seconds into the experiment

Figure 2.22.: Building costmaps with the help of the ROS navigation stack

In Figure 2.22, we display the outside vie of the experiment. The robot was set-up to only use the laser scanner as means of orientation. Furthermore, we used our framework



(a) Costmap of a prior experiment

(b) Costmap build during the experiment of Figure 2.22

Figure 2.23.: Costmaps build during the experiment

in order to provide the robot with some checkpoints. Figure 2.22 shows the costmap build during the experiment and another costmap that was acquired during a previous experiment in the same locations. As we can see in both maps, the planar Hokuyo laser has problems with clearly recognizing the ramp (multiple edges detected).

The raster (light gray squares) have the size of 1 square meter and the light blue area around the otherwise thin corners shows what the laser scanner is identifying at the moment of the shop. Another light blueish line shows the history of the path the robot has taken. The costmap classifies static objects such as walls, etc. as such. These objects make it into the metric map. Objects that move will also be recognized, but will not be saved onto the map. For the airport environment with its aircraft hangars and tarmac, having a metric map build and updated throughout the inspection missions executed by Air-Cobot can seriously improve its success rate.

2.5. Conclusion

The work presented in this chapter was mainly focused on the two first navigation modes (approach and inspection) that were presented in the first section of this chapter. More precisely, the chapter's main purpose was to provide an overview of how the navigational framework was designed in order to handle / support the strategies that will be presented in Chapter 3 and 4. Furthermore we have highlighted the robots capabilities and given an explanation for the favoritism of reactive as oppose to planning−based approaches. The modular approach for our framework allows a versatile basis for integration of new controllers. During our experiments we have also tested the introduction of a controller that allows for elliptic obstacle avoidance, a controller that combines the go to goal controller with velocity obstacles and so on.

Additionally we were able to show the versatility of framework by showing the incorporation of the ROS navigation stack to build a metric map of the environment. This technique will be the first step of creating meaningful costmaps which than later can

be used for obstacle avoidance in Chapter 4. Furthermore, it will allow the navigational framework to anticipate obstacle occlusions and thus avoid them or modify the topological map completely which will in turn then optimize the visual navigation.

Lastly it should be noted that the localization strategies will be the bases for the following chapters. Having a precise estimate of where the robot is located is fundamental to build a robust visual navigation system that can handle features loss, as well as allowing the obstacle avoidance to perform safely.

3. Multi Visual Servoing

Contents

3.1. Introduction

As mentioned earlier, the navigation strategy is composed of six subprocesses which are: perception, modeling, planning, localization, action and decision. In the following chapter, we will take a closer look towards the action process. As previously mentioned, it relies on several controllers allowing to reach a goal (using metric or visual information), avoid obstacles, re-orient the platform when needed, etc. In this chapter, we mainly focus on the first group of processes which allow the platform to reach a goal. Throughout the more than the three year duration of this project we have discovered that specifically during the second phase of our mission (autonomous inspection phase) visual sensory is what makes the difference between an acceptable and an exceptional localization. Thus, we can not stress enough how fundamental important visual sensory is for a successful mission. Indeed, it is used both to localize the robot relatively to the aircraft and to perform the **N**on-**D**estructive **T**esting (NDT) tasks. These latter rely on visual cues characterizing the specific aircraft items to be inspected. It is then only logical to also use these informations to control the platform with respect to the aircraft. This is the main reason why visual servoing controllers have been chosen to control the robot during the inspection phase. In addition, they allow to directly connect the processes of action and perception, especially in the case of the so-called 2D visual servoing where the visual features are used in the feedback without any processing step. It is worthwhile to mention that this last controller allows to reach a desired position with a high accuracy (a detailed explanation will follow), which will be of great interest to perform the NDT tasks. The visual servoing controllers are then at the core of this chapter.

We first propose a brief state of the art of this area, before stating our problem and focusing on our contributions. The closing section of this chapter will contain a summary of our presentation as well as a collection of future tasks.

3.1.1. State of the Art of Visual Servoing (VS)

The term visual servoing covers several techniques which aim to guide a robot solely on visual information provided by a camera sensor. The approach was first developed by Shirai and Inoue in 1973 [Shirai and Inoue, 1973] and, since then, many improvements were made [Malis et al., 1999], [Chaumette and Hutchinson, 2006], [Chaumette and Hutchinson, 2007]. Visual servoing has since become a mature technique which is widely used on many kinds of robots (manipulator arms, wheeled robots, UAVs, humanoids, etc.). Its field of application is extremely vast. For example there have been successful efforts to establish VS in the medical domain (automated surgery [Mebarki et al., 2010], micro-robotics [Tamadazte et al., 2012], etc.). As explained before, the key-idea of the approach is to directly connect vision to control. The control inputs of the robot are computed using visual features detected in the image, avoiding the metric localization step in a global map of the environment (in the navigation context).

Visual servoing techniques can be split into three sub-groups that are used by the

majority of the community. They are: **I**mage-**b**ased (or 2D) **v**isual **s**ervoing (IBVS), **p**osition-**b**ased (or 3D) **v**isual **s**ervoing (PBVS) and a slightly more recent hybrid approach, 2.5d visual servoing [Chaumette and Hutchinson, 2006], [Chaumette and Hutchinson, 2007], [Malis et al., 1999]. PBVS was the first developed control scheme. It relies on a set of 3D parameters characterizing the camera pose which must be estimated from the extracted image features [Chaumette and Hutchinson, 2006]. It is then necessary to know the intrinsic camera parameters and the 3D model of the observed object. In [Kermorgant and Chaumette, 2011], the authors have proven that this approach is to be globally asymptotically stable if the pose is assumed to be perfectly estimated. It allows smooth control of the 3D trajectory performed by the camera [Kermorgant and Chaumette, 2011]. However, it suffers from several drawbacks. First, as there is no real control in the image, the observed object may leave the camera field of view during the displacement. In addition, if the pose is not correctly estimated, the task can only be performed with reduced accuracy.

On the contrary, IBVS relies on the direct use of the visual features in the control loop. The camera pose is no more necessary, which allows to increase the precision of the task realization. However, its main drawback is that the induced 3D trajectory is not predictable and may be unsatisfactory or sometimes fairly non intuitive for a human observer. In addition, Jacobian singularities or local minima may exist [Chaumette, 1998]. To overcome these problems and try to benefit from both approaches respective advantages, an hybrid solution was developed, known as $2.5D$ visual servoing [Malis et al., 1999]. In this case, both $2D$ and $3D$ information are considered in the control law. Theoretically, this allows to derive an analytical proof of convergence and study the sensibility to calibration errors. Practically, it guarantees that the object reference point (usually the majority of its visual cues) are kept in the image, but the object may partially leave the camera field of view [Malis et al., 1999], [Kermorgant and Chaumette, 2011]. However, to deal with the visibility issue, other approaches have been developed in the literature. Some of them rely on a switch between different control laws depending on the risk of the visual features loss [Chesi et al., 2004], [Corke and Hutchinson, 2001], [Gans and Hutchinson, 2007], [Hafez and Jawahar, 2007], [Kermorgant and Chaumette, 2011]. Other solutions using optimal [Danès et al., 2010], or predictive [Allibert et al., 2010] control are also available. Finally, it is also possible to benefit from planning techniques which provide a path free of occlusions by using a metric model of the environment [Sharma and Sutanto, 1997], [Mezouar and Chaumette, 2002], [Kazemi et al., 2010].

However all these approaches aim to preserve the visibility of the necessary visual features at all costs. But, this strategy might not always lead to the best performances. Indeed, several phenomenons can lead to the lack of visual features: temporary camera breakdowns and sensor failures, image processing errors, landmark occlusions due to persons or vehicles moving through the camera field of view, large distance between the object and the sensor, unavailability of paths allowing to simultaneously reach the goal and preserve the visibility, etc. Thus, to guarantee the success of a visual servoing

task, it is necessary to develop a method managing the total loss of the visual signal. A first idea is to use tracking techniques such as [Jackson et al., 2004], [Comport et al., 2004], [Lepetit and Fua, 2006] and [Breitenstein et al., 2009]. In this case, the problem of the total occlusion of the interest landmark is treated by using measures extracted from the current image stream. This means that some information must be available to recover the missing informations. Thus, none of the above mentioned methods allow to deal with all the phenomenons leading to the visual features loss and especially when the image is no longer available. To be able to deal with the most general cases, it is mandatory to develop a technique which can be used without any current image. At LAAS − CNRS, such dedicated approaches have been developed both in mobile robotics [Folio, 2007, Petiteville, 2012, Petiteville et al., 2013] and in the dual arm manipulation context [Fleurmond, 2015]. They all share a common reasoning. The idea is a two-step algorithm where :

- while images are available, the method allows to estimate the depth of the visual features or more generally the 3D structure of the object of interest
- once the visual signal is lost, the visual features are reconstructed using the above 3D information.

The visual servoing control law is then computed using either the real or the estimated image cues depending on the context. The work proposed in this thesis is in the sequel of these approaches but has been adapted to the specificities of the Air-Cobot project.

3.1.2. Contributions

Before detailing our contribution, it is necessary to highlight some particularities of the Air-Cobot project. First of all, the platform must be precisely positioned at each checkpoint, which requires that a control law is designed which is capable to fulfill this requirement. In this context, following our previous analysis, IBVS appears to be an interesting solution. Second, most of the inspections will be performed in an outdoor environment. This means that a robust extraction of the visual features, although mandatory, may nonetheless be difficult to guarantee for several reasons: high variations in lighting conditions (luminosity issues), aircraft size, reflections on the fuselage, etc. In addition, the presence of the staff around the plane will increase the risks of occlusion and, as the environment is highly constrained, it is difficult to find a solution allowing to perform the task while preserving the visibility and the non collision. It then appears that a total visual features loss must be tolerated and not simply avoided. Finally, it is worthwhile to recall that a precise robot relative localization is provided by 3D matching of the robot environment, a CAD model of the aircraft and of the items to be inspected, and the constantly updated wheel odometry. This means that the necessary 3D information data to the visual features estimation are already available.

On the base of all previous remarks and analysis of visual servoing techniques, we propose the following approach. Since the main problem originates from the possibility

of total visual signal losses which may occur at any time during the mission we propose: The key-idea to tackle this issue is to define an augmented (or virtual) image which will be built by adding measured 2D visual cues together with estimated ones.

In this way, an " image " is always available, whether the features are visible or not. Hence, it is then always possible to control the robot using vision.

3.1.3. Outline of the chapter

In the upcoming sections, we will present the two control techniques at the core of this chapter, namely IBVS and PBVS. We will highlight the advantages and drawbacks of each method, demonstrating the interest of their coupling. Furthermore, we will show the interest of using the augmented image. Throughout this section, simulations conducted in MATLAB and experiments on Air-Cobot will help to highlight and validate several conclusions of our research.

3.2. Prerequisites of Visual Servoing

In the following section we will introduce all necessary information that is essential to clearly highlight our contribution. We will introduce the robot and all the sensory that is needed to perform a visual servoing task. Additionally information about the available coordinate systems will be shared. In order to provide a better understanding of the strategy behind our approach we will deduce the robot Jacobian before introducing combination and interaction matrices.

3.2.1. Robot presentation

Before we will go into further details about the visual servoing algorithms, a short description of the robot and its coordinate frames will be given. Therefore, we will focus mainly on geometry and frames that are necessary in order to move the camera and the robot body in the correct way while applying the visual servoing technique. The inputs of visual servoing to control the robot are the following :

$$\dot{q} = \begin{pmatrix} \nu_{robot} \\ \omega_{robot} \\ \omega^{cam_{pan}} \\ \omega^{cam_{tilt}} \end{pmatrix} \tag{3.1}$$

where v represents the linear velocity of the robot and ω its angular velocity. $\omega^{cam_{pan}}$ (respectively $\omega^{cam_{tilt}}$) is the pan (respectively tilt) angular velocity from the **P**an − **T**ilt − **U**nit (PTU).

The following section will explain in more details the frames which have been introduced.

3.2.2. Robot coordinates system

Figure 3.1 displays the four frames used on the robot for the implementation of visual servoing and Figure 3.2 present the conventions of each frames These conventions are taken from the Lagadic open source software 'VISP'[1] [web,].

Figure 3.1.: Air-Cobot and the associated frames [Demissy et al., 2016]

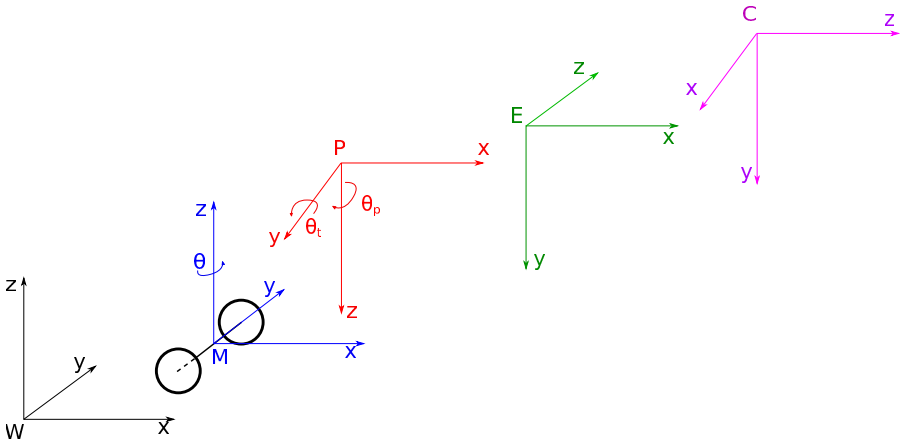


Figure 3.2.: Robot coordinate system for the front cameras [Demissy et al., 2016]

The first frame is the world frame (in black) which is static and used as a reference. Its origin point is noted W. The second coordinate frame, denoted R_m, is the robot frame (in blue). Its origin point is M. It turns around the z axis with an angle θ. The

[1] ViSP referring to the Visual Servoing Platform is a modular cross platform library that allows prototyping and developing applications using visual tracking and visual servoing techniques at the heart of the researches done by Inria Lagadic team.

third coordinate frame, denoted R_p, is the Pan-Tilt-Unit frame (in red). Its origin point is noted P and represents the center of the PTU. Therefore the distance between M and P is a fixed constant. After rotation with an angle of π around the axis x, the blue frame results in the red frame (given the translation between M and P). Physically, this third frame in our presentation (red) is able to rotate around its z axis with a $\theta^{cam_{pan}}$ angle (equivalent to yaw rotation) and can rotate around its y axis with a $\theta^{cam_{tilt}}$ angle (equivalent to pitch rotation). The fourth frame, denoted R_e, is the end effector frame (in green). This frame was solely used to decompose the rotation between the PTU and camera. Its origin point is noted E. The distance between P and E is fixed. A second difference between the end-effector frame and the PTU frame is a $\frac{\pi}{2}$ rotation around the x axis. The fifth frame, denoted R_c, is the camera frame (in purple). Its origin point is noted C and corresponds to the center of the camera and the distance between E and C is fixed. Another difference between the green frame and the purple one is a $\frac{\pi}{2}$ rotation around the y axis.

These frames are presented in much more detail in the appendix starting on Page 166.

3.2.3. Robot Jacobian and Velocity Twist Matrix

The Robot Jacobian J provides the relationship between the kinematic screw of the camera $T^{R_C}_{C/R_W}$ and the control vector $\dot{q}$. A complete derivation of this equation can be found in the appendix in Section A.2 on Page 171 and its application to Air-Cobot on Page 179. The result of the derivation for the front cameras is displayed in Equation (3.2).

$$
\begin{aligned}
T^{R_C}_{C/R_W} &= \begin{pmatrix}
-s_p & y_P\, s_p - x_P\, c_p - (x_E + x_C)\, c_t + (z_E + y_C)\, s_t & x_E + x_C & 0 \\
-c_p\, s_t & y_P\, c_p\, s_t + x_P\, s_p\, s_t + (z_C - y_E)\, s_t & 0 & x_E + x_C \\
c_p\, c_t & -y_P\, c_p\, c_t - x_P\, s_p\, c_t - (z_C - y_E)\, c_t & -y_E + z_C & -z_E - y_C \\
0 & 0 & 0 & -1 \\
0 & -c_t & 1 & 0 \\
0 & -s_t & 0 & 0
\end{pmatrix} \times \dot{q} \\
&= J \times \dot{q}
\end{aligned}
\tag{3.2}
$$

where J is the robot Jacobian which expresses the relationship of the robot control inputs and the camera motion. It is interesting enough to mention that by removing columns from that matrix we can simply deny certain degrees of freedom of the system. For example, if the robot shall not move (neither in a linear or angular notion) the first two columns can be removed. Of course the same has to be done for $\dot{q}$ which is formerly a 4 by 1 matrix.

3.3. Image-Based Visual Servoing

Image **b**ased **V**isual **S**ervoing will be the first technique we will describe in detail in this chapter. It has the undeniable advantage of being able to use information (about each features position on the camera plane) that is taken directly from the sensor. No geometrical interpretation of the image is needed. However, the features need to be detected and / or tracked during the mission. A more detailed explanation on this process will be given in one of the subsequent sections of this chapter.

3.3.1. Method

In order to properly describe the basis of the IBVS method we will explain the control law and how to built the interaction matrix.

Control law

Figure 3.4 provides some insight towards the underlying control system analysis background of the method. The start is given by a desired representation of the features in the image (s^*). This can be done by either providing the vector explicitly, or by defining a desired robot pose and projecting the target in the camera image, or by learning s^* when the robot is manually put in the desired position. s^* is then compared with the current feature vector (s) and the control loop error (e_I) is formed. Once the control loop error has been determined, it can be used to generate the control inputs using the controller. With the resulting $\dot{q}$, the robot can be controlled so that e_I progressively diminishes and the desired image is reached. Equation (3.10) is presenting these relationships. Here we can see that the controller is working in the feature space, whereas position based visual servoing is expressed in the Cartesian space [Hutchinson et al., 1996b].

$$e_I = s - s^* \tag{3.3}$$

where s^* contains the n desired visual features.

To make the error e_I decrease exponentially, we impose:

$$\dot{e_I} = -\lambda_I \times e_I \tag{3.4}$$

$$\text{with } \lambda_I \in \mathbb{R}^{+*} \tag{3.5}$$

An exponential decrease must lead (with $t \to \infty$) to an e_I of zero and in an ideal case one should be able to observe a solely negative gradient throughout the experiment if no physical constraints (velocity, acceleration limits etc.) are set for the robotic platform. Knowing that:

$$\dot{e_I} = \dot{s} - \dot{s^*} \tag{3.6}$$

and recalling that $\dot{s^*}$ is constant, we get:

$$\dot{e_I} = \dot{s} \tag{3.7}$$

$$\dot{e_I} = L_I \times T^{R_C}_{C/R_W} \tag{3.8}$$

where L_I is the interaction matrix. With this matrix we are able to relate the evolution of the visual features in the image to the camera kinematic screw $T^{R_C}_{C/R_W}$ defined in Equation (3.2). L_I depends on the visual features and is defined in the following paragraph. Additionally, as $T^{R_C}_{C/R_W} = J \times \dot{q}$ (see Equation (3.2)), $\dot{e_I}$ expresses as:

$$\dot{e_I} = (L_I \times J) \times \dot{q} \tag{3.9}$$

The control law is consequently the following :

$$\dot{q} = -\lambda_I \times (L_I \times J)^+ \times e_I \tag{3.10}$$

where $(L_I \times J)^+$ being the Moore-Penrose pseudo-inverse of the matrix product $(L_I \times J)$.

Interaction matrix L_I

The interaction matrix relates the evolution of the visual features in the image to the camera kinematic screw. It depends on the visual features type. Here we will consider that our object of interest is made of points. In order to determine the interaction matrix L_I, the coordinates of each 3D point has to be expressed in the image-plane frame, as displayed in Figure 3.3.

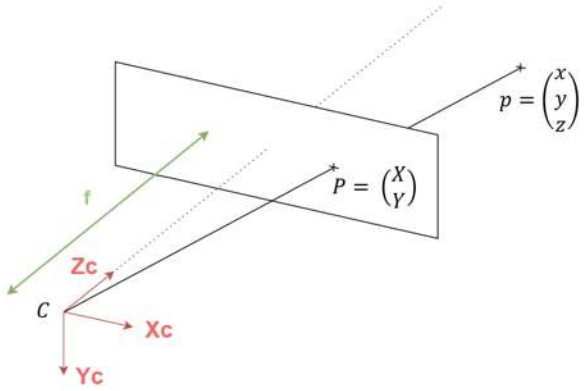


Figure 3.3.: Projection in the image-plane frame

In Figure 3.3, $p = (x, y, z)$ refers to the $3D$ point expressed in the camera frame, and $P = (X, Y)$ stands for the image point metric coordinates. Regarding a specific static point P_i of the object, according to Thales, the relation between the $3D$ and $2D$ coordinates is :

$$\begin{cases} X_i = f\,\dfrac{x_i}{z_i} \\ Y_i = f\,\dfrac{y_i}{z_i} \end{cases} \tag{3.11}$$

where f is the focal length of the camera.

L_I links the camera kinematic screw and the derivative of s according to Equation (3.8). With the derivation of (X_i, Y_i), we find:

$$\begin{cases} \dot{X}_i = f\,\dfrac{\dot{x}_i z_i - \dot{z}_i x_i}{{z_i}^2} \\ \dot{Y}_i = f\,\dfrac{\dot{y}_i z_i - \dot{z}_i y_i}{{z_i}^2} \end{cases} \tag{3.12}$$

From this, it follows that:

$$\begin{aligned} \begin{pmatrix} \dot{X}_i \\ \dot{Y}_i \end{pmatrix} &= \begin{pmatrix} \frac{f}{z_i} & 0 & \frac{-X_i}{z_i} \\ 0 & \frac{f}{z_i} & \frac{-Y_i}{z_i} \end{pmatrix} \times \begin{pmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{z}_i \end{pmatrix} \\ &= L_1 \times V_{P/R_C}^{R_C} \end{aligned} \tag{3.13}$$

Moreover :

$$V_{P_i/R_W}^{R_C} = V_{C/R_W}^{R_C} + \Omega_{R_C/R_W} \wedge {CP_i}^{R_C} + V_{P_i/R_C}^{R_C} \tag{3.14}$$

Since the transformation between P_i and W is rigid, $V_{P_i/R_W}^{R_C} = 0$.

$$\begin{aligned} V_{P_i/R_C}^{R_C} &= -V_{C/R_W}^{R_C} - \Omega_{R_C/R_W} \wedge {CP_i}^{R_C} \\ &= -V_{C/R_W}^{R_C} + \hat{CP_i}^{R_C}\,\Omega_{R_C/R_W} \\ &= \begin{pmatrix} -I_3 & \hat{CP_i}^{R_C} \end{pmatrix} \times T_{C/R_W}^{R_C} \\ &= L_2 \times T_{C/R_W}^{R_C} \end{aligned} \tag{3.15}$$

where $\hat{CP_i}^{R_C}$ is the skew-symmetric matrix associated to vector $CP_i^{R_C}$. Finally we have :

$$\begin{aligned} \dot{s}_i = \begin{pmatrix} \dot{X}_i \\ \dot{Y}_i \end{pmatrix} &= L_1\, L_2\, T_{C/R_W}^{R_C} \\ &= L_{P_i}\, T_{C/R_W}^{R_C} \end{aligned} \tag{3.16}$$

Where

$$L_{P_i} = \begin{pmatrix} \dfrac{-f}{z_i} & 0 & \dfrac{X_i}{z_i} & \dfrac{X_i Y_i}{f} & -(f + \dfrac{X_i^2}{f}) & Y_i \\ 0 & \dfrac{-f}{z_i} & \dfrac{Y_i}{z_i} & f + \dfrac{Y_i^2}{f} & \dfrac{-X_i Y_i}{f} & -X_i \end{pmatrix} \tag{3.17}$$

Finally, for n feature points, the interaction matrix L_I is :

$$L_I = \begin{pmatrix} L_{P_1} \\ L_{P_2} \\ L_{P_3} \\ ... \\ L_{P_n} \end{pmatrix} \tag{3.18}$$

Conclusion

Image-based visual servoing is a control method which allows to create a direct link between image processing and robot control. Once the target is identified (and can be tracked), IBVS will calculate the correct commands to move the sensor to minimize e_I. However, IBVS will not make sure that the target is kept in the camera **F**ield **O**f **V**iew (FOV). On the other hand, IBVS can lead to unfavorable or unexpected motions of the robot (advance retreat problem, singularities, local minima) [G. Palmieri and Callegari, 2012] [Chaumette, 1998], [Chaumette and Hutchinson, 2006] and [Cai et al., 2014].

3.3.2. Simulation

The following section will provide some results obtained in MATLAB simulations.

Modeling

The IBVS method is modeled as a block diagram in Figure 3.4 for simulation purposes. We will present our target definition, how the feature vector is calculated and we will explain how we have calculated reactions of our robotic platform. For obvious reasons, certain simplifications were made in order to reduce the complexity of the simulations. However, this does not change the validity of the later comparison of both visual servoing techniques.

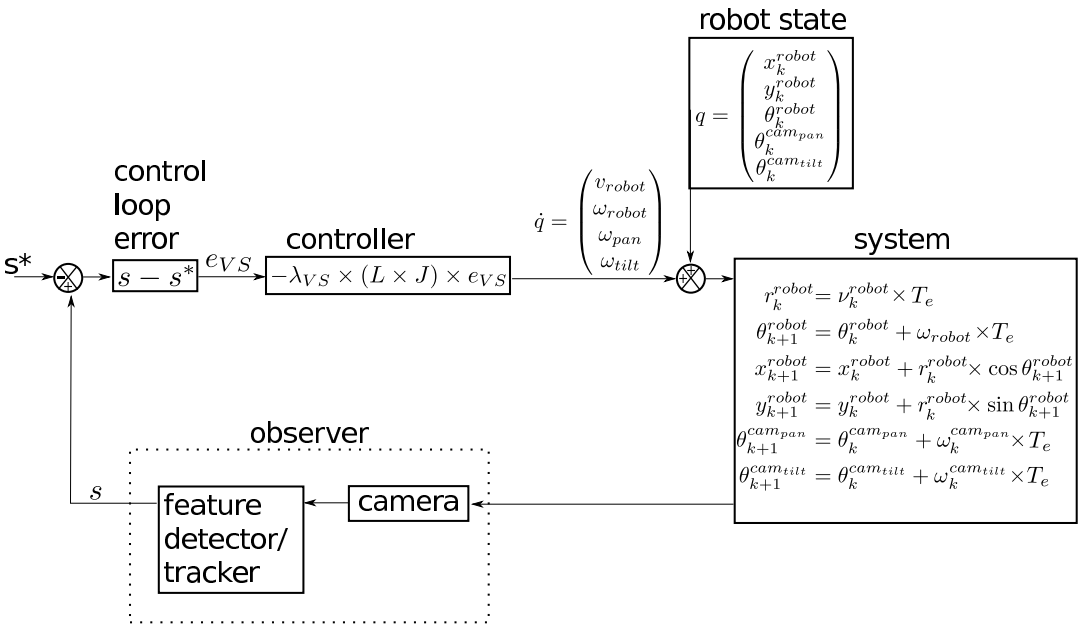


Figure 3.4.: Control loop of an Image-based visual servoing example

The simulation approach involves the definition of the target, the calculation of the image features vector s given by the camera and the robot equations implementation.

Target definition For the following simulations, the target has been chosen as a set of four features points for the sake of robustness (see Figure 3.5). These four points define a ten centimeter square. The position of its center of gravity in the world frame is $(1000, 0, 715)\,[mm]$ where 715 stands for the height of the camera in standard position ($\theta^{cam_{pan}}$ and $\theta^{cam_{tilt}}$ are equal to zero).

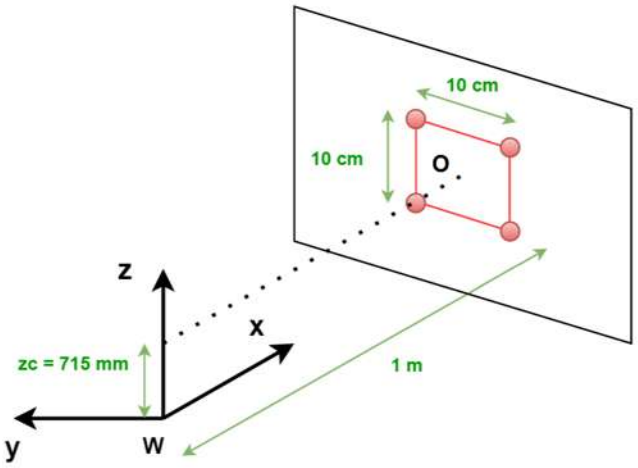


Figure 3.5.: Target chosen for simulations

Calculation of the image features s by the camera The feature detection algortihm provides us with (X_i, Y_i), the coordinates of point P_i which is the projection on the focal plane of the point $p_i(x, y, z)$ following Equations (3.11) and knowing z_i value. As stated beforehand, knowing the camera intrinsic and extrinsic parameters is essential to complete the visual servoing mission.

Reaction of the robotic platform The following equations will simulate the response of the physical system. As we have hinted on from the beginning, the basic approach can be understood as a speed control technique. Since our calculations take place in the digital domain, all signals and calculations cannot be continuous. From a mathematical point of view, it is defined by a discrete integral calculation on the input control vector $q = (x,\ y,\ \theta,\ \theta^{cam_{pan}},\ \theta^{cam_{tilt}})^T)$ where x, y are the coordinates of the robot position, θ is the robot orientation and θ_p, θ_t are the angles of the PTU.

For this purpose, Euler's method is used between k and $k+1$ sample as follows :

$$
\begin{aligned}
r_k^{robot} &= \nu_k^{robot} \times T_e && (3.19)\\
\theta_{k+1}^{robot} &= \theta_k^{robot} + T_e \times \omega_k && (3.20)\\
x_{k+1}^{robot} &= x_k^{robot} + r_k \times \cos(\theta(k+1)) && (3.21)\\
y_{k+1}^{robot} &= y_k^{robot} + r_k \times \sin(\theta(k+1)) && (3.22)\\
\theta_{k+1}^{cam_{pan}} &= \theta_k^{cam_{pan}} + T_e \times \omega_k^{cam_{pan}} && (3.23)\\
\theta_{k+1}^{cam_{tilt}} &= \theta_k^{cam_{tilt}} + T_e \times \omega_k^{cam_{tilt}} && (3.24)
\end{aligned}
$$

where T_e is the sampling period, r is the distance traveled by the robot, $\omega_k^{cam_{pan}}$ is the pan angular velocity, $\omega_k^{cam_{tilt}}$ is tilt angular velocity.

Controller equations The controller block computes the interaction matrix L_I for each feature point in the image plane following Equation (3.18). As we recall, a total of four points are considered and since every feature has two coordinates in the image-plane, the interaction matrix L_I will be a matrix of size 8×6 ($2n \times 6$; n being the number of features and six corresponds to the degrees of freedom of the whole system).

The Jacobian matrix, J, is calculated for the current configuration of the robot with the aid of Equation (3.2). It is a 6×4 matrix, where four corresponds to the number of control inputs, $\dot{q}$. If you recall, the system is able to drive linearly, turn around its z axis, pan and tilt the camera.

Once L_I and J are determined, the Moore-Penrose pseudo inverse matrix of $(L_I\, J)$ is evaluated. This evaluation will return the control vector $\dot{q}$ according to Equation (3.10).

Simulation results

Simulation conditions For all simulations, the underlying geometry will be the one of the Air-Cobot platform with all its degrees of freedom (DOF) for the cameras and all constraints. This is necessary to relate all of our findings onto the robotic platform.

Something we will simplify in order to avoid too complex simulations are physical constraints of the electric motors such as speed, acceleration and jerk limits. These physical quantities will be neglected. This is important to remember since on the real life robotic platform these limits need to be kept.

The simulation conditions are as follows :

- The desired pose is the simplest one, all the angles from the camera degrees of freedom ($\theta^{cam_{pan}}$, $\theta^{cam_{tilt}}$) and also θ are equal to zero.
- The position of the camera (point C) should be in the center of the world frame so the point M (see Figure 3.2) is at $(0, 150, z_M)_{R_W}$, because $MP \times y_W = -150$ mm.
- The initial pose is the same as the desired one, except that the robot is 3 meters back on the x axis.
- The simulation has been executed with a visual servoing gain, λ_I, of 1 (static gain) and the sampling period $T_e = 1e^{-3}\,s$ (i.e sampling rate of 1000 Hz). Note that this sampling rate has been chosen sufficiently small to be able to use the Euler's scheme to estimate properly the robot motion. It is possible here as the camera function is modeled using only the pinhole model. On the real Air-Cobot platform, a sampling rate of $\sim$ 25 Hz has been proven to be sufficient and the maximal control input for the CAN-BUS of the robot is limited to 50 Hz.

Table 3.1 gathers the general conditions :

	$X_M[mm]$	$Y_M[mm]$	$\theta[rad]$	$\theta_p[rad]$	$\theta_t[rad]$
Desired pose	0	150	0	0	0
Initial pose	-3000	150	0	$\frac{\pi}{4}$	$\frac{\pi}{6}$
Final pose	0.24	154.8	-0.0082	-0.006	~ 0

Table 3.1.: Parameters of the IBVS simulation

The two first lines of Table 3.1 provides the robot initial and desired poses values.

The obtained results Table 3.1 shows that during the first simulation the desired pose was reached precisely enough (less than 5 mm in y, far less than 1 mm in x and less than 1° for θ, θ_p and θ_t camera pan and tilt).

Figure 3.6 shows how the features evolve during the execution of the simulation. A continuous movement from the initial position of the features in the camera frame to the final one can be observed. This matches the results shown in the last row of Table 3.1.

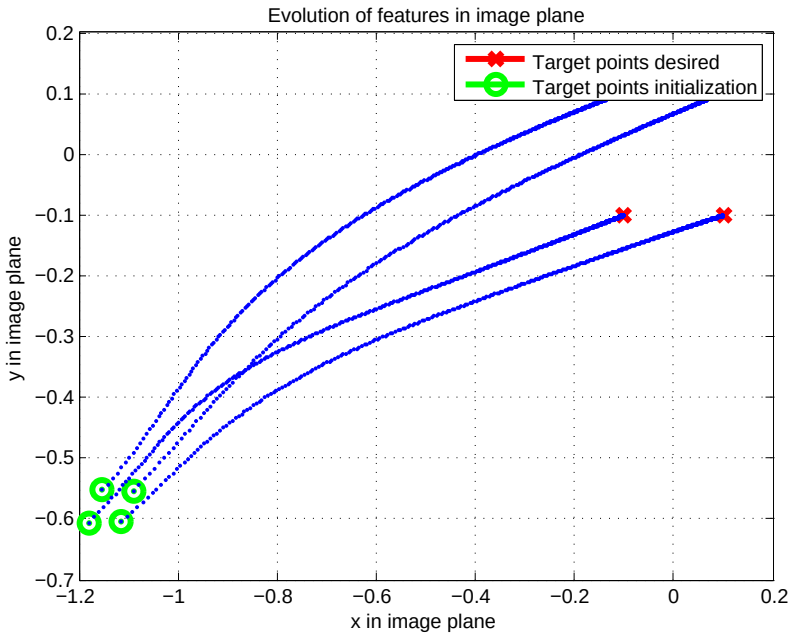


Figure 3.6.: Evolution of features in an IBVS simulation

Figure 3.7 displays the robot state vector ($q = (x,\ y,\ \theta,\ \theta^{cam_{pan}},\ \theta^{cam_{tilt}})^T$) throughout the simulation. Due to the initial pan angle, the system compensates (and therefore slightly drifts) on the y-axis. This explains the low error at its final position of $\ll 10$ mm.

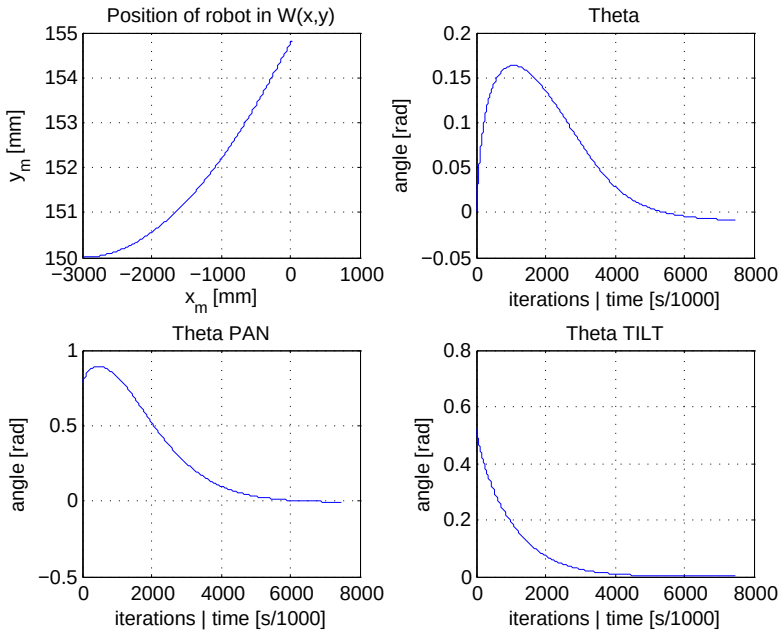


Figure 3.7.: Robot path, orientation and camera state in a IBVS simulation

Figure 3.8 highlights the error, e_I, in the simulation. As one can see, it progressively decreases and vanishes after approximately five seconds. This evolution may help to determine the convergence of the algorithm and to define a convenient threshold to stop the simulation. Such a threshold is necessary since the gain is static during these simulations. Due to the exponential decrease of the error, the control inputs to the robotic platform will become 'weaker' towards the end of the simulation. In order to find an optimal point to stop and start the post-processing of our results, a threshold for the error is chosen.

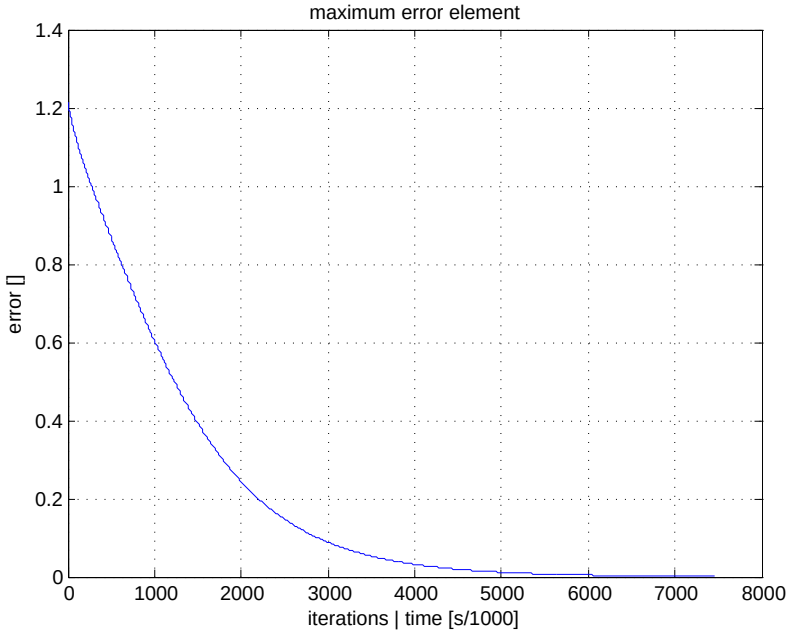


Figure 3.8.: Evolution of the visual error in a IBVS simulation

The last figure, Figure 3.9, shows the evolution of the control inputs, $\dot{q}$. As mentioned before, no velocity or acceleration limits will be respected in order to achieve an ideal experiment. Later, on the Air-Cobot platform however, maximum velocities such as $\sim 10000\frac{mm}{s}$ or $\sim 10\frac{m}{s}$ can surely not be reached and must therefore be avoided. However, these velocities can easily be controlled by choosing a lower λ_I and are not that far of from the reality (Air-Cobot can reach up to 5 $\frac{m}{s}$).

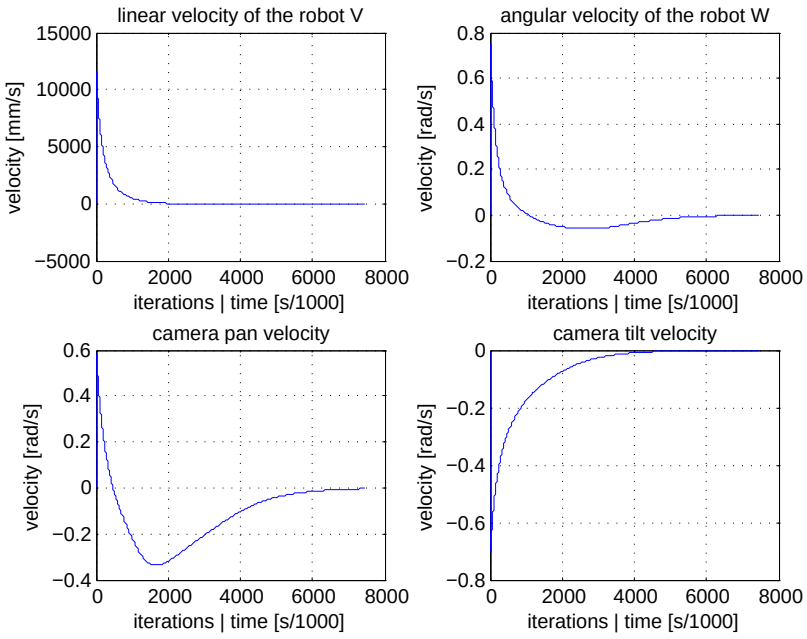


Figure 3.9.: Evolution of velocities in a IBVS simulation

3.4. Position-Based Visual Servoing

Contrary to IBVS, **P**osition **B**ased **V**isual **S**ervoing does require a geometrical interpretation of the taken image before an effective control law can be computed. Therefore, a relative pose estimation between camera and target is crucial. One of the interests of this approach lies in the chance of defining tasks in a standard Cartesian frame [G. Palmieri and Callegari, 2012], which allows to obtain nice robot trajectories. A major downside to this is that the control law now depends on the camera pose, which makes the approach sensitive to errors (calibration, etc.).

3.4.1. Method

Control law

In PBVS, our servoed variable χ is now defined in the following 3D terms :

- The 3D coordinates of the object centroid O expressed in the camera frame ;
- The camera orientation with respect to a frame linked to the object and centered in O. This orientation is expressed thanks to the angle/axis $(\alpha\, r)$ representation. In this representation, the axis r stands for the vector around which one must turn with an angle α in order to go from the current camera frame to the object frame. Beforehand, the desired camera frame has the same orientation as the object frame.

Thus the PBVS error, e_P, is defined as below :

$$\begin{aligned} e_P &= \chi - \chi^* \\ &= \begin{pmatrix} {}^C t_O{}^{R_C} \\ \alpha\, r \end{pmatrix} - \begin{pmatrix} {}^{C^*} t_O{}^{R_{C^*}} \\ 0 \end{pmatrix} \end{aligned} \tag{3.25}$$

where ${}^C t_O{}^{R_C}$ is the translation from C (origin of camera frame) to O expressed in the camera frame. As well as IBVS, our control vector $\dot{q}$ is designed so that the error decreases exponentially, i.e. :

$$\dot{e}_P = -\lambda_P\, e_P \tag{3.26}$$

From this, we get:

$$\begin{aligned} \dot{e_P} &= \dot{\chi} - \dot{\chi^*} \\ &= \dot{\chi} \\ &= L_{PBVS}\, T^{R_C}_{C/R_W} \end{aligned} \tag{3.27}$$

Moreover :

$$T^{R_C}_{C/R_W} = J\, \dot{q} \tag{3.28}$$

The control law is unchanged :

$$\dot{q} = -\lambda_P\, (L_{PBVS} J)^+\, e_P \tag{3.29}$$

As J only depends on the robot, it is not modified. On the contrary, L_{PBVS} depends on the servoed variable.

Interaction matrix L_{PBVS}

L_{PBVS} links the camera kinematic screw and the derivative of χ. Let us express the derivation of $({}^C t_O{}^{R_C}, \alpha\, r)$.

$$\begin{aligned} \frac{dCO}{dt}^{R_C}_{/R_W} &= \frac{dCW}{dt}^{R_C}_{/R_W} + \frac{dWO}{dt}^{R_C}_{/R_W} \\ &= -\frac{dWC}{dt}^{R_C}_{/R_W} \end{aligned} \tag{3.30}$$

Recalling that the transformation between O and W is rigid, we get:

$$\begin{aligned} \frac{dCO}{dt}^{R_C}_{/R_W} &= \frac{dCO}{dt}^{R_C}_{/R_C} + CO^{R_C} \wedge \Omega^{R_C}_{R_W/R_C} \\ &= \frac{dCO}{dt}^{R_C}_{/R_C} - \hat{CO}^{R_C}\, \Omega^{R_C}_{R_C/R_W} \end{aligned} \tag{3.31}$$

where $\hat{CO}^{R_C}$ is the skew-symmetric matrix associated to vector CO^{R_C} . Finally we have:

$$\begin{aligned} \frac{dCO}{dt}^{R_C}_{/R_C} &= -V^{R_C}_{C/R_W} + \hat{CO}^{R_C}\, \Omega^{R_C}_{R_C/R_W} \\ &= \begin{pmatrix} -I_3 & \hat{CO}^{R_C} \end{pmatrix} \times T^{R_C}_{C/R_W} \end{aligned} \tag{3.32}$$

As for the angle/axis parameters, according to Ezio Malis' thesis [Malis, 2008]:

$$\begin{aligned} \frac{d(\alpha\, r)}{dt} &= \frac{d\alpha}{dt}\, r + \alpha\, \frac{dr}{dt} \\ &= L_{\alpha\, r} \times \Omega^{R_C}_{R_C/R_W} \end{aligned} \tag{3.33}$$

Where :

$$L_{\alpha\, r} = I_3 - \frac{\alpha}{2}\, \hat{r} + (1 - \frac{sinc\alpha}{sinc^2 \frac{\alpha}{2}})\, \hat{r}^2 \tag{3.34}$$

Finally :

$$L_{PBVS} = \begin{pmatrix} -I_3 & \hat{CO}^{R_C} \\ 0 & L_{\alpha\, r} \end{pmatrix} \tag{3.35}$$

Conclusion

The PBVS method works in the same way as IBVS method. Indeed the controller, robot and camera are built similarly to IBVS (see subsection 3.3). The main differences are the centroid determination and the pose calculation explained in the following section.

3.4.2. Simulation

We now explain how the position-based visual servoing method has been simulated.

Modeling

As for the IBVS, the PBVS method was modeled as a block diagram displayed in Figure 3.10.

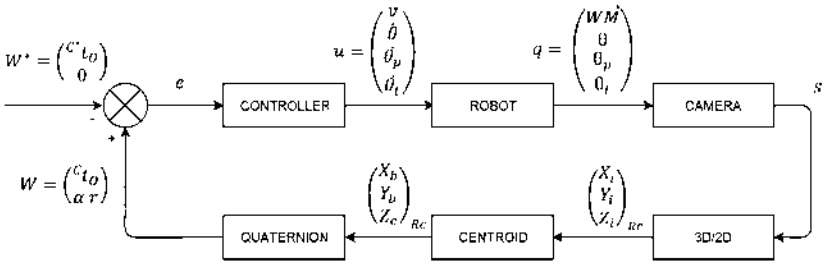


Figure 3.10.: Block diagram for the PBVS approach

As pointed out earlier, the main differences with IBVS concerns the definition of the servoed variable and therefore its computation.

The centroid calculation O From the previous results, we compute the centroid O of the target points whose coordinates (x_i, y_i, z_i) are given in the target frame. The coordinates of O are first expressed in the target frame before being converted into the camera frame. To do so, a pose calculation is made. It consists in projecting of the coordinates of one point in the camera frame from the coordinates of this point in the image plane thanks to Equation (3.11). Apart from knowing the distance between this point and the camera information about the intrinsic parameters of the camera (mainly the focal length of the camera) is essential.

The computation of the servoed variable The servoed variable is computed within the quaternion function. We calculate :

- The distance between the camera and the object centroid which has been obtained at the previous step;
- The quaternions which represents the orientation of the camera frame with respect to the object frame, that is the coordinates of the rotation axis r and the corresponding angle of rotation α.

Unit quaternions provide a convenient mathematical notation for representing orientations and rotations of objects in three dimensions. Compared to Euler angles they are simpler to compose and avoid the problem of gimbal lock. Unlike rotation matrices, they are numerically stable and allow much to perform computations much more efficient. Quaternions have found their way into applications in computer graphics, computer vision, robotics, navigation, molecular dynamics, flight dynamics, orbital mechanics of satellites and crystallographic texture analysis. Unfortunately, by using quaternions the operator looses the high intuitivity that is provided by Euler angels and homogeneous matrices. When used to represent a rotation, unit quaternions are also called rotation quaternions. In the context of orientation (rotation relative to a reference coordinate system), they are called orientation quaternions or attitude quaternions.
For their computation, formulas of article [Kelly, 2013] were used.

Simulation results

Simulation conditions: For the PBVS simulation, four features points were chosen. Having four individual features is not necessary, but increases the robustness and even thought this might not be necessary for a simulated test, it is much closer to the set-up that will be used on the real robot. From them, as shown above, we compute the center of gravity for all points. We will show later that this is particularly helpful when dealing with noise during the feature acquisition. The visual servoing gain λ has been fixed to 1, the same value which has been used in the image-based visual servoing experiment. It will ease the comparison. The two first lines of Table 3.2 provide the initial and the final poses of the robot.

	$X_M[mm]$	$Y_M[mm]$	$\theta[rad]$	$\theta_p[rad]$	$\theta_t[rad]$
Desired pose	-300	350	0	$-\frac{\pi}{4}$	$\frac{\pi}{6}$
Initial pose	-8000	300	$\frac{\pi}{2}$	$\frac{\pi}{4}$	$\frac{\pi}{10}$
Final pose ($\sim$ 22k iterations)	-288.2664	370.6035	-0.0097	$-\frac{\pi}{4.2}$	$\frac{\pi}{6.1}$

Table 3.2.: Parameters of the PBVS simulation

Obtained results Compared to the experiments for IBVS in Section 3.3.2 it stands out that the distance the robot needs to travel has been increased. Thus, resulting in an absolute distantial error towards the desired position that is higher than in the previous simulation. This can be assumed comparing the desired and the initial positions given in Table 3.2. However considering the distance traveled ($\sim$ 7702 mm) the error ($\sim$ 24 mm) is comparably small ($\sim$ 0.3%). And can be further decreased by more iterations (longer simulation time) or increasing λ.

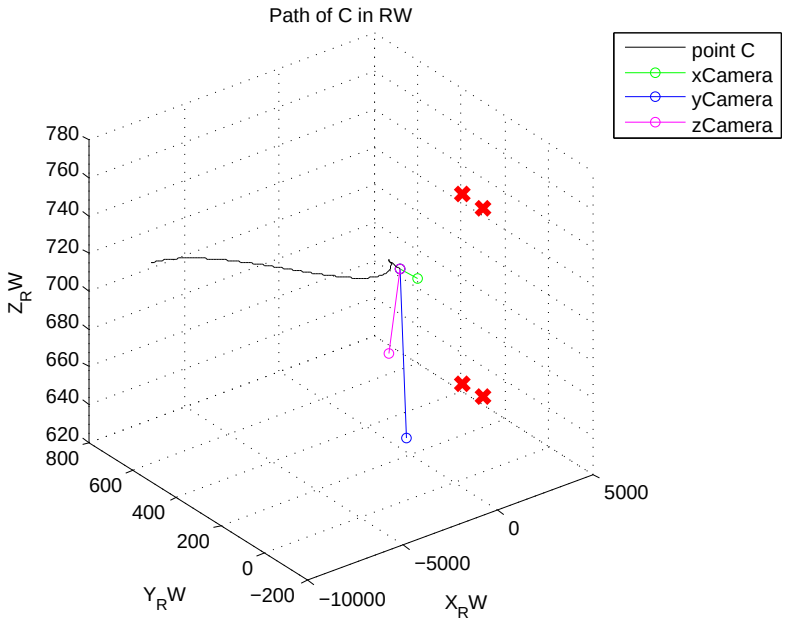


Figure 3.11.: Evolution the point C (camera origin) in a PBVS simulation

As before, the three following figures highlight the movement of the robot (Figure 3.12), the evolution of the targeted features in the camera frame (Figure 3.12) and of the error (Figure 3.14). Looking at the feature evolution and at the error, one can notice that the desired objective is almost perfectly reached. If the simulation would have been continued for another ten seconds (10k iterations) or λ_P would have been increased towards the end, Figure 3.12 would have shown all four feature lines (blue) connected to their respective positions in the image frame (red crosses). Figure 3.12

show that the robot reaches the desired pose indicated in Table 3.2. All these figures are then consistent.

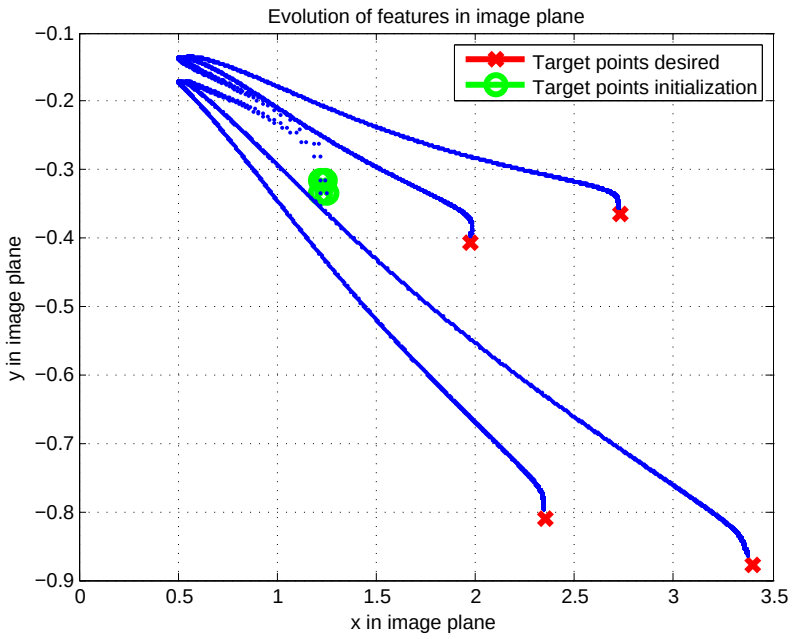


Figure 3.12.: Evolution of features in a PBVS simulation

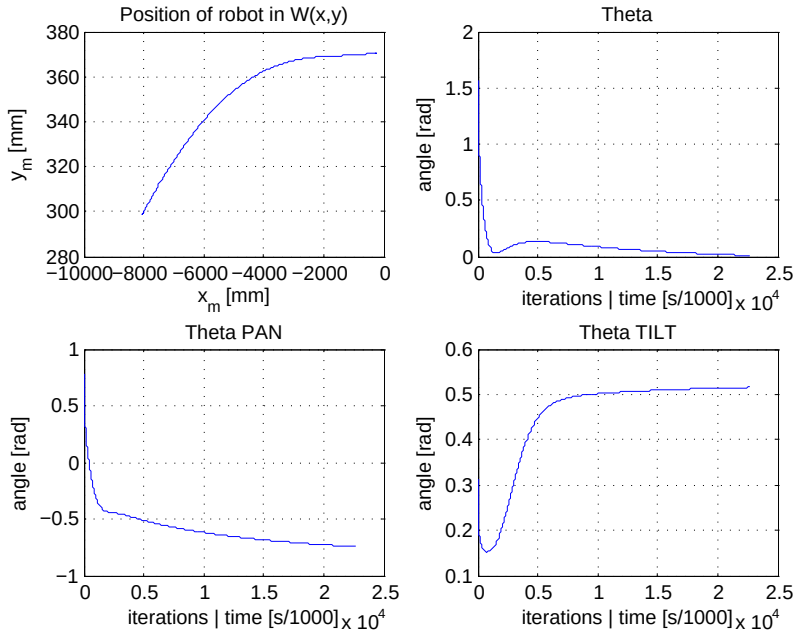


Figure 3.13.: Robot path, orientation and camera state in a PBVS simulation

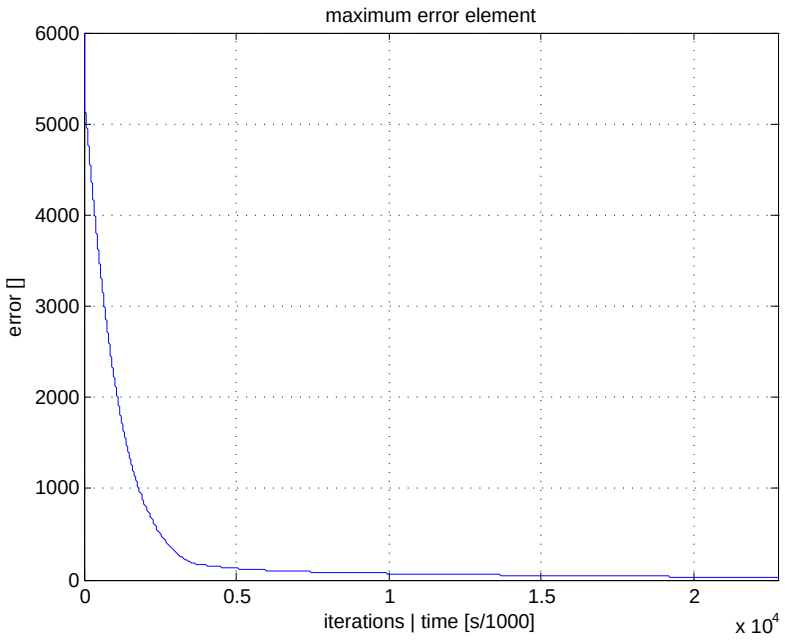


Figure 3.14.: Evolution of error in a PBVS simulation

We have also plotted the velocities applied to the robot in Figure 3.15. High velocities are produced at the beginning of the task because of the corresponding high values of the error. A smaller gain λ_P would allow to limit the problem, but it would have also reduced the convergence rate of the error. The choice of the gain might then be difficult when the robot starts far from the desired goal and that the corresponding error is large. An adaptive gain can then be of interest. The same remark holds for IBVS.

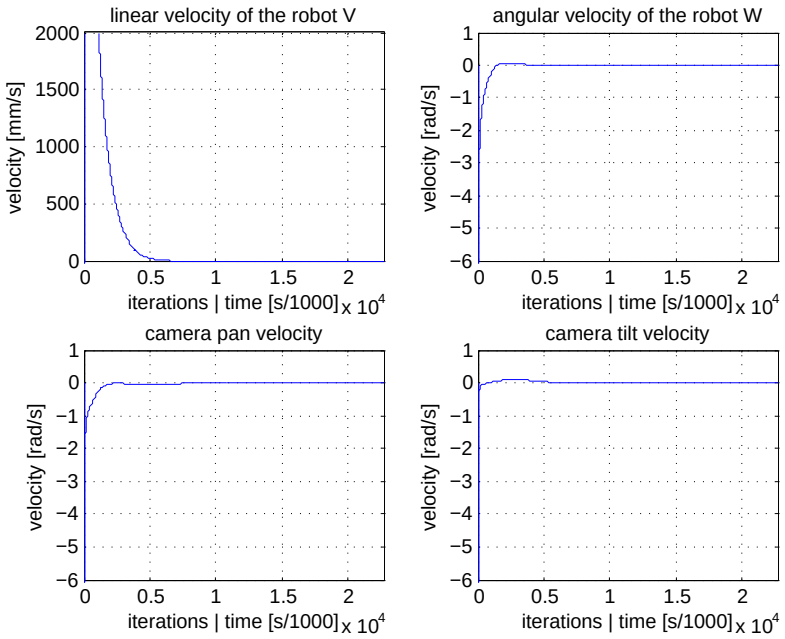


Figure 3.15.: Evolution of velocities in a PBVS simulation

Finally, it should be mentioned that having a consistent threshold to stop the visual servoing experiment is much harder to find than for IBVS which is due to the partly quaternion nature of the error, e_P. Figure 3.14 hints towards this issue. Therefore, other means of determining a good stopping point for PBVS have been implemented in this simulation. The one that was finally used can be seen in Algorithm 3.1 and can be used either to stop PBVS or increase the value λ to achieve an adaptive gain depending on the convergence (assuming that lower velocities are the result of convergence in a task based on exponential decrease).

Algorithm 3.1 Algorithm that determines the stop of position-based visual servoing

1: Calculate the velocities to robot (linear and angular) and camera velocities (angular pan and tilt) depending on the currently detected features
2: **procedure** DETERMINE WHETHER PBVS CAN BE TERMINATED ▷ Called during PBVS execution
3: **if** $\#_{iterations} \geq$ `threshold` **then**
4: $flag_{\nu} \leftarrow$ `FALSE`
5: $flag_{\omega} \leftarrow$ `FALSE`
6: $flag_{\omega^{cam_{pan}}} \leftarrow$ `FALSE`
7: $flag_{\omega^{cam_{tilt}}} \leftarrow$ `FALSE`
8: $flag_{\texttt{terminate PBVS}} \leftarrow$ `FALSE`
9: **if** $|\nu| \leq$ `threshold` **then**
10: $flag_{\nu} \leftarrow$ `TRUE`
11: **end if**
12: **if** $|\omega| \leq$ `threshold` **then**
13: $flag_{\omega} \leftarrow$ `TRUE`
14: **end if**
15: **if** $|\omega^{cam_{pan}}| \leq$ `threshold` **then**
16: $flag_{\omega^{cam_{pan}}} \leftarrow$ `TRUE`
17: **end if**
18: **if** $|\omega^{cam_{tilt}}| \leq$ `threshold` **then**
19: $flag_{\omega^{cam_{tilt}}} \leftarrow$ `TRUE`
20: **end if**
21: **if** $flag_{\nu}$ = `TRUE` && $flag_{\omega}$ = `TRUE` && $flag_{\omega^{cam_{pan}}}$ = `TRUE` && $flag_{\omega^{cam_{tilt}}}$ = `TRUE` **then**
22: $flag_{\texttt{terminate PBVS}} \leftarrow$ `TRUE`
23: **end if**
24: **else**
25: $flag_{\texttt{terminate PBVS}} \leftarrow$ `FALSE`
26: **end if**
27: **end procedure**
28: **if** `terminate PBVS` = `TRUE` **then**
29: `Either terminate PBVS or increase` λ
30: **else**
31: `Continue PBVS`
32: **end if**

3.5. Comparison of IBVS and PBVS regarding robustness towards sensor noise on the basis of simulations

The following section will highlight the capabilities of both, IBVS and PBVS in dealing with sensor noise. For reasons of repeatability and simplicity the comparison will be conducted in simulation. As for the robotic platform, Air-Cobot, both techniques are used depending on which is the more effective one. Through empirical studies we have found the optimal techniques for each visual servoing target. However as a rule of thumb, so to say, IBVS tends to be much faster and better if the target can be well identified and tracked. PBVS on the other hand tends to work better with more challenging targets since it allows to track them even in difficult conditions since the pose estimated can be continued with odometry or other means of localization whenever the target is occluded.

In order to allow a thorough analysis we have decided to use simulations to compare both techniques. This decision enables us to control the amount and distribution of noise. Although the lab environment is already much more controllable than tests in an aircraft hangar or experiments on the airport tarmac, it is still possible that the initial conditions will not be the same for every experiment. During our tests in the Gérard Bauzil at LAAS-CNRS room we have found that experimental results differ depending on the time of day and even the season of the year. Since the room is equipped with very large windows the luminosity is not constant.

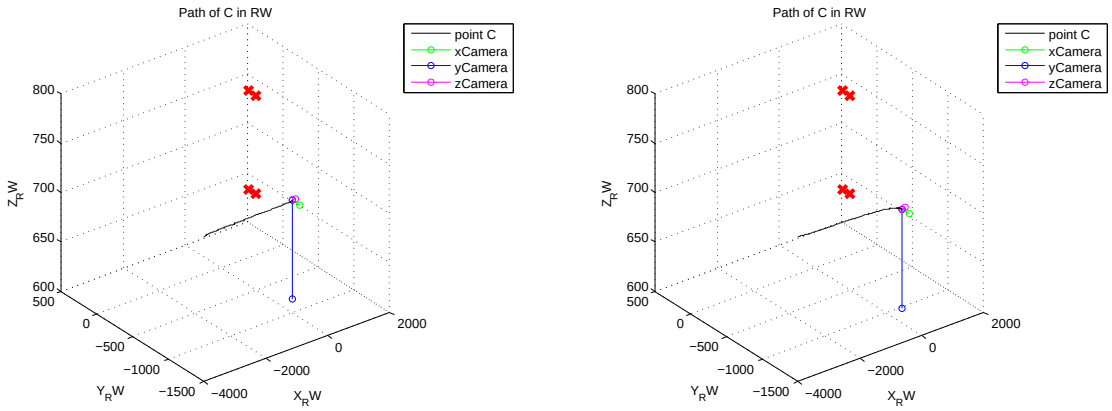
The following sections will explain the simulation conditions that were chosen for the comparison.

3.5.1. Simulation conditions

A white Gaussian noise is added on the image plane coordinates with a 2 % magnitude, for each point in the camera image $P_i = (X_i, Y_i)^T$, a noisy one $\hat{P}_i = (\hat{X}_i, \hat{Y}_i)^T$ where :

$$\begin{aligned} \hat{X} &= X\,(0.99 + 2\,rand(0.01)) \\ \text{with} & \\ &= X \pm 1 \end{aligned} \tag{3.36}$$

To add noise to the simulation we make use of the MATLAB AWGN function. This function includes an "Additive white Gaussian noise" to the simulated camera signal, thus altering the position at which the feature detection would suspect the target. Oppose to general probability terminology the SNR (**S**ignal to **N**oise **R**atio) factor determines the amount and therefore the amplitude of the applied noise. Adding noise is important to determine the quality and robustness of our results. Since a simulation can provide the controlled environment we are looking for adding Gaussian white noise allows us to capture some of the properties of real life experiments.



(a) IBVS: signal to noise ratio = 45

(b) PBVS: signal to noise ratio = 10

Figure 3.16.: Evolution the point C (camera origin) in a noisy simulation

3.5.2. The obtained results

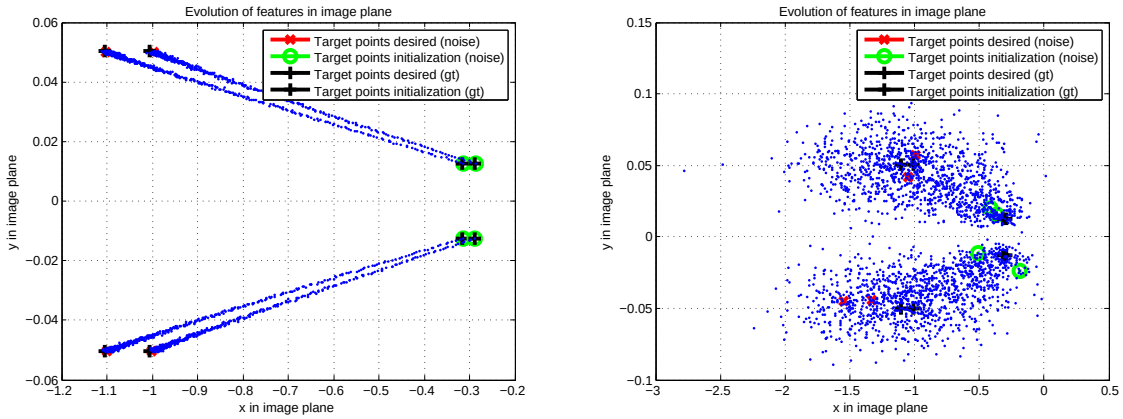
Table 3.3 gives the initial and the desired pose of the robot. As can be seen the camera sensor is placed about 3 meters from its desired goal. In our experiments we determine the desired feature vector, s^*, by virtually placing the sensor at the desired position and taking a snapshot. The displacement on the $y - axis$ simply allows to eliminate the possibility of sign errors in the robot Jacobian.

	$X_M[mm]$	$Y_M[mm]$	$\theta[rad]$	$\theta_p[rad]$	$\theta_t[rad]$
Desired pose	-500	-900	0.0	0.0	0.0
Initial pose	-3500	-900	$-\frac{\pi}{180}$	$\frac{\pi}{180}$	0.0
Final pose (IBVS)	-451.67	-905.31	-0.1127	-0.1672	~ 0
Final pose (PBVS)	-478.54	-905.52	-0.3240	-0.3239	-0.0022

Table 3.3.: Parameters of the IBVS/PBVS comparison

Figure 3.16 displays the movement of the camera in the world frame. During the ~ 2 meter displacement in this mission there is hardly a difference to recognize which is to be expected since the targeted configuration can be reached almost with a straight line.

The second figure relevant in this discussion shows the evolution of the features in the image plane. It basically displays an image that was build from the position of the four features for each cycle of the simulation. Figure 3.17 highlights the difference in the signal to noise ration used for IBVS and PBVS. In Figure 3.17(a) we can observe that the evolution to each feature resembles almost a straight line (blue), where else the features for PBVS (SNR = 10) are much more spread throughout the image plane. Therefore, it is remarkable that PBVS still manages to converge. The most likely reason for that is that the methods input is not any of these features, but however an average



(a) IBVS: signal to noise ratio = 45

(b) PBVS: signal to noise ratio = 10

Figure 3.17.: Evolution of features in a noisy simulation

of all four. Hence, the Gaussian distributed noise is reduced.

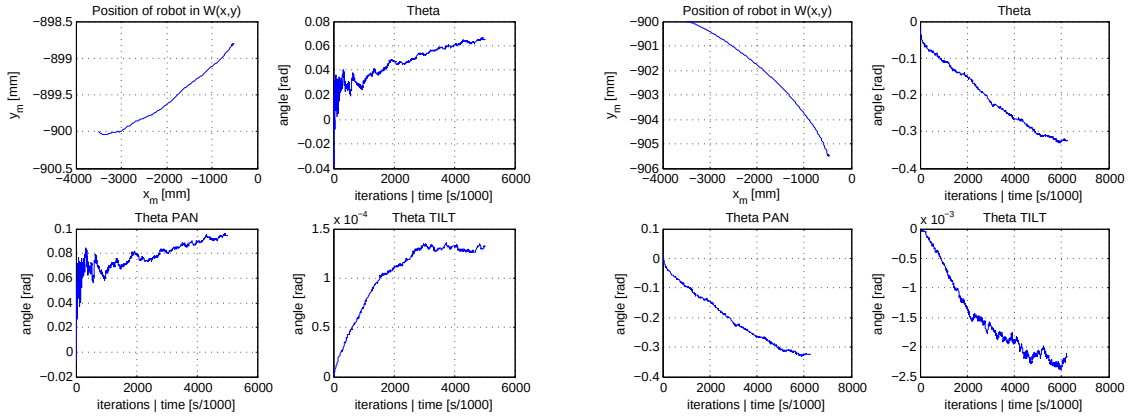
A third figure monitoring the successive position and orientation of the robot and the camera angles furthermore underlines that there is a difference to IBVS and PBVS. However, keeping the scale of the diagrams in mind, Figure 3.18 shows that this difference is rather small and can also attributed to the different noise ratio for a small part.

To complete the survey two last figures are presented. Figure 3.19 gives insight to the system's error and Figure 3.20 to provide a view of the robot velocity differences. It is obvious that for IBVS in Figure 3.19(a) the noise ration never becomes a major factor. Only towards the end of the simulation the error actually seems to spike. For PBVS (Figure 3.19(b)) however, the noise determines the rate of convergence quite early (iteration $\sim$ 2000). As to be expected, the velocities shown in Figure 3.20 are quite similar.

The outcome of these experiments show that PBVS can be more robust if the necessary computational power can be provided and the target consists of many redundant features. IBVS on the other hand is a fairly straight forward approach that does not need a pose estimation (depth estimation), but can profit of it. As stated before, on the Air-Cobot platform both techniques are used, depending on which features are currently recognized (the rest of the aircraft features are estimated with the help of an 3d model of the A320). The following section will provide these real life experiments.

3.6. The Air-Cobot visual servoing strategy

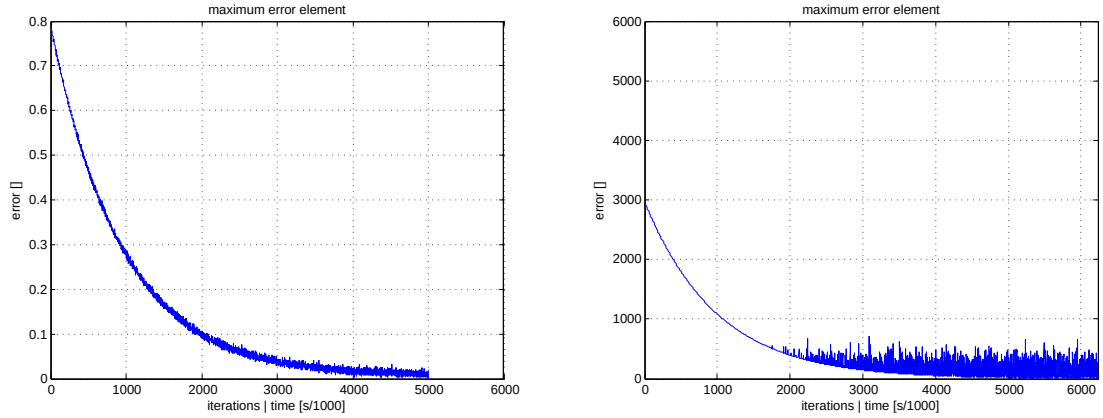
To define our strategy, it is first necessary to detail how the visual signal losses will be treated throughout the execution of the mission. Then we will focus on the definition of the switching instants and of the coupling between the virtual image and the above mentioned controllers.



(a) IBVS: signal to noise ratio = 45

(b) PBVS: signal to noise ratio = 10

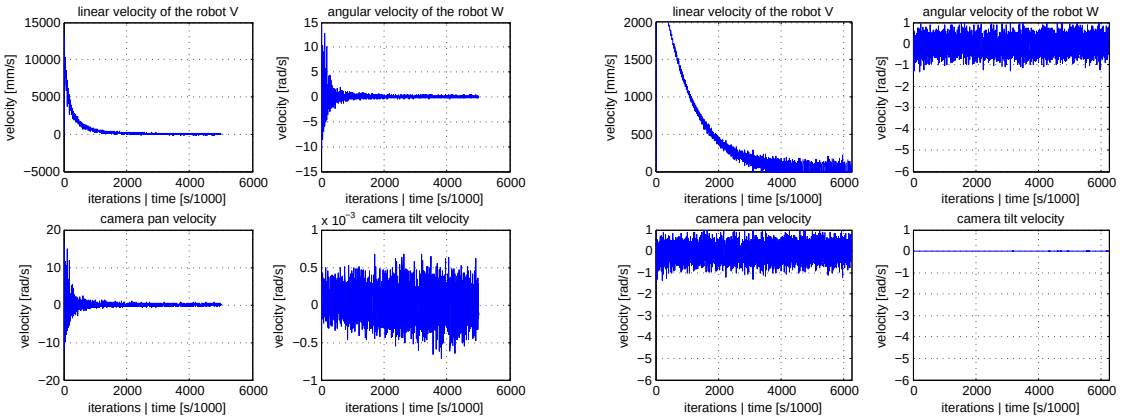
Figure 3.18.: Robot path, orientation and camera state in a noisy simulation



(a) IBVS: signal to noise ratio = 45

(b) PBVS: signal to noise ratio = 10

Figure 3.19.: Evolution of error in a noisy simulation



(a) IBVS: signal to noise ratio = 45

(b) PBVS: signal to noise ratio = 10

Figure 3.20.: Evolution of velocities in a noisy simulation

3.6.1. Handling the visual signal losses

The Air-cobot system is expected to mainly operate in outdoor environments. In such cases, the image processing has to deal with various issues such as the illumination variations (e.g., clouds passing over the sun, ...), the weather conditions(cloudy, rainy, sunny, ...), etc. In addition, even with optimal conditions, because of the distance and of the size of the aircraft, all the necessary visual features cannot be always visible. Furthermore, it may also happen that a temporary breakdown occurs at a hardware or a software level (e.g., ROS nodes crash, etc). Finally, the presence of staff or crew members around the aircraft will increase the risks of occlusion and, as the environment is highly constrained, it is difficult to find a solution allowing to perform the task while preserving the visibility and the non collision. For all these reasons, it is necessary to be able to tolerate total visual signal losses of any origin. A more simple solution allowing to avoid them is not sufficient.

Therefore, instead of performing visual servoing on the raw image in which the features can disappear at any moment due to ambient occlusions, obstacles or airport staff walking in front of the robot or poor lighting conditions, we have developed the concept of virtual (or augmented) image. We take benefit from all the available information. In our particular working context, information can come from :

- the localization given by odometry, IMU and other visual localization methods, that are mentioned in Section 2.2.5 (see Page 45). As explained before, this localization is quite accurate.
- the CAD model of the aircraft that is provided prior to the navigation task and which is kept as a library.

The idea is then to project the missing landmarks (of which we have the precise pose

according to the CAD model of the aircraft) in the image using any localization information available (SLAM, odometry even GPS). This way, even in cases where the visual cues are occluded or can not be identified, it is still possible to find their expected 2D positions in the image. The camera field of view is then extended beyond what is physically possible, increasing the detection robustness in the visual stream. Naturally, this approach works while the localization informations are sufficiently accurate. This means that this solution is adapted for temporary visual cues losses. More precisely, the augmented image is built by adding:

- the real 2D visual cues provided by both available cameras (see the robot description)
- the reconstructed 2D visual cues obtained thanks to the CAD aircraft model and the relative localization[2]

In this way, an " image " is always available, whether the landmarks of interest are visible or not. However, it should be pointed out that the virtual image now depends on the robustness and accuracy of the localization. Hence, the quality of the virtual image deteriorates the same way and in some cases worse than the localization. This can be an issue if the localization is solely based on odometry, which can be the case if IMU, SLAM or GPS are not available (badly lit hangar environment).

3.6.2. Coupling the augmented image with the visual servoing

As previously explained, the augmented image is intended to provide the necessary visual information (either measured or reconstructed) for the control. Consequently, it is always possible to build our visual features vector s from it. However, to control our robot, a reference value of this vector denoted by s^* must be available. This value is generally chosen before launching the visual servoing controller and allows to define the goal to be reached. One of the proposed approaches has the advantage that, once the reference visual features has been fixed, its value can be kept during the whole execution of the task, whether the visual features are available or not. This is possible because the type of the considered cues is never modified during the task. Knowing s and s^*, the visual servoing controller can always be computed and applied to the robot. The mission can then be performed, even if no visual feature is available temporarily.

3.6.3. PBVS and IBVS switching

Following the previous section, we have chosen to combine both image-based and position-based visual servoing to benefit from their respective advantages while limiting their drawbacks. More precisely, our key-idea is to switch between PBVS and IBVS. It is then necessary to define the switching instant between them. As previously explained, PBVS will be used far from the aircraft (that is when the features are not visible yet)

[2]This relative localization is updated using the detectable visual features.

and IBVS close to it to improve the positioning accuracy. Thus, the switching instant is defined by the instant when the item to be inspected becomes fully (robustly) visible in the real image. During our experiments we have found that we can get acceptable results with PBVS even with the window line on the aircraft. This set of features is fairly easy to detect even when the robotic platform is far from the aircraft. The same goes for the aircraft wheels. Static port and duct on the contrary are features that can only be reliably detected whenever the robot is sufficiently close to them. If they are detected the visual servoing node switches from PBVS to IBVS.

3.7. Experiments on Air-Cobot

The following section provides visual servoing experiments conducted on the robot. First off, the test environment (that was necessary whenever an aircraft or the airport itself was unavailable) is introduced. After that a brief introduction to the library used as baseline for visual servoing algorithms is presented and explained. Then, different test scenarii are displayed and finally a conclusion about the experiments closes the chapter.

3.7.1. Test environment

Since the time for experiments on the airport, tarmac and hangars was limited we have created a test environment. Its main purpose, was mainly to allow performing interesting experiments in an environment that was constantly available. In this way, test cases could always be presented to project partners. Furthermore, this solution had the advantage that these tests could be made repeatably with little change in the surrounding conditions. The following section gives a short description of these circumstances and general framework built in the lab. Due to the lack of having an aircraft available in the lab the testing of feature detection, tracking and robustness of our visual servoing module had to be conducted in the following improvised environment.

Figure 3.21 presents the main outline of how we have re-created an aircraft. The visual cues mounted on the fence seen in Figure 3.21(c) are mounted in roughly the same relative position as they would appear on the aircraft.

Please recall Figure 2.4 on Page 44 and also remember that we were talking about re-imagining the pre-flight inspection with respect to the capabilities of the robotic platform. The reader will then realize that the upcoming test will replay a situation during an approach of check-point 1 and 2 of the aircraft. During this approach a person (airport staff) will then walk past the camera, occluding all features for a brief moment of time.

The second experiment that will be shown will be closer to the expected missions of the robot. Air-Cobot will start at the initial point of the envisioned pre-flight inspection (namely point 0) and will make its way towards point 1 and 2. Both of these points

(a) Duct and static port

(b) Duct close up

(c) construction site fence with visual cues

Figure 3.21.: Example of the lab environment where the visual servoing experiments were conducted in

will be reached using PBVS and the closer the robot gets to the designated check points the higher the likelihood that IBVS is used. After a short use of a metric map based guidance technique, IBVS is used in order to reach point 3. Due to the lack of space in the lab, tests of reaching successive points were only conducted on the tarmac and will be presented in the video.

Furthermore, an inflatable air mattress tire was used to mimic the aircraft jet engine. Its size and shape was a sufficiently close abstraction of a real jet engine so that the feature recognition could be trained on it. Finally, a real piece of aircraft skin was available during the beginning of the project which we modified with prints of the static port. Unfortunately, towards the end of the thesis, this piece was needed for experiments unrelated to the Air-Cobot project. This environment allows to remove or temporary occlude features in order to test the robustness of the navigation system towards changes

in the environment (bad lighting can lead to some visual cues not being detected).

3.7.2. Applying ViSP in the Air-Cobot context

As ROS, the middleware chosen for the Air-Cobot project (see Section 2.2.2 on Page 37), ViSP is an open source software designed to minimize the time spent on integration of functionalities. ROS provides middleware specific functionalities such as drivers for sensors, communication interfaces and so on. ViSP on the other hand contributes by allowing for fast prototyping of visual servoing approaches (all necessary basic functions like IBVS and PBVS can be directly implemented, modified or even combined to the users needs). Furthermore, the Lagadic team (the research group that created and currently maintains the code), provides a version of ViSP that runs directly as a module in ROS, as the navigation stack, or OpenCV for example. Additionally, the user can choose from several procedures to estimate depth or complete $3D$ poses which is essential for PBVS (if no other option of evaluating the sensor to object frame can be estimated). It is also helpful to improve robustness for IBVS (depth in interaction matrix). Lastly, it should be mentioned that for all matrix computations (homogeneous matrices, etc.) the ViSP library was used instead of the also common EIGEN implementations. This is solely due to the fact that this way the Air-Cobot team could circumvent installing additional packages.

3.7.3. Experimental tests combining IBVS with PBVS

The upcoming section will provide some "real world" experiments of the previously discussed approaches on the Air-Cobot platform. Three different test cases will be shown. The first two experiments will simply combine an IBVS with a PBVS approach with very limited parts during which the movement of the robot is generated by the go-to-goal controlled presented in Chapter 2. During both of the first examples we will furthermore show that the system can deal with feature loss. A final experiment will show how the proposed system can be used as the main means of navigation during a pre-flight inspection mission. Therefore, some of the movements will be fully controlled using metric-map-based algorithms while others are executed with the help of either IBVS, PBVS or both techniques.

IBVS & PBVS combination

Our first experiment will present a combination of IBVS with PBVS. Desired poses provide the necessary s^* and is generated by keeping track of the robot pose with the help of odometry and visual pose estimation (of the sensor towards the landmarks).

Figure 3.22 provides a view of the laboratory environment were the experiments were conducted in. It also shows the robot in operation and how the robustness to partial and full occlusions was tested (person walking in front of the artificial targets).

(a) t= 110 (b) t= 116

Figure 3.22.: Extract of video sequence showing an outside view onto the experiment

The conducted experiment can be seen in more details from the perspective of the robot in Figure 3.23. Here the actual time stamp of the frame is given as label of the figure.

Figure 3.24 displays how the robot center of gravity has moved during the experiment.

Since the angular difference between the approach and the backward movement is almost zero we can see that the robot moved forward and backward on almost the identical path. Investigating the velocities, we can see that the linear velocity was implemented without a rate limiter (red line). And that the adjustments to the stereo-camera system are extremely low. The reason for this is easily found. Although the Field of View (FOV) of the stereo cameras is fairly low, the lab environment with the scaled aircraft model and its features does not allow for extensive lateral movement. Therefore the features needed to be placed almost directly in front of the robot, eliminating any reason for large lateral displacements (see blue curve to see the platforms angular velocity) and while doing so, also reducing the angular movement of the camera sensor to a minimum (see pan movement in the third image). One of the most striking challenges during the experiments on real aircraft was certainly the vertical position of landmarks and features. Due to the size of the aircraft and the travel distances an angular tilt change of more than 45 degrees was very common.

Experiment highlighting the capability of handling Feature loss

As we have shown, our system can handle partial feature loss. With the following experiment we want to take a step further. Here, the person walking in front of AirCobot will be much closer to the robotic platform and therefore its sensors. The effect of this experimental set-up will be that all features will be lost at one point in time. With the help of the virtual images it will not be necessary to directly switch to a metric-map-based guidance solutions, which is however, available should the need arise. In this experiment, the virtual image will be used in order to continue sending valid movement requests to the platform.

It is of utmost importance that the robot receives valid steering commands for the

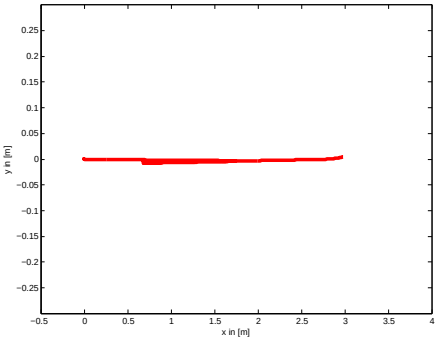
(a) t= 25 (b) t= 27

(c) t= 52 (d) t= 80

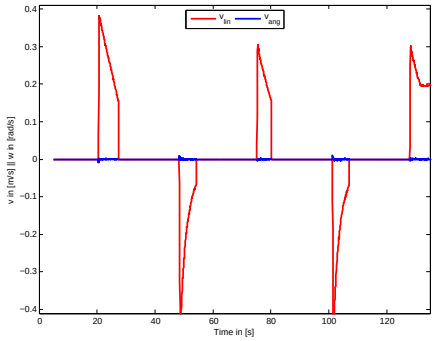
(e) t= 106 (f) t= 132

(g) t= 133 (h) t= 133.3

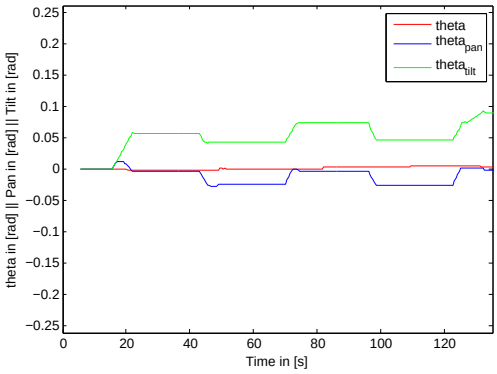
Figure 3.23.: Extract of video sequence showing the system handling switching between IBVS and PBVS



(a) Robot location as a set of successive positions during the experiment

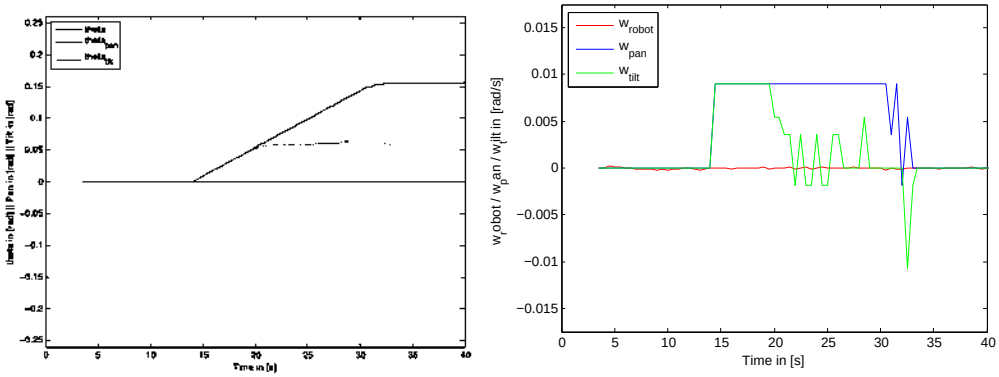


(b) Linear and angular velocities of the platform



(c) Orientation of the platform and the front PTU camera unit

Figure 3.24.: Position and velocity (linear and angular) plots obtained during the IBVS-PBVS combination experiment



(a) Orientation of the robot(world-frame) and camera PTU (robot-frame)

(b) Derived angular velocities

Figure 3.25.: Orientation and velocity plots obtained during the feature loss experiment

platform and the camera PTU at all time. Since it is impossible to guarantee an environment without obstacles, the robot needs to be equipped with algorithms to handle feature-losses and collisions. As it has been explained throughout this chapter, the augmented image by visual localization has been introduced in order to avoid failure states whenever such situations arise. In the following experiment, we will present such a case and the reaction of the system.

A moving obstacle will pass through the view of the camera sensor during a visual servoing mission. The robot is constrained to bring the center of gravity of the set of multi features towards the center of the image. This is a scenario that has proven to be extremely useful for small features and long detection distances (e.g.: robot approach from a distance of +30 meters towards the aircraft while using the passenger windows for guidance). During the experiment a person will walk right through the image making it impossible for our blob detection and tracking algorithm to relocate the former chosen window. Since the person is moving, the feature will not be hidden for long. Additionally, the set of features (windows and duct) are known by the system, so it is able to estimate the pose of lost features if a minimum of two features remain detectable.

Figure 3.25 in conjunction with Figure 3.26 show that feature loss is managed by the virtual image. The detectable landmarks then help to estimate the pose of the occluded ones. Figure 3.26 shows screen-shots of a video recorded of the Air-Cobot desktop during the experiment. Features that are detected have a border and appear in the color green. While the obstacle is moving through the scene some features are occluded while others remain visible for the sensor. Occluded features that the system can still estimate (due to remaining features) are marked by a red border. Figure 3.25 highlights that during the time of feature occlusion (t$\approx 30 - 31\,s$) the camera is continuously moving towards the local minima. The overshoot and minor oscillations in the PTU velocities can be explained by the lack of an appropriate threshold that would allow the sensor to stop moving whenever the solution is between two incremental positions of the PTU.

For this experiment the time of occlusion is not important. As long as movements relative to the target are small our studies have shown that re-detection is possible. However, changing light conditions during the occlusion can lead to problems. This needs to be taken into account for further improvements of the algorithm.

Experiment using IBVS and PBVS in conjunction with metric-map-based navigation

Additionally, we want to present an experiment in which we have combined the use of IBVS and PBVS depending on the visual cues and the situation. The improvised aircraft that was used in the last experiment will be extended with the rubber ring that was introduced before. This ring will be positioned approximately at the location where usually the aircraft engine would be. The underlying challenge of this experiment is firstly, to utilize different visual servoing techniques depending on their success rate that was empirically determined during our tests and secondly, to improve the detection of all visual cues by incorporating self localization and prior knowledge of the aircraft.

During the experiment, the robot will receive an inspection plan in form of one .xml file for the navigation and one that helps the robot to figure out the proper choice of visual cues and inspection method. Both files were written specially for the fake aircraft in our laboratory. It is this .xml file that represents the only difference during the experiments in the hangar, the laboratory or on the tarmac. For this particular experiment, the robot will first leave its designated charging area in the Gerald Bauzil laboratory room. Since no consistent targets for guiding where available this part is achieved with the controller that handles robot displacements with the help of self localization in the metric environment. Figure 3.27 provides a very simple outline of the experiment environment. Furthermore, the path on which the robot will move is given. Along the lines of the path we have given the controller which will be dominant during this phase of the experiment. Any feature loss will be compensated by the odometry helping to preserve the virtual image.

Having left its designated charging area, the robot will approach the targets on the yellow construction site fence using mainly the line of black dots that represent the windows of the aircraft. For this particular target we have found that PBVS has the highest success rate since it can profit of the metric relative localization between the target and the robot the most. During this approach the robot will however switch to another target once it can sufficiently detect said visual cue. The main idea here is that during our experiments on the real aircraft, we have found that although most of the aircrafts inspection points have some visual markings (red circular or rectangular shape around them), but these markings can only be detected by our cameras, once the sensor (and so the robot) is sufficiently close to it. The reason for this is that these lines are usually extremely thin and due to the lucid surface of the aircraft can reflect the environment quite strongly. Hence, for the first part of the approach the line of windows are a robust visual cue that can be tracked while the robot is moving. Even if the ground is not smooth these features will not be lost. Once the robot is closer, we will switch to

(a) t= 28 (b) t= 29

(c) t= 29.5 (d) t= 30

(e) t= 31 (f) t= 32

Figure 3.26.: Extract of video sequence showing the systems capability to handle feature loss during execution of a visual servoing-based navigation mission

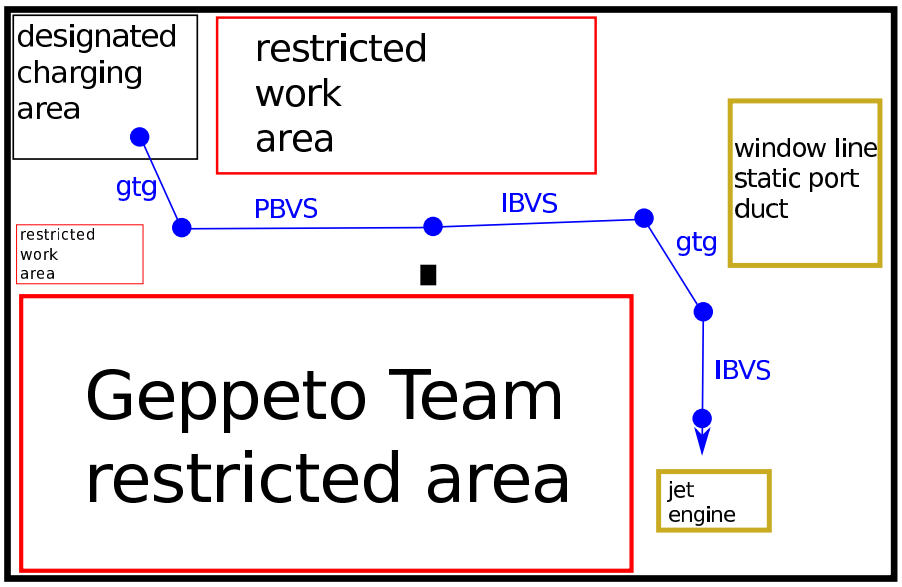


Figure 3.27.: Outline of the Gerald Bauzil room in which the experiment takes place with expected path for the robot

targets such as the static port or the duct (see Figure 3.21(a) on Page 108). One of the main reasons for this is that it will become much harder to track the line of windows of the aircraft the closer the gets to it and since the the main interest belongs to items such as the static port, duct, etc. and these become more and more visible we will switch the main visual cue. Of course, keeping in mind our virtual image, the location of each window is still known.

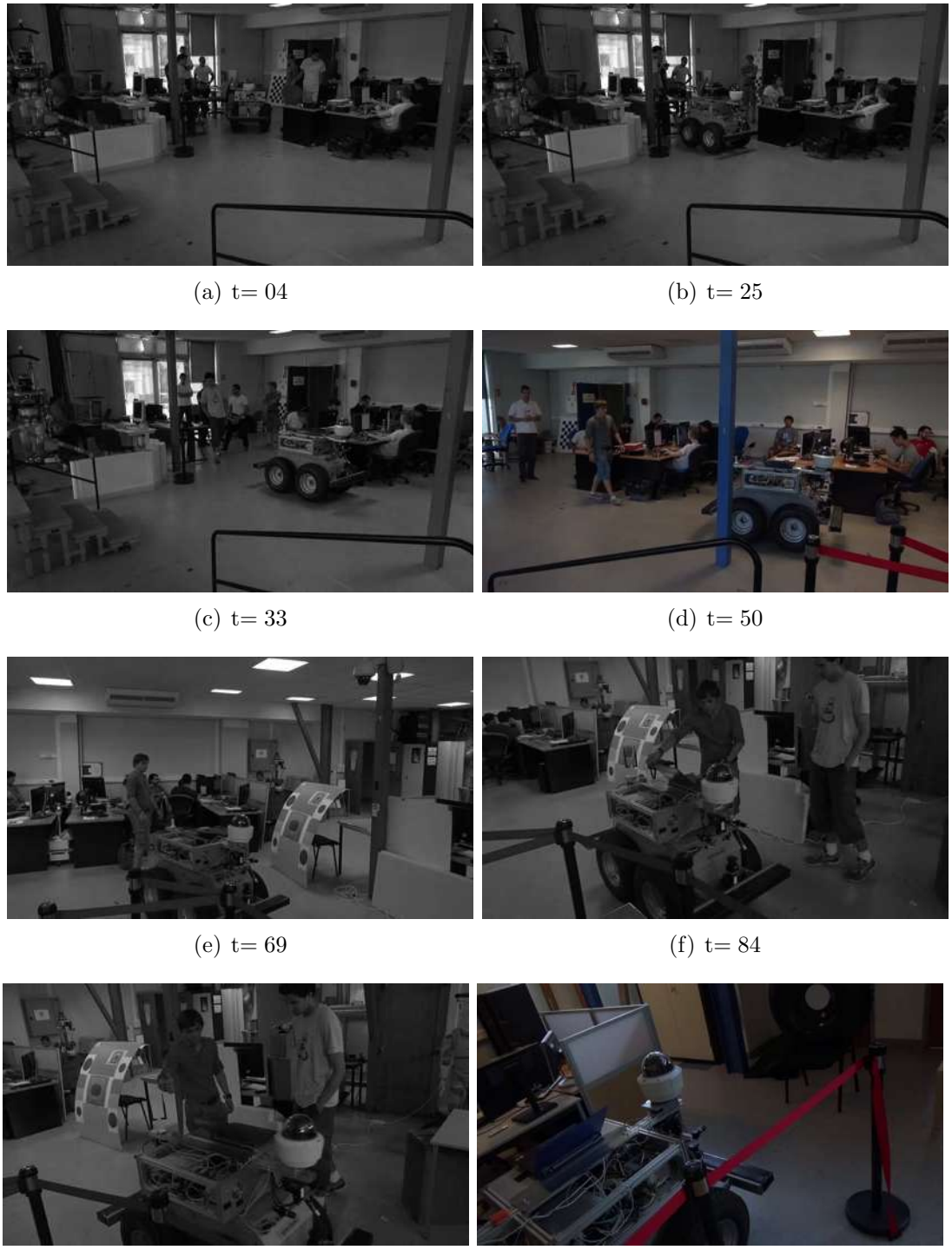
Following these two separate approach steps the robot will use the knowledge of its position relative to the aircraft to orient its sensors so that they are able to detect the jet engine (in our experiment the jet engine is represented by the rubber tube mentioned before). During our extensive experiments we have found that having the sensors pointing at the target is not sufficient for a precise visual servoing. This is due to the steering technique of the robot and will be explained later (see Section 5.2.3 on Page 157). A short explanation is that the steering technique of the robot is not well suited for urban environment. It is much better for cross country deployment. We have found that it is better to avoid high steering angles combined with linear velocity in order to reduce the shaking of the platform. Since the jet engine is located at almost an 90 degree angle to the static port, the robot will first turn in place these 90 degrees and keep the cameras static until the maneuver is completed. After this movement (which is quite shaky) the detection modules will be re-engaged. As we can see from the screen-shots of the video, the robot will then move towards the fake jet engine for inspection. This is also done in two steps. The first step is a short approach using the metric go to goal controller that was explained in the last chapter. Once this maneuver is completed the robot uses the PTZ camera and IBVS in order to position itself to perform the next inspection.

Once the inspection is completed (in the experiment at the laboratory the inspection is skipped for obvious reasons), the robot will switch to the finished state and wait for new instructions from the operator. Due to the confined space of the Gerald Bauzil room, it was necessary to reduce the obstacle avoidance controller's capabilities while still preserving its emergency safety functionalities. Hence, for the presented scenario we had to artificially limit the features of the controller to only react if an object would get into the robots final security area (an area about 0.1m around the robot that is usually used to completely stop the robot, should an object ever get that close to it). For experiments on the tarmac or in the hangar environments this step was not necessary.

3.8. Conclusion

This chapter has focused on the visual servoing strategy which was developed in the Air-Cobot project. While our first goal was to recall IBVS and PBVS principles and propose a comparative analysis, our second intention was to highlight the interest of defining a virtual (or augmented) image to deal with partial as well as total visual signal losses which are very likely to occur during a inspection / maintenance mission. This virtual image is fed with both measured and reconstructed visual features of the landmarks of interest. The reconstruction is obtained by using both the aircraft CAD model and the data provided by the localization process. We have also shown the interest of switching between PBVS and IBVS depending on the distance separating the robot and the aircraft, the type of visual feature and availability of secondary localization functions (odometry, SLAM, etc.). Simulation and experimental tests have also been provided.

For future works, it is still necessary to improve the robustness of the image processing algorithms to changing luminosity conditions. Indeed, despite the construction of the virtual image, the experiments have highlighted issues with the robustness of the system. As we have described in this chapter, the change of luminosity during tests that span over a full day can even influence the results of tests conducted in a semi controllable environment such as the Gerald Bauzil room at LAAS-CNRS. This issue is magnified when the tests were conducted outside on airport-tarmac environment or Hangars with large light translucent roofs. Additionally, whenever the luminosity to changing extremely rapidly (Air-Cobot navigates from the airport Hangar into an artificially lid hangar) the obtained results still show much room for improvement. For these cases, the metric localization was of great help. Another interesting lead concerns the building of the visual servoing control. Indeed, as previously mentioned, our robotic system is endowed with several cameras: the stereo-camera-system in the front, the one in the back and a PTZ that is mainly focused towards the front. As pointed out during this chapter the PTZ camera has the disadvantage of being less agile than the stereo camera system and also it is not equipped with a global, but a rolling shutter. Compensation for these issues were presented.



(a) t= 04 (b) t= 25

(c) t= 33 (d) t= 50

(e) t= 69 (f) t= 84

(g) t= 96 (h) t= 122

Figure 3.28.: Extract of video sequence showing the systems capability to perform a combination of IBVS-PBVS-Metric based navigation

In this work, three visual servoing nodes are running in our ROS architecture. It is then worthwhile to consider how to define the visual servoing control law sent to the robot. For this project, we have simply chosen to operate Air-Cobot using solely the node providing the highest reliability factor. This factor is evaluated on the accuracy of the feature detection and resulting pose estimation. All remaining nodes will be used to operate the pan-tilt positions of each camera system, thus keeping the camera on target and ensuring that a switch back to a different vs node is always possible. Additionally these nodes provide a reasonable option to fall back on should the primary node crash, camera sensors experience failure or occlusions happen. In the future, it would be then very interesting to analyze more deeply this aspect to find an efficient way to combine the different visual servoing when it is pertinent to do it. Indeed, it must be mentioned that, due to the positioning of the different cameras on the robot, it is unlikely that all the visual servoing instances will send similar reliability information. Hence, finding additional factors that evaluate reliability would be advisable.

4. Obstacle avoidance in dynamic environments using a reactive spiral approach

Contents

When something is important enough, you do it even if the odds are not in your favor. Elon Musk in an interview about him creating SpaceX [mus,]

4.1. Introduction

The following chapter explains what kind of measures were taking to design a robotic platform that can be considered safe for its purpose. Looking into other areas like autonomous vehicles for example, the word safety does convey a much more precise meaning. For the purpose of this project however, safety will be used in its colloquial sense. Instead of making a full analysis like it is usually mandatory in software development (right side of the V−model [Forsberg and Mooz, 1991]) a simple investigation of the robot's limitation will suffice. The real time capability based on the robot's maximum velocity will be taken to evaluate all functions. Air−Cobot is still a prototype and has to be evaluate exactly as such. This means its operator still needs to be trained to understand the robot's limitations. The platform is equipped with several emergency stop buttons, it will engage the brakes once it looses connection with its remote control or the front / rear bumper is pressed. However, the robot is expected to handle some perceivable situations automatically. With human force in its vicinity, the Air−Cobot is equipped Obstacle Avoidance techniques (OA). The objective of the chapter is then to present the Obstacle Avoidance strategy implemented on Air−Cobot. First, we present the Air−Cobot environment.

4.1.1. Air−Cobot navigation environment

When designing a mobile robot for the purpose of autonomous navigation through environments that include static and moving obstacles, objects that are known a priori as well as objects that are dynamically appearing, it is essential to equip this platform with some kind of obstacle avoidance algorithm. This becomes further relevant the more this platform is supposed to interact, integrate itself smoothly into its environment (abandoning the use of security cages as they are quite common in the industrial automated production). Although the airport is a highly structured and controlled environment, obstacles handling (especially for moving objects like airport carts, buses and airport staff) is an important feature in a navigational framework, just like robot localization, controllers, etc...

Additionally, the robot must be able to deal with forbidden zones which can range from loading area and parking spots for aircraft specific vehicles, to fueling stations and other hazardous areas (see Figure 4.1). To prevent the vehicle from entering them, the obstacle avoidance method treats them as virtual obstacles included in the obstacle detection algorithm. As shown in Figure 4.1, the algorithm will be augmented with values of the forbidden zones to simulate obstacles in the robot vicinity whenever it is getting too close to them. The locations of these areas are provided to the platform using a GPS sensor and a priori generated forbidden zones map. Depending on the area (green and red) it is located in during the approach, a wall will be simulated. The robot will then avoid the forbidden zones as it would be a real obstacle. For safety reason, the GPS box has a direct access to the CAN-Bus and can stop the robot directly if it is about to enter one of these zones. In such a case, only a human operator can make the

robot leave this zone though remote control.

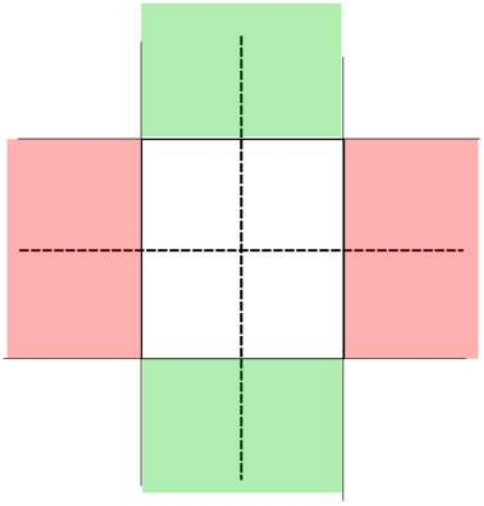


Figure 4.1.: Dealing with airport forbidden zones

In order to generalize the problem, the following assumptions where made: For example, the obstacle speed and trajectory is never changing rapidly and the obstacle's linear velocity can be considered constant. The same way, the obstacle's trajectory is evolving slowly in comparison of the cycle time of our algorithms. Furthermore, the obstacle geometric shape is either circular or its geometric aspect ratio (length to width) is close to 1. Obstacles with higher aspect ration (e.g.: walls will be treated differently).

4.1.2. State of the art of obstacle avoidance

The following subsection presents an overview about already matured obstacle avoidance methods and highlights the contribution of this thesis.

Many approaches that can handle static obstacles are well known and have matured during the past decades. This field is a very vast domain and methods differ a lot depending on the way the navigation strategy processes have been instantiated. Generally, whenever a map is available for the environment, a planning method produces a path allowing to reach the goal while taking into account the obstacles which are known a priori. On the other hand, when no map is given prior to the navigation, the robot motion towards the goal is defined using sensory data. Obstacles are then considered on the fly, leading to reactive approaches. We propose hereafter a brief review of the most well known techniques in the domain of obstacle avoidance, keeping in mind the need of treating the case of both static and dynamic obstacles.

Reactive approaches

Artificial potential fields One of the furthest developed and widely accepted reactive methods for obstacle avoidance, initially meant for manipulator collision avoidance [Zhang et al., 2012] is artificial potential fields [Khatib, 1985]. Virtual potential fields produce attracting (goal) and repulsive (obstacles) forces that "steer" the mobile robot through the environment [Khatib, 1985], [Koren and Borenstein, 1991]. An obvious advantage of this technique is its reactiveness. Provided a sufficiently accurate robot, obstacle and target localization, an artificial potential field can be generated and modified on-line. Its modification can be necessary if obstacles move, change shape or are newly discovered. As it has been pointed out, a general drawback of the original method has been its sensitivity to local minima [Zhang et al., 2012]. These local minima can lead to situations in which the robot appears to be stuck. This problem can be overcome by either introducing unstable states dynamics [Mabrouk and McInnes, 2008a] or by allowing the robot to switch to a different behavior such as for example wall following [Mabrouk and McInnes, 2008b]. It is also possible to use potential field techniques at the control level. For example, in [Cadenat, 1999], potential field have been defined to perform a visual servoing task while avoiding fixed obstacles. In [Folio, 2007], the same approach has been used to guarantee both non collision in a static environment and non occlusion of the target. In those cases, the problem of local minima has been significantly reduced by insuring the robot is given a nonzero linear velocity at any time during the mission. However, despite these improvements, it is not always easy to find an efficient potential function that allows to satisfy all the constraints (goal reaching, non collision, non occlusion, etc.). This problem has been pointed out in [Mantegh et al., 2010].

Finally, it should be mentioned here that more advanced methods that introduce fuzzy logic or even hydrodynamics exist which address these flaws of the artificial potential field approach [Pérez-D'Arpino et al., 2008], [Zhang et al., 2012].

Vector Field Histograms (VFH) Another technique falls into the domain of local approaches is **V**ector **F**ield **H**istograms VFH [Borenstein and Koren, 1991] and its advancements VFH+ [Ulrich and Borenstein, 1998] and VFH* [Ulrich and Borenstein, 2000]. The main idea behind this real time motion planning method is to make use of a histogram grid built from sensor information. Robot dynamics and geometry can also be taken into account. However, as the artificial potential field method before, VFH struggles with global path optimality.

Velocity Maps Velocity maps [Owen and Montano, 2005], velocity obstacle [Fiorini and Shiller, 1998] or forbidden velocity map [Damas and Santos-Victor, 2009] all refer to an approach that allows control of the robot linear velocity safely through an environment filled with moving obstacles. The assumption is that all obstacles that can become a danger to the robot are known and that their velocity, orientation and current position

can be estimated fairly accurately. Furthermore, it assumes that they move at constant speed in a linear motion during the relatively short time increment that the technique is applied. By knowing the environment and its obstacles, a map can be created containing potentially dangerous velocities. When maneuvering the robot at speeds that are not inherent in the velocity map, this approach can guarantee a collision free movement. The big advantage of such an approach is that the computational effort necessary to find a suitable velocity for the robotic platform is quite low. However, the velocities of the robot's surrounding objects need to be measured very accurately.

Re-planning or modifying a planned path

Previously presented approaches belong to the group of reactive techniques. This means that the path or the motion of the robot is defined using sensory data only. Approaches from another group are more tightly connected to a global planner. For example, works [van den Berg and Overmars, 2005] and [Koenig and Likhachev, 2005] proposes to re-plan a new path to take into account non mapped obstacles. It is also possible to modify a pre-planned path as in [Khatib et al., 1997b] or [Lamiraux et al., 2004b]. This last idea was always at the core of two widely used methods known as DWA (**D**ynamic **W**indow **A**pproach) and TEB (**T**imed **E**lastic **B**ands). We briefly focus on them because they have been implemented through ROS and are utilized by a large community of researchers.

DWA The **D**ynamic **W**indow **A**pproach, or simply DWA, is an alternative for the NAV stacks local planner (base local planner) - an open source, of the shelf software accumulation designed to allow roboticist to test new approaches. DWA allows to locally deform any globally planned trajectory when it is necessary with the help of a costmap (or weighted occupancy grid) [Fox et al., 1997] [Gerkey and Konolige, 2008]. This costmap is built using the sensory data which characterize the robot immediate environment. It is always anchored in the center of the robot (2d) so that it moves with the robot. Thus, appearing to be a dynamic window and hence the name. Other articles such as [Kelly, 1994] follows a similar idea.

TEB As DWA, TEB, **T**imed **E**lastic **B**ands, is a local planner capable to deform a given global trajectory (bias) so that obstacles can be avoided. See for example [Rösmann et al., 2012] and [Rosmann et al., 2013]. Basic idea and concept of the approach are somewhat different. The method takes temporal aspects such as robot dynamics and maximum velocities into account to deal with static and dynamic obstacles alike.

Discussion and positioning of the suggested approach

Our goal is to develop a method able to deal with the particular specifications of the airport environment (forbidden zones, static and dynamic obstacles, pedestrians, etc.). Furthermore, we need to take into account the way the different navigational processes have been instantiated. As explained before, both a metric and topological map is

available. These maps then contain all the information about known obstacles, typically the forbidden zones, the buildings, and so on. Hence, the planned path allowing to reach the inspection zone or providing the sequence of checkpoints will most likely not contain obstacles. Even though the airport belongs to one of the most regulated areas we can imagine, the risk of having dynamically appearing objects in the robot's lane is not zero. Thus it will be the main task of the approach we will propose in this chapter to deal with them. Several remarks have to be given before continuing with its description. Firstly, the motion of the robot is defined by a sequence of different controllers (go-to-goal, visual servoing, ...), showing that the action part in our navigation strategy is mainly in the local domain. Secondly, solutions consisting of re-planning a path or modifying it on-line are often more time-consuming and introduce some rigidity opposed to reactive, sensor based methods.

Taking these elements into account leads us to choose an obstacle avoidance method belonging to reactive approaches. An analysis of the above mentioned techniques has pointed out their main drawbacks : local minima for potential fields and VFH, the very constrained hypotheses for velocity maps. In addition, the latter are often used in a planning context, whereas our work is at the control level. Finally, most presented techniques can handle moving obstacles up to a certain velocity. For example one can imagine that if the obstacle velocity is sufficiently lower than the robot velocity [Khatib, 1985], [Koren and Borenstein, 1991] and [Borenstein and Koren, 1991] will allow the platform to perform safe obstacle avoidance. But what will occur if the velocities are greater? To deal with dynamic ones and keeping the constraint of using vision-based navigation for the go-to-goal behavior of the robot, [Cherubini et al., 2013] and [Cherubini et al., 2014] propose an approach which explicitly takes into account obstacle velocities, estimated by a Kalman filter. These velocities are then used to predict the obstacle positions and choose the best tentacle that provides the optimal path in order to guarantee collision-free navigation. However, the avoidance robustness then depends on the estimation quality and these methods do not remain only sensor-based. For all these reasons, we have chosen to develop a new approach explained in the following subsection.

4.1.3. Introduction to spiral obstacle avoidance

We propose an approach tightly connected to spirals to fulfill the demands posed by the Air−Cobot project. This suggested approach belongs to the reactive domain of obstacle avoidance techniques for obvious reasons. It is inspired from articles explaining the somewhat odd behavior moths and other bugs seem to exhibit around artificial (man-made) light sources during the night time [Himmelman, 2002] and [Boyadzhiev, 1999]. Indeed, it has been observed that they perform a spiral, the center of which is the bright light source (see figure 4.2). In the literature, its designation ranges from logarithmic spiral and equiangular (René Descartes) spiral to simply "growth" spiral and "Spira mirabilis" (Latin for miraculous spiral) by Jacob Bernoulli.

The reason for this seemingly odd behavior of bugs around artificial light sources is their way of navigation. For millennia they were able to use the sun and (by night) the

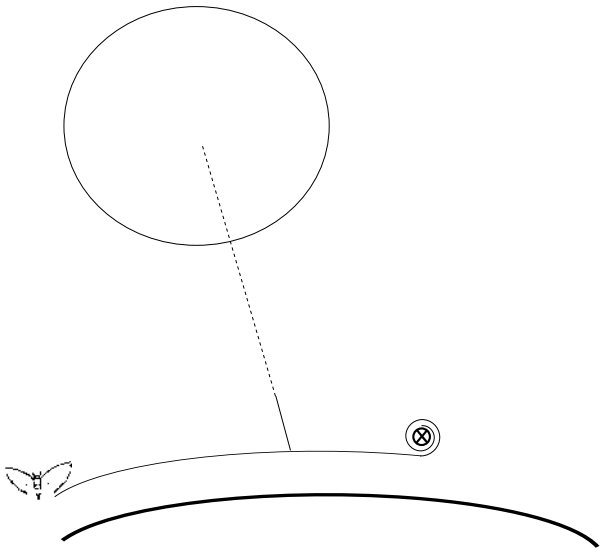


Figure 4.2.: Moth flight path being "redirected" by an artificial light source

moon to navigate their environment. While keeping always the same angle towards this natural light they ensured a quasi straight line (since moon and sun are never reached). Figure 4.2 exemplifies this navigation approach that is possible because of the enormous distance and size of these light sources compared to the almost insignificant distance traveled by the bugs. However, artificial light sources (especially during night time) can be, at least locally, of much higher intensity and thus be "mistaken" as e.g.: the moon. When trying to keep the same angle towards an artificial light source with a center point that is now many times closer than was ever used before, disturbances in keeping an exact angle towards the light source which were insignificant noise before result in clearly visible spirals that ultimately lead the bug into the artificial light source. A visualization of this description is available in Figure 4.3. It is worth to mention that the approach flight path of a hawk towards its prey also resembles a logarithmic spiral as was demonstrated in [Chin, 2000].

Several articles ([Mcfadyen et al., 2012a], [Mcfadyen et al., 2012b], [Teimoori and Savkin, 2008] and [Futterlieb et al., 2014]) have shown that it is possible to apply this navigational strategy to obstacle avoidance. In articles [Mcfadyen et al., 2012a] and [Mcfadyen et al., 2012b], the authors enable a UAV (**U**nmanned **A**erial **V**ehicle) to avoid collision during visual servoing. [Teimoori and Savkin, 2008] are using spirals not only to avoid obstacles, but also to follow a moving target.

Inspired by all these realizations, we have shown that spiral obstacle avoidance is a valid option for mobile ground robots as well, whenever a static environment is considered [Futterlieb et al., 2014]. With the help of our control strategy, the robot follows a spiral during the overall avoidance phase, which allows to guarantee that it is always provided a nonzero linear velocity. This in turn allows to reduce the risk of local minima as shown in [Cadenat, 1999] . It will then be possible to bypass static obstacles by defin-

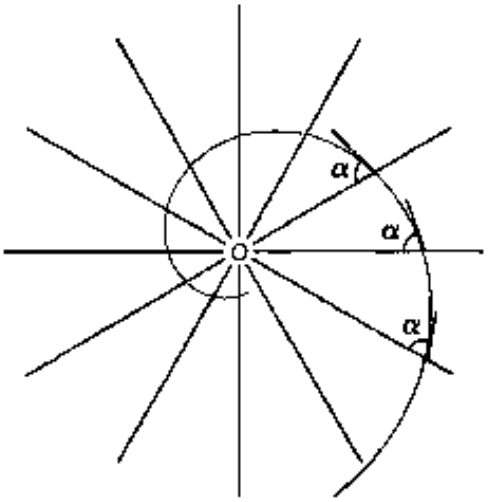


Figure 4.3.: Spiral Concept as presented in [Boyadzhiev, 1999]

ing an adequate spiral. To take into account dynamic ones, our idea is to modify on-line the shape of the spiral depending on the sensory data. In this way, the same control law will always be applied, only the path to be followed will be modified. We then obtain a general method, which can be easily adapted to both static and dynamic environments. After a brief description of this chapters outline, we will present the approach in more detail.

4.1.4. Structure of Chapter

The structure of the chapter is as follows. First, we present the developed avoidance method. We highlight our main contribution, namely the choice and the definition of the spiral, together with the design of suitable control laws. Primary focus will be the incorporation of moving obstacles into the approach. Then we present the results which have been obtained, both in simulation and on our experimental tested. Finally there will be a conclusion that outlines summarizes the presented work and gives a short outline of what is still to come.

4.2. Spiral obstacle avoidance

This section presents our avoidance technique. We first introduce the necessary definitions and the conventions which will be used throughout this chapter. Then, we detail how to determine the different parameters allowing to choose a suitable spiral. We will highlight different solutions depending on whether the environment is static or not. Finally, we design the control laws allowing to make the robot follow the spiral. We end

this section by showing both simulation and experimentation results on Air−Cobot.

4.2.1. Spiral conventions and definition

The following paragraph focuses on equiangular spirals, exhibiting the variables and parameters which will be used throughout this thesis. Let us consider figure 4.4.

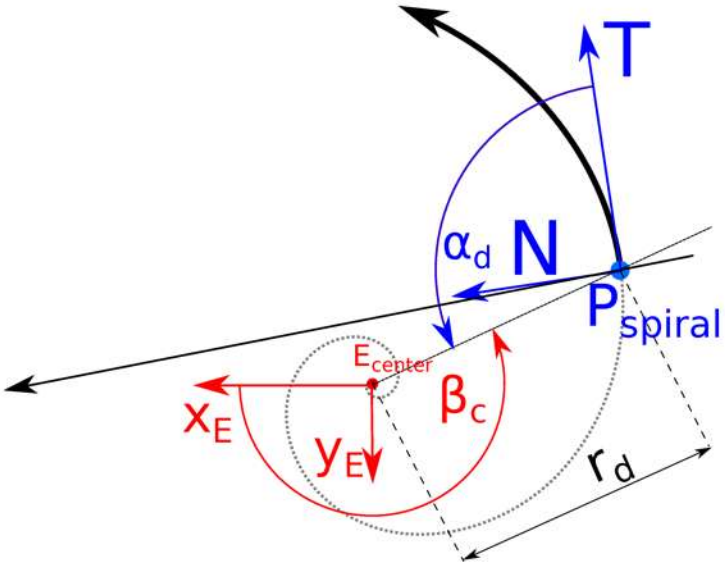


Figure 4.4.: Spiral conventions for an ideal case

We respectively denote by E_{center} and P_{spiral} the spiral center point (SCP) of the envisioned spiral and one of its point. We also introduce T and N which are respectively the tangent vector on the spiral at P_{spiral} and the normal vector to it. Finally, we attach to the spiral a frame denoted by $F_E(E_{center}x_E, y_E, z_E)$. In this frame, it can be shown [Boyadzhiev, 1999] that the equiangular spiral is defined by the following equation 4.1:

$$r_d = r_0 \times \exp^{\cot(\alpha_d)\times(\beta_0-\beta_c)} \tag{4.1}$$

where r_d denotes the distance between E_{center} and P_{spiral}, β_c represents the angle between x_E and $\vec{E_{center}P_{spiral}}$. Furthermore α_d is defined as the angle between the tangent T and this last vector. Finally, r_0 and β_0 corresponds to the initial values of r_d and β_c. We will see in the sequel how these different elements can be instantiated in the context of obstacle avoidance.

Analyzing equation 4.1 shows that the executed spiral depends on the value of parameter α_d:

- if $\alpha_d > \frac{\pi}{2}$ (case (a) of Figure 4.5: an outward spiral is performed and the robot pulls away from the spiral center point (SCP);

- if $\alpha_d = \frac{\pi}{2}$ (case (b) of Figure 4.5: a circle is obtained and the robot maintains at a constant distance to the SCP.
- if $\alpha_d < \frac{\pi}{2}$ (case (c) of Figure 4.5: an inward spiral is realized and the robot closes the distance towards the SCP;

This analysis is very helpful to build our avoidance strategy, if we imagine that our SCP lies on the obstacle (the chosen point will be defined later) as shown on figure 4.5. In such a case, we understand that the first and third cases offer very attractive motion behaviors. However, the second one should not be neglected because it may also be interesting if several obstacles lie in the robot vicinity (e.g. a cluttered environment) or if a mobile obstacle is expected to cross the vehicle path. In such cases, temporarily approaching the obstacles might help to find a safe navigation path towards the goal. This is even more clear when considering that the spiral will normally not be execute for more than 180°.

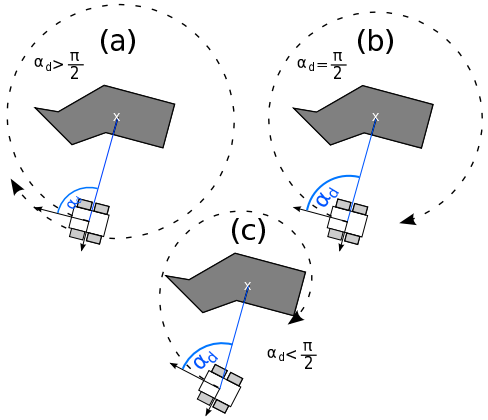


Figure 4.5.: Three schematics of spiral obstacle avoidance patterns

The equiangular spiral is then defined by four parameters r_0, α_d, β_c and β_0. Our objective in the sequel will then be to instantiate them the best way to obtain an efficient avoidance strategy.

4.2.2. Definition and choice of spiral parameters for the obstacle avoidance

Once an obstacle is detected, a dedicated algorithm based on LRF data allows to track it and evaluate its level of danger towards the robot (e.g.: obstacles that are in the path of the robot will get a higher level).

From this, our objective is now to instantiate the four parameters highlighted before: r_0, α_d, β_c and β_0. The first step to define them is to choose frame F_E, that is to impose the SCP and the orientations of the different axes.

Choosing the orientation of the frame F_E axes

In this work, $\vec{x}_E$ has been chosen to point towards the current target to be reached. It then always provides the direction to the goal, as this process is done on-line during the mission. Its computation requires to find the value of r_{target}, the distance between the spiral center point and the considered target. To determine it, we take into account that all targets (that is the successive checkpoints) are static and we use the available localization information. From this, we can deduce the following homogeneous matrices:

- cMt, the transformation from camera coordinate system to the coordinate system of the target provided by the pose estimation of VISP or our localization algorithm
- lMo, the transformation from laser coordinate system to the coordinate system of the obstacle provided by the Hokuyo
- rMc, the transformation from camera coordinate system to the coordinate system of the robot
- rMc, the transformation from laser coordinate system to the coordinate system of the robot

Now, the homogeneous matrix oMt can be expressed as follows:

$$oMt = inv(lMo) \times inv(rMl) \times rMc \times cMt \tag{4.2}$$

This relation expresses the transformation between the frames respectively linked to the obstacle and the target. We can then extract the missing parameter r_{target} from it. If all conventions are correctly applied, the y-parameter in oMt will be close to zero and r_{target} can be assumed to be equal to the x value.

Now, it remains to choose $\vec{z}_E$ and $\vec{y}_E$. We have chosen to orientate the first one skyward, while the second one is set by staying conform to a right hand coordinate system. Figure 4.4 displays this concept.

Remark: The so-determined value of r_{target} can be used in another manner. Indeed, in the case where the environment is sparse, it can be interesting to define the avoidance spiral so that its end-point lies on the checkpoint to be reached. It allows avoiding a switch back towards the go-to-goal controller, which in turn allows to reduce the problems of local minima. Of course, this option is not very relevant when too many obstacles lie in the scene or when the robot is too far from the goal. (see section 4.2.2).

Now that the axes orientations have been chosen, we have to focus on the two other issues: finding the SCP from LRF data and deducing the four above mentioned parameters.

Choosing the Spiral Center Point (SCP)

Our experiments have shown that the obstacle avoidance is more stable whenever the spiraling center point is chosen as the closest point of the obstacle. This way, obstacles with high aspect ratios (length to width) are avoided like one. Certainly, we have integrated several techniques during the course of this thesis that are specialized on these types of obstacles (bug algorithm, wall follower, etc.). However, in the interest of this contribution we wanted to show that the algorithm is still able to handle these types of obstacles (see video of [Futterlieb et al., 2014] for more information. Still, during our experimentation we have seen that for high aspect ratio obstacles (walls, etc.) this approach is less interesting. It is better to use another dedicated controller allowing to follow the obstacle geometry.

Initialize the values r_0 and β_0 of r_d and β_c

The following paragraph will explain how the most important parameters of our control strategy are initialized.

r_0 and β_0 will be set to the values of r_d and β_c at $t = t_0$, the time when the OA is first triggered. These values are provided by the obstacle detection module (see Section 2.2.3 on Page 40). Tracked obstacles then receive a state-vector containing vital information (α_c, β_{init}, r_c, information about the object's geometry and the object's estimated center point, E_{center}) for the obstacle avoidance. The figure 4.6 gives the definition of state vector. We denote r_c, as the measured distance between points E_{center} and P_{ref} (initial point of the robot, it could be P_{spiral}) and α_c as the angle between $\vec{x}_E$ and $\vec{E_{center}P_{ref}}$. r_c as well as α_c should be equal to their counterparts, r_d and α_d. In section 4.2.3 we explain how we can deal with this deviation.

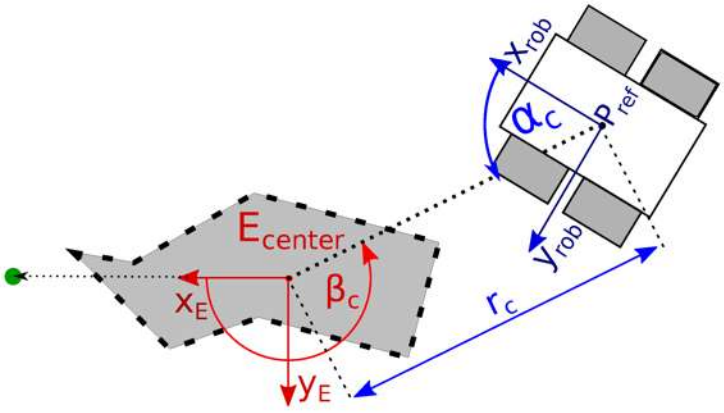


Figure 4.6.: Spiral convention in a real life situation

Choosing α_d

As previously mentioned, parameter α_d allows to tune the shape of the spiral (see figure 4.5). The way it is defined will depend on the dynamics of the environment.

For static sparse environments, a constant α_d is legitimate as there is no need to deform the followed spiral to guarantee non collision. In this case, we have chosen to perform a circle around the obstacle and we define:

$$\alpha_d = \alpha_{dstatic} \tag{4.3}$$

Two cases can be considered for the value of $\alpha_{d_{static}}$:

- a fixed value $\alpha_{d_{static}} = \frac{\pi}{2}$ (see subsection 4.2.1).
- α_d as a function of the checkpoint position. As mentioned earlier, the case where the environment is sparse (few obstacles) and where the robot is close to the checkpoint to be reached, it appears to be interesting to define α_d as a function of the current goal position. The idea is to define the avoidance spiral so that its end-point lies on the target. It allows avoiding a switch back towards the go-to-goal controller, which in turn allows to reduce the problems of local minima.

Let us recall the equiangular spiral equation (see)relation 4.1 from Page 129):

$$r_d = r_0 \times \exp^{\cot(\alpha_d)\times(\beta_0-\beta_c)}$$

Now, we assume that the current checkpoint is close to the robot (this can be easily deduced from the informations provided by the localization process). To make the spiral end-point coincide with the goal position, we replace r_d with r_{target}, the distance of spiral center point to target position, and β_c by β_{target} the angle at which the target is located in the frame of the obstacle. We get the following equation:

$$r_{target} = r_0 \times \exp^{\cot(\alpha_d)\times(\beta_0-\beta_{target})} \tag{4.4}$$

From this, we can calculate the corresponding value of α_d:

$$\begin{aligned}
&\frac{r_d}{r_0} = \exp^{(\cot(\alpha_d)\times(\beta_0-\beta_{target}))} \\
\Leftrightarrow\ &\cot(\alpha_d) = \ln\frac{r_{target}}{r_0} \times \frac{1}{(\beta_0-\beta_{target})} \\
\Leftrightarrow\ &\alpha_d = \cot^{-1}(\ln\frac{r_{target}}{r_0} \times \frac{1}{(\beta_0-\beta_{target})}) \\
&\alpha_d = \tan^{-1}(\frac{1}{\ln\frac{r_{target}}{r_0} \times \frac{1}{(\beta_0-\beta_{target})}})
\end{aligned} \tag{4.5}$$

Equation 4.5 allows us to calculate the α_d that will finally direct the robot towards the target, while avoiding the obstacle.

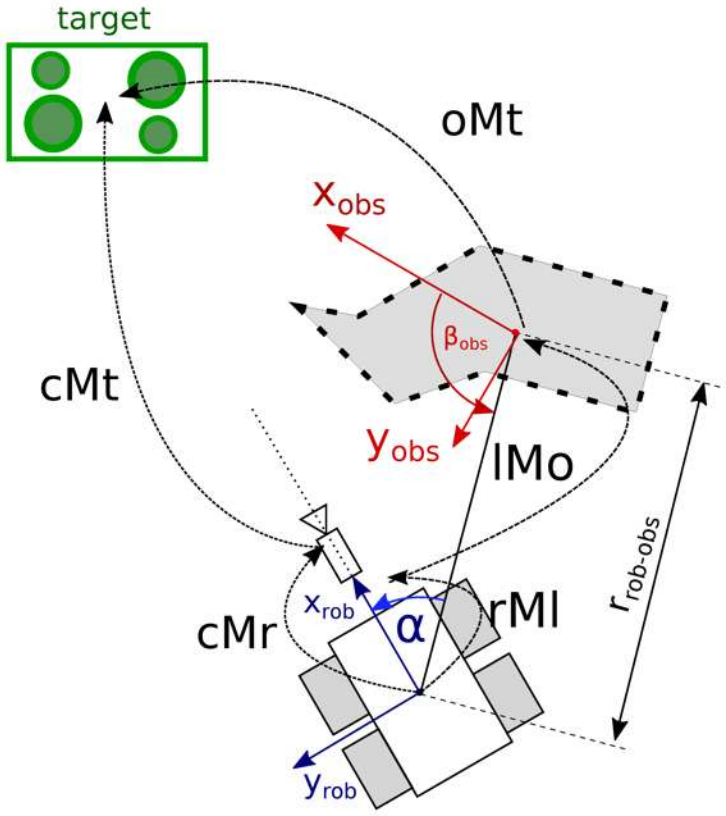


Figure 4.7.: Target and Obstacle Schematics

For dynamic or static, highly cluttered environments, α_d cannot remain a fixed parameter. Indeed, in such an environment, it appears natural that the spiral has to be modified because several obstacles can be encountered successively. For example, it is possible to start avoiding a first object and during the bypassing motion a pedestrian crosses the path of the vehicle. The initial avoidance motion has then to be adjusted. We tackle this problem in this part.

We propose to modify Equation 4.3 by adding a $\Delta\alpha_d$ which allows a variation around $\alpha_{dstatic}$ if the environment demands it.

$$\alpha_d = \Delta\alpha_d + \alpha_{dstatic} \tag{4.6}$$

In every instance along the spiral the algorithm is now adjusting α_d to its needs. Depending on the distance and obstacle's relative velocity in the robot frame adjustments to this angle are carried out. $\Delta\alpha_d$ is computed by calculating the derivative of e_r, $\dot{e_r}$. This derivative can be considered as a rough approximation of the obstacle relative velocity to the robot. Higher velocities will then lead to $\alpha_d > \frac{\pi}{2}$ which in turn will modify r_d in Equation 4.1 and lead to an outward spiraling behavior (red spiral in Figure 4.8).

We propose to define $\Delta\alpha_d$ as follows:

$$\Delta\alpha_d = \arctan(((-\widetilde{\xi}) \times a)^b) \tag{4.7}$$

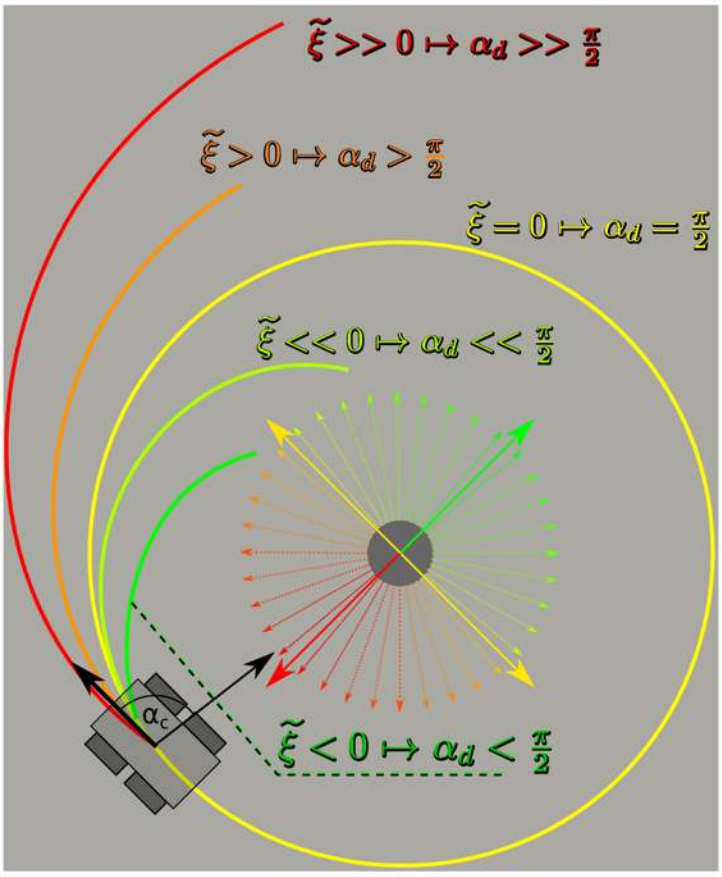


Figure 4.8.: Visualization of spiral obstacle avoidance extension for moving obstacles

where $\widetilde{\xi}$ is expressed as follows:

$$\widetilde{\xi} = \quad (\operatorname{sign} \xi) \frac{|\xi| - \xi_{min}}{\xi_{max} - \xi_{min}} \tag{4.8}$$

with :

$$\xi = \quad \dot{e}_r = \frac{de_r}{dt} \tag{4.9}$$

and

- $e_r = r_c - r_d$ is the error between the measured (or current) parameters (r_c) and the desired one (r_d).
- $b \in \mathbb{N} \hookrightarrow$ b has to be element of the natural numbers. $b = (2 \times k) + 1 \hookrightarrow$ b has to be an odd number
- $a \in \mathbb{R} \hookrightarrow$ a has to be element of the real numbers

Equation 4.6 abstracts our proposal to vary α_d depending on the derivative of the distance error, $\frac{de_r}{dt}$, which can be understood as an indirect estimate of the obstacle's

relative velocity towards the robot. We use the term indirect, because ξ does not measure the evolution of the distance towards the obstacle, but the evolution of e_r which implies the positional error of the robot. Assuming that, for static obstacles, the OA controller is keeping this error close to zero, high changes in the derivative of e_r must come from the obstacle, thus implying that the obstacle is indeed moving. We deduce $\widetilde{\xi}$ from ξ in Equation 4.8. As a first step, $\widetilde{\xi}$ is normalized so that it varies between -1 and 1 . This procedure allows to insure that later computations are more generic and can be applied to other robotic platforms. Parameters ξ_{max} and ξ_{min} are set as follows: ξ_{min} is fixed to the speed at which obstacles are considered to be static (about 10% of the average human walking speed) and ξ_{max} is dependent on how agile the robotic platform is (we have chosen 300% of the human walking speed). The second step is to use the Arc Tan function to define $\widetilde{\xi}$. This function was chosen for its handy property of ranging from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. This way α_d (Equation 4.6) is limited to zero and π (see Figure 4.9).

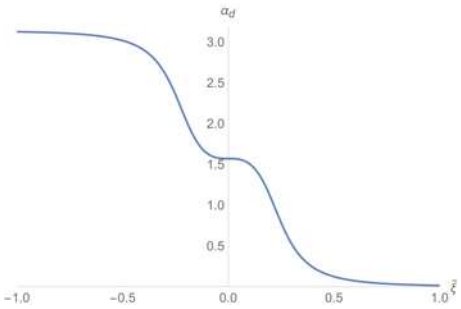


Figure 4.9.: Graph visualizing relationship between $\widetilde{\xi}$ and α_d (blue,uninterrupted line $\hookrightarrow$ Air−Cobot configuration)

Figure 4.9 displays the chosen evolution of α_d with respect to $\widetilde{\xi}$. An $\widetilde{\xi}$ below zero will lead to an increase of α_d from the default of 90° up to a maximum of 180°. Obviously, it is up to the set-up of the robotic platform how far this angle can actually be applied. The Air−Cobot is equipped with a 270° Hokuyo LRF in the front and in the back which enables it to have a 360° Field of View (FOV), thus supporting the full range of α_d. In the case where the FOV is lesser, the evolution of $\Delta\alpha_d$ in Equation 4.6 has to be limited accordingly. This can either be done by limiting how much of the computed α_d is applied on the control law or by carefully choosing the parameters a and b so that the maximal bounds on $\Delta\alpha_d$ are not reached. By choosing $a = 4$ and $b = 3$ the ArcTan function is modified in such a way that at the extrema of $\widetilde{\xi}$ (-1 and 1) the extrema of α_d are reached (0 and π). Finally, a sign function was added to take into account the sense of the obstacle movement towards the robot. The whole procedure is summarized on algorithm 4.1.

4.2.3. Robot control laws

In the following section, the control laws allowing the robot follow the chosen spiral are presented. We consider the model of the robotic system which has been presented in

Algorithm 4.1 Checklist for OA

1: **procedure** DETERMINE OPTIMAL α_d ▷ Called by OA
2: Calculate $\widetilde{\xi}$ with the help of the derivative of e_r
3: Compute $\Delta\alpha_d$ following Equation 4.7
4: If necessary, adjust $\Delta\alpha_d$ according to your robot's limitations
5: **end procedure**
6: The new α_d is used to compute a desired distance from the obstacle r_d as shown in Equation 4.1
7: A PID controller is now assisting the minimization of e_α and e_r (Equations 4.14, 4.15, 4.16)

the previous chapter and recalled on figure 4.10.

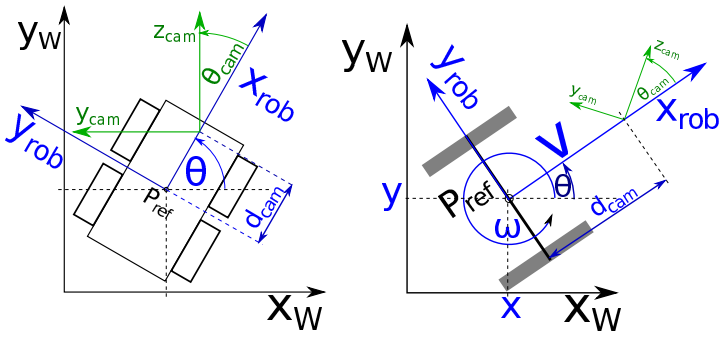


Figure 4.10.: Real life robot model with application point for control inputs

The robot is controlled using two control inputs: its **linear** velocity ν_{OA} and its **angular** velocity ω_{OA}. We present their design hereafter.

The goal is to make the robot follow the chosen spiral. To do so, it is necessary to guarantee that the two errors, distance and angular errors, defined here-below vanish:

$$e_\alpha = \alpha_c - \alpha_d \tag{4.10}$$

$$e_r = r_c - r_d \tag{4.11}$$

To do so, we have chosen to impose a suitable velocity profile for ν_{OA} . This solution guarantees that the robot will always be provided a nonzero linear speed during the avoidance phase. In this way, problems of local minima will be significantly reduced. The angular velocity will be designed to regulate e_α and e_r to zero.

The linear velocity profile - ν_{OA}

We denote by ν_{OA} the linear velocity applied to the robot during the avoidance phase. In the publication [Futterlieb et al., 2014], we have imposed a constant (non-zero) value. However, this solution has proven to be limited when tight bypassing motions were to

be performed. To improve the control law efficiency, we have defined ν_{OA} as a function of e_α as shown on figure
4.11:

$$\nu_{OA} = \frac{1}{\exp^{|e_\alpha|}} \times \nu_{OA_{max}} \tag{4.12}$$

$$\tag{4.13}$$

where $\nu_{OA_{max}}$ is the maximum value allowed for the linear speed for obstacle avoidance. It has to be fixed by the user prior to the navigation.

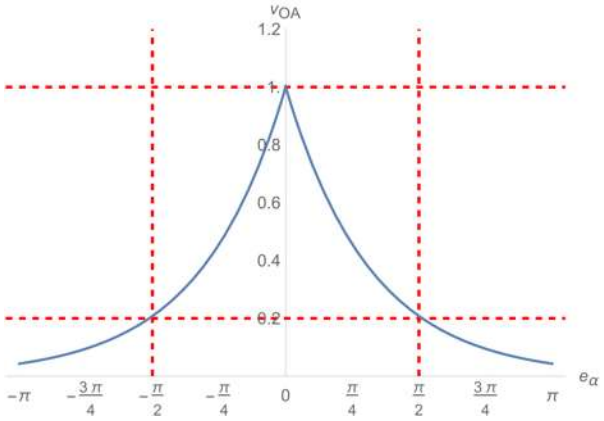


Figure 4.11.: Dependency of $\nu_{OA_{max}}$ on e_α

In this way, we enable the robot to decrease its linear velocity when e_α becomes too high, making sharp turns easier. On the contrary, if this error is small or around zero, the vehicle can move more rapidly (see figure 4.11).
Remark: We could also have tried to integrate e_r into Equation 4.12 together with e_α. Considering the two errors to be regulated to zero in the linear velocity profile is not interesting because it significantly increases the risks of local minima when they are about to vanish. Another important point is that e_α appears to be the most relevant parameter to define ν_{OA} because one of the goal is to ease the execution of tight turns, which requires to limit the linear velocity value.

Control of the angular velocity - ω_{OA}

Now, let us design ω_{OA} so that e_r and e_α vanish. We propose the following PID control:

$$\omega_{OA} = \text{sign}\,(SOM) \times (\Psi_{e_\alpha} + (\Upsilon_{e_r} \times \lambda)) \tag{4.14}$$

$$\Psi_{e_\alpha} = K_p \times e_\alpha + K_i \times \int_{t_0}^{t_c} e_\alpha \, dt + K_d \times \frac{de_\alpha}{dt} \tag{4.15}$$

$$\Upsilon_{e_r} = K_p \times e_r + K_i \times \int_{t_0}^{t_c} e_r \, dt + K_d \times \frac{de_r}{dt} \tag{4.16}$$

where SOM stands for "Sense Of Motion". For *ccw* (counter-clockwise) movements, this parameter will be $+1$ and -1 for *cw* (clockwise) motions. More precise information about its choice will be given in the next paragraph. t_c and t_0 are respectively the current time and the instant at which the OA has been triggered. Parameter λ is a strictly positive scalar allowing to control the weighting between the first and second tasks. Different solutions have been tested. At first, we have chosen to give a greater priority to the angular error regulation ($\lambda < 1$). However, a constant value for this gain rapidly appears to be limited because it follows that the priority cannot ever been changed. To allow these necessary changes, we propose the following evolution for λ[1]:

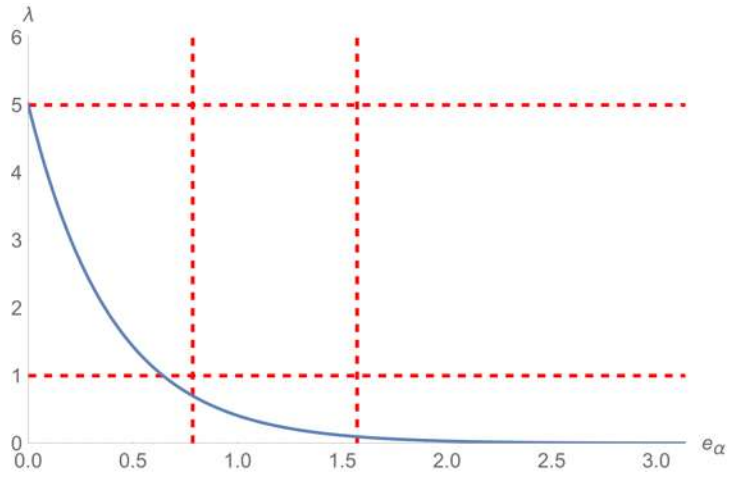


Figure 4.12.: λ as a function of the angular error e_α

In this way, the smaller e_α becomes the more λ will increase which will in turn lead to a rise of the weighting of e_r in Equation 4.14.

Choosing the sense of motion (SOM) and switching back to a Go to Goal behavior

Although these aspects belong to the supervisor and the decision process, we have chosen to present them in this chapter, as it is directly related to the chosen obstacle avoidance technique. We first present the choice of the SOM before giving some information about the guarding conditions.

The SOM: The first developed solution has consisted in making the SOM dependent on the position of the target towards the robot and the obstacle [Futterlieb et al., 2014]. In this way, it allows to reduce the tight turns at the start of the avoidance movement. Figure 4.13 displays how the SOM was chosen in a first draft of the approach: counter-clockwise if $P_{targetA}$ is the checkpoint; clockwise if $P_{targetB}$ is the checkpoint.

[1]We nonetheless guarantee that λ remains always positive.

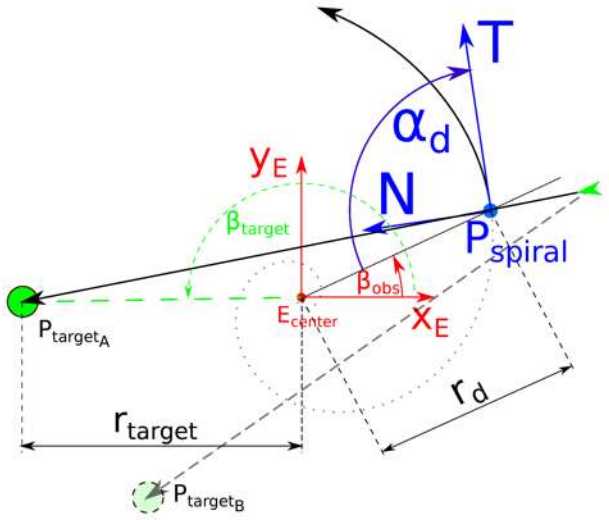


Figure 4.13.: Visualization of the first draft for the SOM determination

However this solution is mainly valid for a static environment. When dynamic obstacles are encountered, it is necessary to adjust the SOM with respect to the obstacle motion. We recall that whenever an obstacle is encountered, its parameters (frame orientation, r_c, α_c, β_c) are determined by the perception process. We propose to track the change in angle β_c from its earliest detection to the instant when the OA is engaged. If the obstacle is static, then the variation of this angle will remain negligible. If not, $\dot{\beta}_c$ will change substantially (see Figure 4.14 upper example where only the robot is moving).

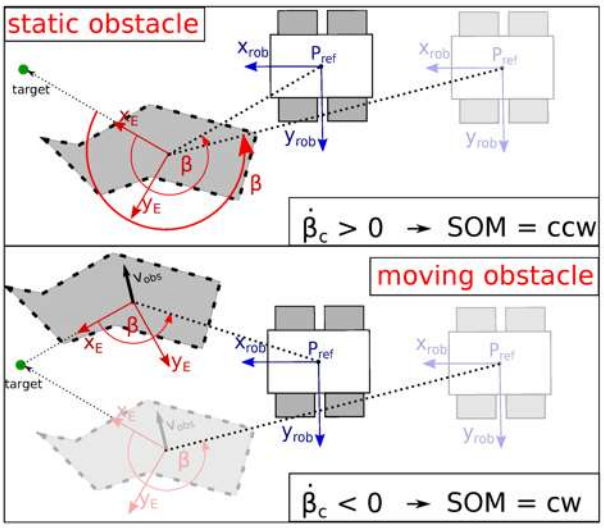


Figure 4.14.: Visualization of SOM determination

Knowing the evolution of β_c will allow to set the SOM in much easier manner. Indeed, if $\dot{\beta}_c < 0$, it means that the relative orientation (towards the robot) of the velocity vector

of the obstacle is close to the robot y-axis (the obstacle passes from the left to the right). On the contrary, if $\dot{\beta}_c > 0$, the obstacle is moving from the right to the left.

Therefore, we propose to choose the sense of motion in the following manner:

Algorithm 4.2 Decision tree for choosing SOM

```
1: procedure DETERMINE SOM                    ▷ Called by Obstacle Detection
2:     if |β̇_c| < threshold then choose SOM according to the method proposed in [Futterlieb et al., 2014]
3:     else
4:         if β̇_c > 0 then optimal SOM is ccw
5:         else optimal SOM is cw
6:         end if
7:     end if
8: end procedure
```

Switching between avoidance and goal controllers Now, we focus on the guarding conditions allowing to switch between the avoidance and goal controllers (go-to-goal or visual servoing). These conditions are essential to reduce local minima problems. Two cases are possible:

- *Transition from go to target controllers to avoidance one:* We consider the case where the obstacle is located on the robot trajectory and is considered to be a danger for it. At first, we have chosen to engage the bypassing motion when the distance falls below a predefined threshold denoted by $r_{OA-trigger}$. But, as shown below, it will be necessary to improve it.

- *Transition from avoidance to goal controllers:* We consider the case where it is necessary to stop bypassing to go towards the goal. This happens when the level of danger of the obstacle decreases. Thanks to the chosen orientation of frame F_E, a reliable guarding condition is given by the angle $\beta_c = (\vec{x}_E, \vec{N})$. As x_E is orientated towards the target, it provides the direction of the goal. From this, we can deduce that a safe spot is reached when β_c becomes greater than $\frac{3}{4}\pi$ (ccw motion) or gets below $\frac{\pi}{2}$ (cw).

However, there still remains some situations where local minima can occur. Figure 4.15 gives such an example[2]. The bypassing procedure has been engaged and the condition to stop it ($\beta_c > \frac{3}{2}$ or $\beta_c < \frac{\Pi}{2}$) is fulfilled for robots A and B. However, if nothing is done, the guarding condition to switch to obstacle avoidance will be triggered again, because the vehicles are too close to the object. To prevent this from happening, we propose to check, not only the distance to the obstacle but also the distance to the target. In

[2]For demonstrative reasons, we consider the case of an outward spiral. But the same reasoning could be done with a circle.

other words, we modify the guard condition to switch to obstacle avoidance as follows: $r_c < r_{OA-trigger}$ and $r_{tar_c} < r_{OA-trigger} + \varepsilon$ where ε is a safety distance depending on the robot dimensions. In this way, it will be possible to reach a target, even if it is located close to an obstacle (provided that the remaining space is sufficient to guarantee non collision).

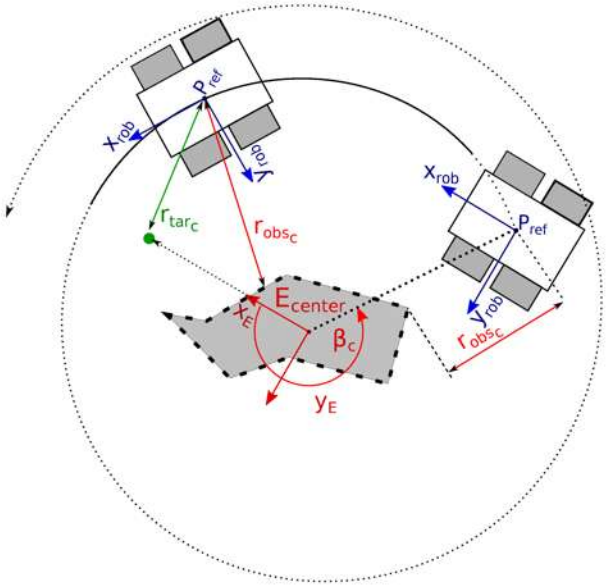


Figure 4.15.: Example of when to overwrite obstacle avoidance.

4.3. Simulation and experimental results

We have validated the proposed method thanks to both simulation and experimentation tests. We showcase hereafter a selection of the obtained results.

4.3.1. Simulation with static obstacles

Simulation environment

The middle-ware used on the Air−Cobot is ROS (**R**obot **O**perating **S**ystem). Thus, to stay close to the testbed, we have chosen to conduct most of our simulations using Gazebo, as it is the most recommended simulator. As the available functionalities allowing to display our results have not been convincing, we have chosen to record the evolution of the necessary variables using ROS-bags. A ground truth Gazebo plug-in has also been used to capture the robot and obstacle positions and orientations. Those recordings are then read into MATLAB and plotted as trajectories in a 2D-plot. An example of the simulation environment is provided in Figure 4.16.

Since there Air−Cobot is a prototype naturally its model does not have a template in Gazebo. To allow for fast implementation, we have used a the CLEARPATH Husky

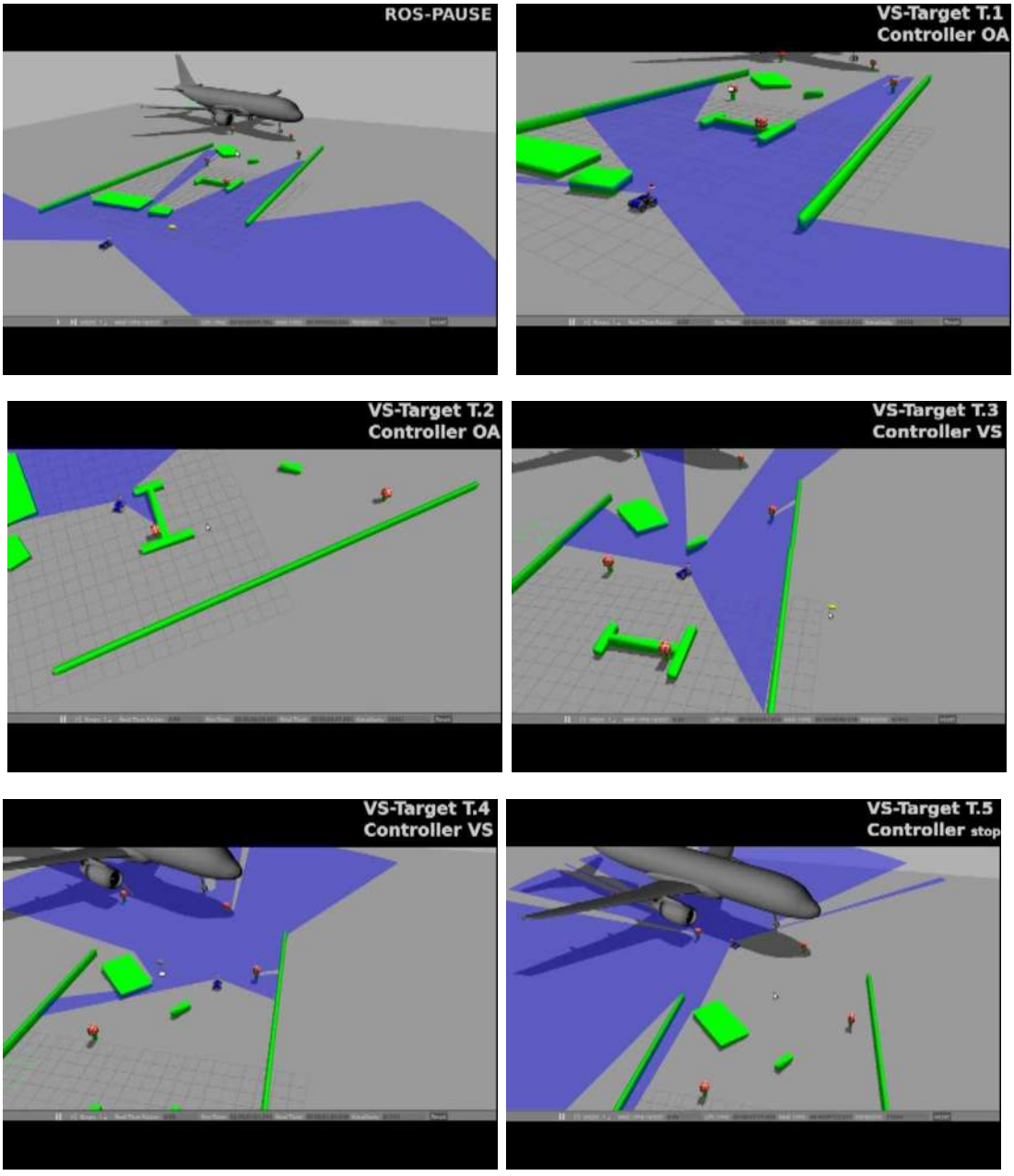


Figure 4.16.: Extract of video sequence [Futterlieb et al., 2014] - static obstacles

robot model, because its dimensions (although a bit smaller) and general features (four wheel skid steering drive) are very similar the once of our platform. Now we focus on the scenario which have been considered to validate our approach.

Chosen scenarii and their respective results

We have evaluated our avoidance method in two separate steps. At first, we have chosen a basic avoidance case where the goal is to make the robot bypass a sole obstacle using an adequate spiral. Then, we have considered a more complex scenario corresponding to the so-called approach phase. We now focus on these two cases.

Avoidance of one single obstacle In this case, the environment is cluttered with only one obstacle. No goal has been defined. Our robot is solely expected to avoid the object by following a predefined spiral which angle α_d has been taken greater than $\pi/2$ to guarantee non collision. Air−Cobot is initially at the position $(0, 0)$ and it starts moving towards the obstacle due to a positive linear velocity. The avoidance phase starts when it enters a dangerous zone around the obstacle (at about 2 meters from it). At this time, controllers given by Equations 4.14 and 4.1 are applied to the robot. Figure 4.17 displays successive poses of a robot (blue rectangles) going from left to right (blue path) and also the static obstacle (red rectangle). As we can see, the obstacle is successfully avoided by the robot executing an outward spiral motion around it, as expected.

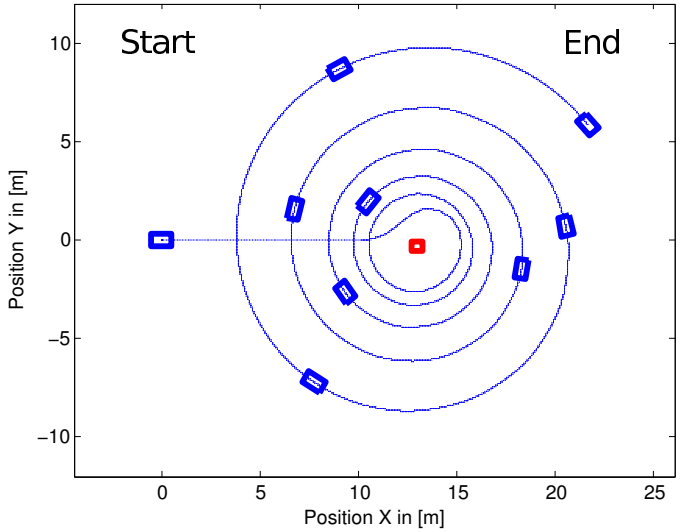


Figure 4.17.: Proof of Concept of an outwards spiraling obstacle avoidance

Long range navigation with static obstacles Our goal is to test our approach in the case of a long range navigation in an environment cluttered with static obstacles. More precisely, we consider the autonomous approach phase where the robot is expected to reach a close neighborhood of the aircraft, starting from its charging station. Figure 4.18

shows this scenario (the Airbus A320 is materialized by the gray area on the far right side of the figure). All of the obstacles (see red rectangular polygons in Figure 4.18) are static and non-occluding objects.

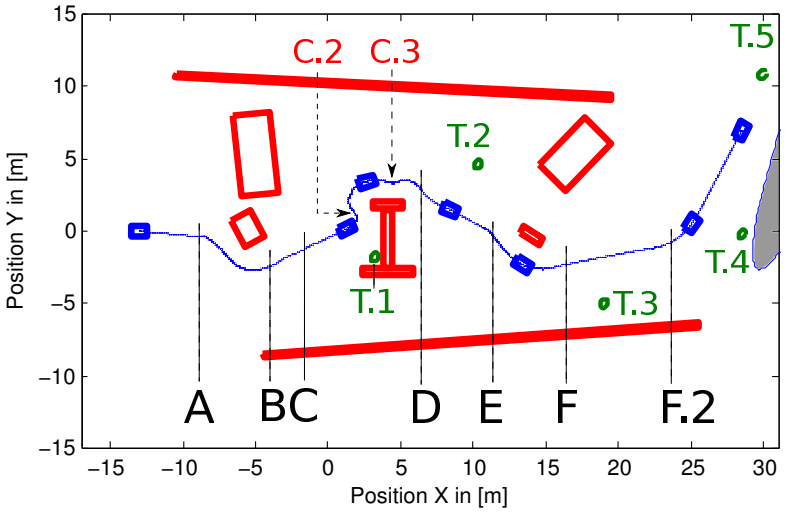


Figure 4.18.: Avoidance Experiment for static Obstacles

In this simulation, the robot is expected to reach target T.5 starting from its initial position. To do so, we have used the navigation strategy explained in the previous chapters. This means that we have first defined a topological map (very basic in this case as it only contains the successive targets to be reached, namely T.1, T.2, ..., T.5). We also need to choose a controller allowing to reach each target and we have selected an IBVS (we could of course have performed the same simulation using a GTG controller). The avoidance motion is performed due to our previous methods based on spirals. The obtained results are displayed in Figure 4.18.

First, we can observe that the navigation task is correctly performed: T.5 is reached and no collision has occurred. This means that the robot has been able to reach the aircraft nose which is considered to be the initial position to start the aircraft pre-flight inspection) as well as the first check point (front side of the A320 right jet engine). Now, let us take a closer look to the robot trajectory. We can see the controller sequencing. IBVS is applied from Start to A, shortly from B to C, from D to E and finally from F through F.2 to the final position. In a similar way, when an obstacle is detected, the Spiral Obstacle Avoidance (SOA) controller is triggered at a distance $r_{OA-trigger} = 2.5$ meters. The supervisor then stops this controller when a safe passage towards the current landmark is possible. In this simulation, the SOA is used from A to B (counter-clockwise), from C to D (clockwise) and from E to F (counter-clockwise) with an α_d of $\frac{\pi}{2}$.

The second obstacle is particularly interesting as it offers the robot a 'convenient trap'. To overcome it, the spiral center point has been recomputed twice during the avoidance

(C.2 and C.3). For the other two obstacles, the bypassing motion is easier and a re-computation of the SCP is not necessary. As we have stated in the introduction, this validates that our approach can handle wall like obstacles if needed. These results show that our approach is meaningful. To go further in our validation, we have also displayed the control inputs sent to the robot.

The linear and angular velocities computed with equations 4.12 and 4.14 are plotted in Figure 4.19.

As for the pan-platform, a sole positioning control was available through ROS. The corresponding position of the pan tilt unit is shown on Figure 4.19.

The time intervals in which the robot is using the visual servoing controller ($\simeq$1.8 m/s) and the obstacle avoidance controller ($\simeq$0.4 m/s) clearly appear, showing that they are correctly sequenced depending on the environment. We believe that the variations observed in the evolution of the different variables could be reduced with a better prediction of the obstacles shape, thus, smoothing the robot trajectory.

4.3.2. Experiments in dynamic environment

We have also experimented the avoidance method and the overall navigation strategy on the Air−Cobot platform. We first present the environment before focusing on the scenario and on the obtained results.

The environment

As shown in Figure 4.20, we have chosen the an outdoor environment for our tests. All the experiments have been performed on the CNRS-LAAS parking lot. In order to show the potential of our approach none of the obstacles are not a priori known. Same applies for their number. Most of them are static (mainly parked cars). The dynamic ones are trolleys which are pushed by a person, as it is illustrated in the figures 4.21 and 4.21.

Scenario and results

The goal of the mission is to reach a position located at 40 meters along the x-axis (red dot), starting from the initial position $(0, 0)$ while bypassing the obstacles (see Figure 4.23). In this part, we then focus on the avoidance motion. The first two obstacles are static (a and b) and convex. Note that their geometries are different, which will demonstrate the ability of our solution to cope with various objects shape. Following these two impediments, are two dynamic obstacles (c and d). Their avoidance will highlight the efficiency of the SOM choice technique and the interest of making α_d vary.

Figure 4.23 visualizes the path the robot is taking through the unknown environment.

In this case, a Go To Goal (GTG) controller is used to reach the desired position. Again, the avoidance motion is performed due to our previous methods based on spirals.

As previously, when an obstacle is detected, the Spiral Obstacle Avoidance (SOA) controller is triggered at a distance of $r_{OA-trigger}$ of 2.5 meters and the robot stops the SOA controller when a safe passage towards the current landmark is possible. The obtained results are shown on figure 4.18.

In order to give a more in-depth view of the avoidance strategy, Figure 4.24 displays a close-up of the SOA about obstacle a containing the most important parameters. This figure shows the controller sequencing and the efficiency of the guarding conditions. When the risk of collision increases steadily (the distance falls below $r_{OA-trigger}$), the SOA controller is engaged. The robot starts the bypassing motion and converges towards the chosen spiral. Once $\beta_c > \frac{3}{2}\pi$, the supervisor switches back to the GTG, leading to a natural trajectory for the platform.

Figure 4.25 presents the linear and angular velocities sent to the robot. As one can see, the different controllers are correctly sequenced by the supervisor. We can note that, when the avoidance phase starts, the linear velocity drops from $0.6m/s$ to nearly zero. At the same time the angular velocity increases to make the robot turn to guarantee non collision. The SOM is defined as shown previously and allows to obtain a smooth trajectory. This behavior is consistent with the designed controllers. At the end of the its mission the go-to-angle controller (GTA) is selected and corrects the robot's orientation (see video) at the final position. This explains the final spike of ω around the 110 second mark.

We have also deeply analyzed the evolution of α_d. As explained before, its variation is governed by parameter $\widetilde{\xi}$ (see equation 4.7). Its plot is given in Figure 4.26. This latter shows how often $\widetilde{\xi}$ has a non-zero value. For static obstacles (a and b), we expect to see this phenomenon occur very rarely. And this is exactly what is observed, as, during the two static objects avoidance, it happens three times which can be explained with sensor noise. However, when bypassing the two dynamic obstacles (c and d), none-zero values for $\widetilde{\xi}$ are much more frequent. Six changes are recorded for object c and for d this number goes up even further (10+). This shows that the avoidance motion was very complicated. Indeed, our video shows that the obstacle d was not a cooperative one. The person pushing the obstacle kart was actively trying to "collide" with the robot, thus generating a much stronger reaction. Some negative values have been also observed when the objects are going far from the robot (see Video).

Even though the presented experiment might not seem to be likely (airport staff might not ever try to collide with the robot), it presents a general example of what a stress test should look like. The fact that Air−Cobot did not collide with any of the obstacles can be taken as a successful carried out experiment of what was theorized at the beginning of this chapter.

4.4. Conclusion

In this chapter, we have presented a new reactive obstacle avoidance approach. This work has been inspired from articles explaining the peculiar behavior of moths and

other bugs around natural and artificial light sources during the night [Himmelman, 2002] and [Boyadzhiev, 1999]. These insects use the angle towards the source as means of navigation and they perform a spiral whose center is the considered light source. Following this idea, we have proposed to guarantee non collision by following a suitable equiangular spiral centered on the obstacle. Our main contribution has been the choice of the different parameters allowing to define the spiral, namely its center and its angle α_d. We have more particularly analyzed the role of this angle, showing that a constant value can be sufficient in a static environment. Then, we have extended this method by proposing a suitable variation for α_d in order to take into account dynamic obstacles. This variation relies on a rough estimation of the relative velocity of the object with respect to the robot. We have finally proposed a control law allowing to make the robot follow the chosen spiral. We have tested this approach first on a simulated system and then on the real one. The obtained results are satisfying and validates the method. In the future, we plan to extend this method following different axes and even geometric shapes. First of all, we will reduce the high variations observed in the velocities. For a first start, the introduction of rate limiters on the acceleration seems to be a sensible approach. Second, we will improve the guarding conditions and the transitions between controllers for a better obstacle anticipation and a smoother robot trajectory. Finally, other non-linear control laws will have to be developed to improve the robot behavior.

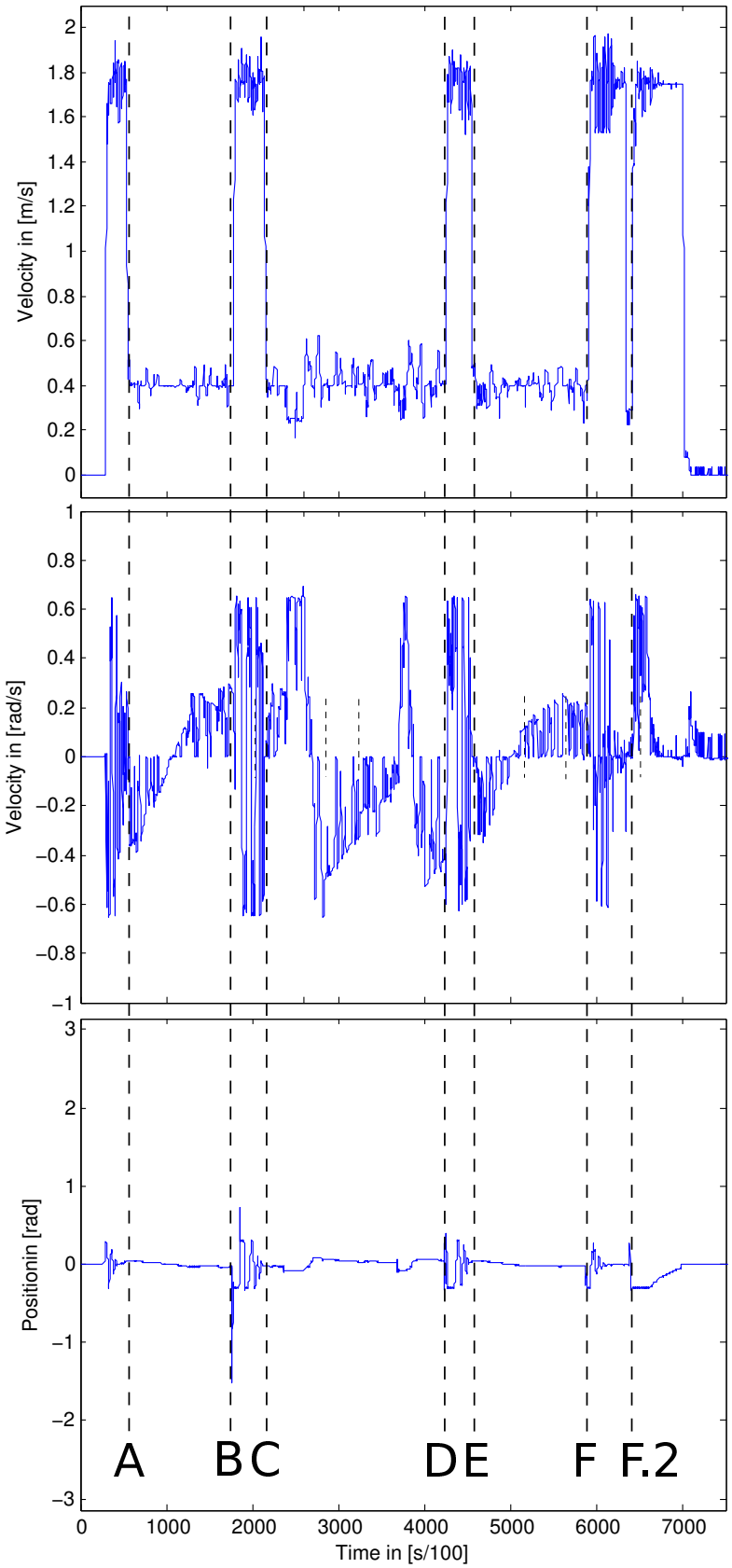


Figure 4.19.: Robot linear and angular Velocities and Camera Angle

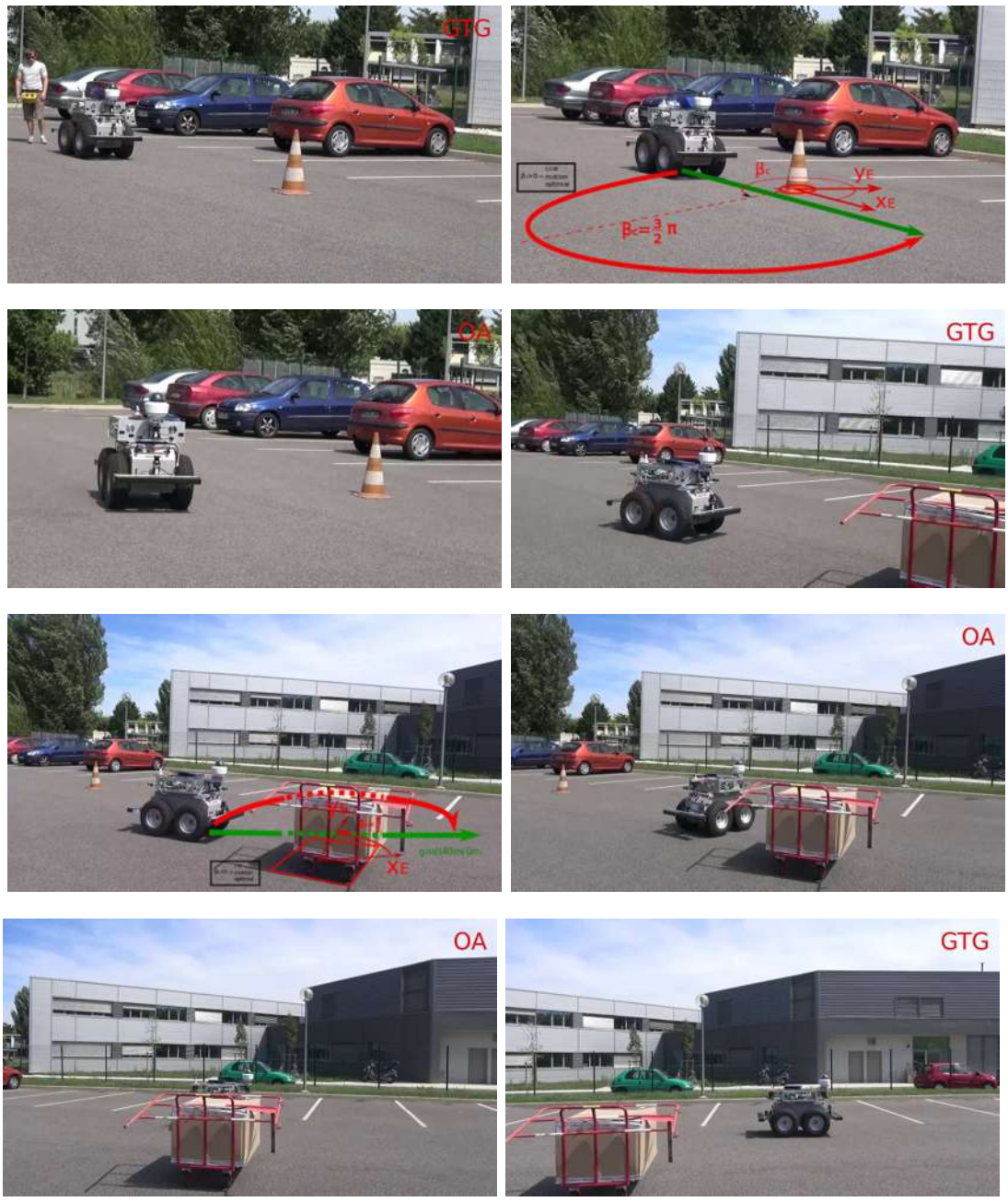


Figure 4.20.: Extract of video sequence - static obstacles

Figure 4.21.: Extract of video sequence - dynamic obstacles (a)

Figure 4.22.: Extract of video sequence - dynamic obstacles (b)

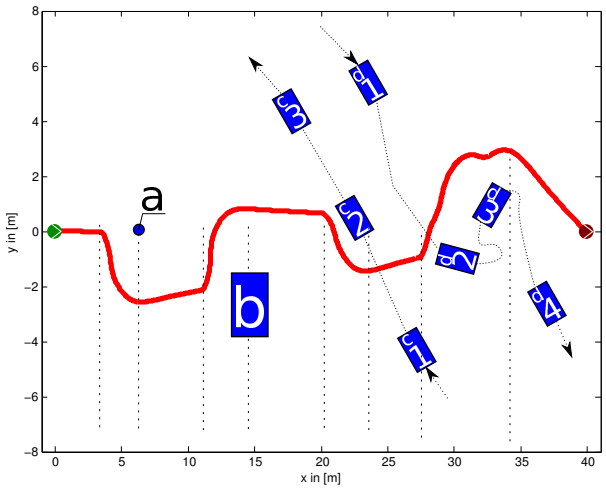


Figure 4.23.: Robot trajectory.

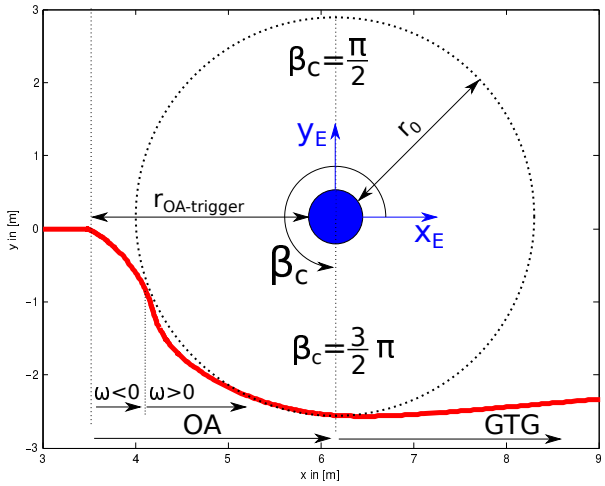


Figure 4.24.: Successive robot positions - Close up of first OA

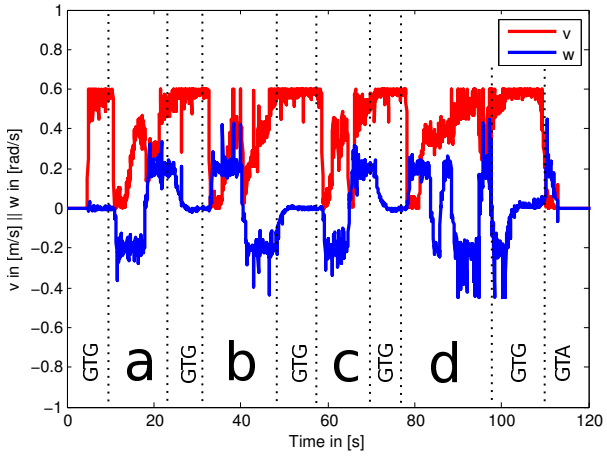


Figure 4.25.: Linear and angular velocities

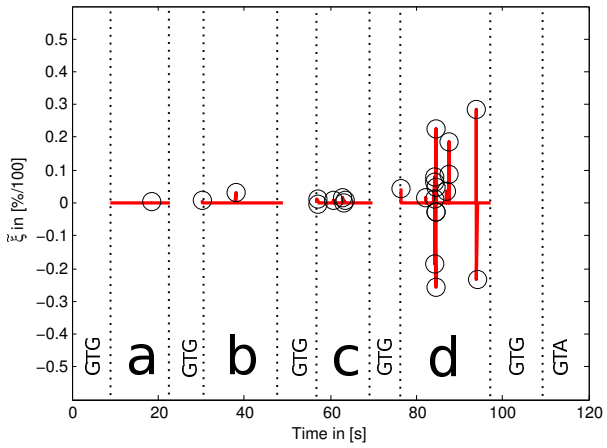


Figure 4.26.: $\widetilde{\xi}$

5. Conclusion

Contents

The following chapter provides closure for this thesis. First, a resume of the main results is given separately for each topic of this work. These topics were: the strategy of long range navigation, the accurate sensor based control of the robot around the aircraft and an adaptive obstacles avoidance. Furthermore, we focus on current challenges faced by these topics (and how they can be overcome).

5.1. Resume of contributions and project summary

In this manuscript, we have presented a framework that enables a mobile platform to perform pre-flight checks on commercial aircrafts. As mentioned in the introduction, the navigation process can be divided into six sub-processes (perception, modeling, planning, localization, action and decision) and this thesis presents a framework that can handle each one of them. Diving into further details, we have provided a platform that is able to **perceive** its environment with the aid of its many sensors. We have described cameras that help to perceive the robot's surroundings passively, laser range finders that basically fulfill the same purpose in an active fashion. Additionally, we have mentioned secondary sensors like GPS, wheel tick sensors and contact bumpers. **Modeling** was achieved with the help of a topological and two metric maps. The first one was built from targets on the aircraft and its immediate surroundings, while the two other ones were constructed with the aid of laser range finders and odometry while executing the mission. In addition to that, obstacles have also been modeled with the data provided from the perception sub-process. **Planning** enables the state machine to give the order in which the mission is executed. So far, it still has to be given prior to the mission by the operator in the form of a list of successive targets. However, the current framework is also able to receive this planning on-line through ROS topics and in the future this will be the general case. As for the sub-process **localization**, it has been instantiated with several methods, allowing to obtain either an absolute or a relative pose of the robot. These methods are listed below:

- Absolute localization with help of a **G**lobal **P**ositioning **S**ystem sensor;
- Relative localization towards the aircraft with the help of both basic wheel and visual odometry (SVO), IMU data, ORB-SLAM techniques, Point Cloud Matching.

The odometry drift was reduced in a first attempt with the help of the IMU sensor. Later we have shown that by using ORB SLAM and different secondary localization informations fused within an extended Kalman filter provided by ROS, allowed us to improve the accuracy when reaching certain key positions. The **Action** sub-process is made of several behavior-based controllers allowing to safely navigate throughout the airport. These controllers can be either aiding the robotic platform to reach a goal or avoid hazardous areas, obstacles. The **decision** process is instantiated by a supervisor which allows to choose the right controller among all the available ones and to monitor the execution. Furthermore, this finite-state-machine will also have a sub module which monitors the execution of said controllers. In order to decentralize part of the decision

process the majority of available controllers are able to take minor decisions (current speed limits, distance to obstacles, etc.) without consulting the supervision module.

During the course of this manuscript two additional contributions were highlighted. The first one is related to the multi-visual servoing control which has been designed to precisely position the robot on each checkpoint. We have discussed two popular approaches in the domain of visual navigation and have presented a framework for selecting the most appropriate technique for each situation. In addition, we have proposed a combination of both Image-based and Position-based visual servoing in a framework that: (i) reconstructs the features if they disappear thanks to a virtual image which is updated with several sensors used simultaneously, (ii) is able to find back these features and (iii) even navigate using only the reconstructed values. The second above mentioned contribution and "the most important one" is related to obstacle avoidance and to the supervisor. Here, we have provided a novel approach towards reactive obstacle avoidance in dynamic environments and new ideas for robust guard conditions. The approach is generic in the way that obstacles, whether they are dynamic, static, known, unknown, can be handled through the same method and within a procedure that allows a safe navigation.

The whole strategy has been validated through simulations and experimental testing. Although the experimental results presented in this work were obtained at LAAS-CNRS, all of the shown tests have also been performed at the Airbus industrial site at Toulouse Blagnac in order to finalize the proof of concept. Due to the confidential nature of these tests, this part of the work will be presented only during the denfense. The obtained results have shown that the proposed framework can be used to guide a platform on the airports tarmac and in hangars, in the presence of static and moving obstacles. Finally ideas were provided to improve the robot behavior towards smoother trajectories and handling of forbidden zones.

In the following section we want to explore further improvements and prospects for the project. The section will be more hardware related since software related improvements have already been discussed in their respective chapter.

5.2. Future work and prospects

This work has been realized in the context of a highly technological project. Our main prospects are then related to hardware and software aspects in order to improve the overall platform. From a scientific point of view, some aspects could be improved. For example, it would also have been interesting to evaluate the 2.5D visual servoing approach, in addition to position-based and image-based control laws which have been compared. The avoidance control laws relies on simple PID control laws. It could be interesting to design more advanced approaches and to propose a complete analysis of the closed-loop system. Now, from a more technological point of view, we present

hereafter some expected improvements of the Air-Cobot platform.

5.2.1. Addition of a graphic card

So far, the navigational computer of the robot is not equipped with a **G**raphic **P**rocessing **U**nit (GPU). This means that all calculations have to be done on the sole CPU which is not optimal, especially if some functions have to be parallelized. SLAM is one of these modules that would largely profit of parallelization. Same goes for feature tracking and other computer vision related functions. Thus it appears very interesting to add a GPU to the robot to relieve computational load from the CPU. This seems like a simple change that could be carried out right away. However, several of the algorithms will have to be modified to support this new system.

The current version of the computer system integrated on the Air-Cobot platform has been optimized to the point that finding physical space on the computer for extra hardware components is a challenging task. As for now, the Air-Cobot platform is equipped with two PCs. One respectively dedicated to navigation and the other for inspection. A possible option would be to add a GPU on the inspection computer, as non destructive testing functions require computer vision algorithms. Computational expensive calculations could then be sent onto this GPU, since it is not used during navigation (obstacle classification, target tracking, etc).

5.2.2. Using the robot as a platform for autonomous drones

During the project we have realized that having all the inspection equipment on the robot is not always helpful, because not every inspection point can be easily reached from the ground. Furthermore, having independent drones might even allow to inspect previously unreachable spots. Lastly, if the robot still keeps part of its sensory some of the inspection checks could even be carried out in parallel. The result would be a much faster overall inspection time. It is highly likely that projects based on Air-Cobot will have this kind of feature.

5.2.3. Rethink the robot drive system

The 4MOB platform used as the Air-Cobot base was primarily designed for outdoor use in challenging terrains (mud, hard to traverse terrains such as forests, etc.) and not for the use in structured, paved environments. Hence, the airport does not allow to completely benefit from the strength and the capabilities of the 4MOB (high torque, ability to climb small obstacles, resistance to difficult weather conditions). The most important drawback of this robotic system comes from its drive system. Indeed, it uses skid-steering-drive on four large and separately controllable wheels, meaning that turning is achieved by applying different velocities on them. Therefore, with a weight (without any load) of $140kg$ [Sterela and Desfosses,], the 4MOB presents a highly shaky behavior during in-place rotations on a ground, with a high friction coefficient of its

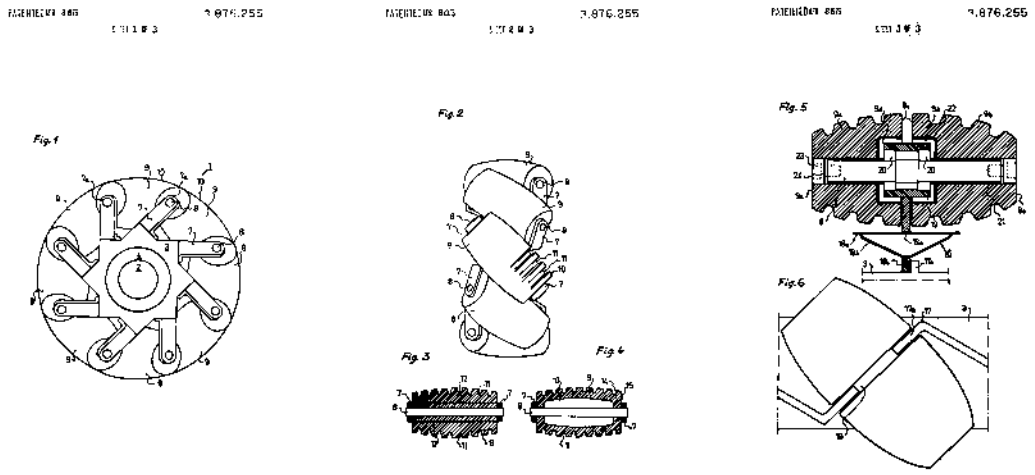


Figure 5.1.: Sketch and cross section of the Mecanum wheel [Ilon, 1975]

wheels. This leads to a challenging localization of the robot with odometry and for feature tracking (shaking sensors IMU noise and tracking errors). This issue was one of the main difficulties we had to overcome during the project.

An obvious fix to this problem is to replace the drive system by another one. A first solution is to allow for omni-directional movements by using for example a Mecanum wheel [Ilon, 1975] [Indiveri, 2009] (commonly known as Swedish wheels). It would significantly improve the robot turning behavior. However, its performance off-road would severely be restricted since such a system does require excellent grip. Figure 5.1 provides technical sketches of this wheel concept. A downside of these wheels for the project might be the maximum velocity limit. However, at the moment, this should not be an issue for the near future, because the robot is still not operating at speeds higher than $2m/s$.

5.2.4. Incorporation of a HUD on a tablet for the operator

So far, the robot mission is still launched with either a direct physical link (Ethernet cable) or a wireless connection. In the future, it is planned to replace this connection by a more user friendly application on a tablet or other mobile device. This application will show the operator (see Figure 5.2):

- how many aircrafts need inspection and the robot scheduling ;
- the current and previous inspection logs and the robot decision whether to clear or ground the aircraft ;
- alert messages if Air-Cobot cannot complete a given mission (obstacle like airport staff or vehicles in one of checkpoints, algorithmic problems, low battery charge,

etc.)

This improvement towards a more intuitive and friendly interface is mandatory in the context of the factory of the future.

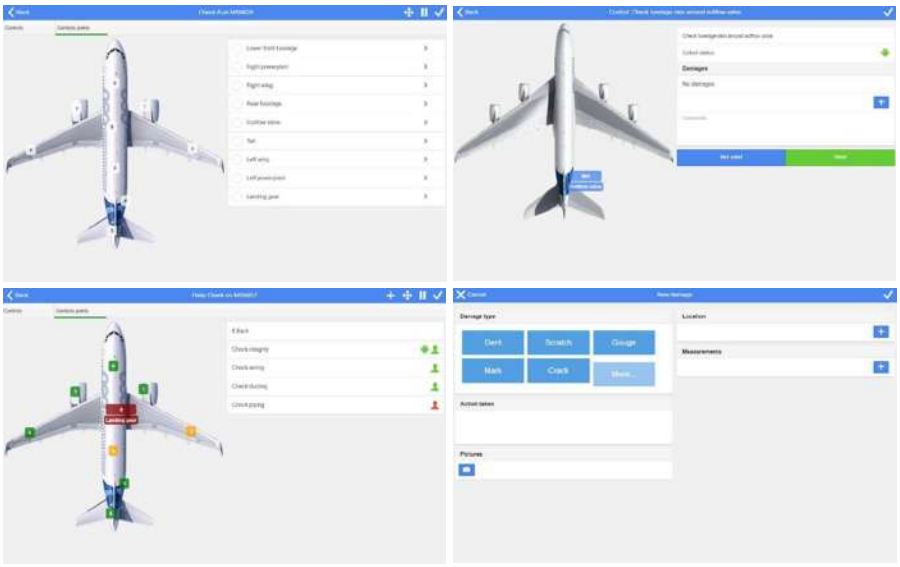


Figure 5.2.: Heads Up Display for mobile devices for Air-Cobot control designed by 2morrow

These different improvements to the existing system will allow to obtain a complete efficient prototype for automating the aircrafts inspection, thus enhancing the maintenance on airports and the flights safety.

5.2.5. Real Time capabilities of the platform

"A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed." [Li, 2009]

One of the major issues of providing a product that can be commercialized is that certain requirements towards its safety need to be provided. To be able to ensure a defined level of safety, it is necessary to implement once algorithms on a system with real time capability. This does always mean that the system is extremely fast. On the contrary, even relatively slow systems can provide real time capability. The definition of a real time system is that in a prior defined time step certain function have to be executed and completed. In the case of mobile robotics and on Air-Cobot the aspiration towards real time capability is mostly defined on the robot's maximum velocity. Thus, wanting to ensure the robot's capability to perform emergency breaking before hitting an

object. Therefore, the time between detecting, evaluating and responding to an obstacle defines the time step towards a deadline.

There exist three different classifications of real time systems:

1. Hard - Any time the system misses a deadline (even a single time) will be considered a system failure.

2. Firm - Missing deadlines infrequently is tolerated even though systems quality is degraded. Any acquired result after exceeding of a deadline is consider to be not useful

3. Soft - These class of systems are rated by their usefulness, which is in turn dependent on the acquisition of results after the deadline (systems usefulness degrades with advancing time after deadline was exceeded)

[Shin and Ramanathan, 1994]

Air-Cobot cannot be considered to be in the first category of real time systems since the underlying operation system alone can already not ensure real time process execution. However, it can be put in the group of firm real time systems, since missing a deadline is extremely rare.

6. Abstract

Navigation référencée vision dans un environnement dynamique

Résumé: Cette thèse s'intéresse au problème de la navigation autonome au long cours de robots mobiles à roues dans des environnements dynamiques. Elle s'inscrit dans le cadre du projet FUI Air-Cobot. Ce projet, porté par Akka Technologies, a vu collaborer plusieurs entreprises (Akka, Airbus, 2MORROW, Sterela) ainsi que deux laboratoires de recherche, le LAAS et Mines Albi. L'objectif est de développer un robot collaboratif (ou cobot) capable de réaliser l'inspection d'un avion avant le décollage ou en hangar. Différents aspects ont donc été abordés : le contrôle non destructif, la stratégie de navigation, le développement du système robotisé et de son instrumentation, etc. Cette thèse répond au second problème évoqué, celui de la navigation. L'environnement considéré étant aéroportuaire, il est hautement structuré et répond à des normes de déplacement très strictes (zones interdites, etc.). Il peut être encombré d'obstacles statiques (attendus ou non) et dynamiques (véhicules divers, piétons, ...) qu'il conviendra d'éviter pour garantir la sécurité des biens et des personnes. Cette thèse présente deux contributions. La première porte sur la synthèse d'un asservissement visuel permettant au robot de se déplacer sur de longues distances (autour de l'avion ou en hangar) grâce à une carte topologique et au choix de cibles dédiées. De plus, cet asservissement visuel exploite les informations fournies par toutes les caméras embarquées. La seconde contribution porte sur la sécurité et l'évitement d'obstacles. Une loi de commande basée sur les spirales équiangulaires exploite seulement les données sensorielles fournies par les lasers embarqués. Elle est donc purement référencée capteur et permet de contourner tout obstacle, qu'il soit fixe ou mobile. Il s'agit donc d'une solution générale permettant de garantir la non collision. Enfin, des résultats expérimentaux, réalisés au LAAS et sur le site d'Airbus à Blagnac, montrent l'efficacité de la stratégie développée.

Mots clés : Asservissement visuel, évitement d'obstacles, ...

Vision based navigation in a dynamic environment

Abstract: This thesis is directed towards the autonomous long range navigation of wheeled robots in dynamic environments. It takes place within the Air-Cobot project. This project aims at designing a collaborative robot (cobot) able to perform the preflight

inspection of an aircraft. The considered environment is then highly structured (airport runway and hangars) and may be cluttered with both static and dynamic unknown obstacles (luggage or refueling trucks, pedestrians, etc.). Our navigation framework relies on previous works and is based on the switching between different control laws (go to goal controller, visual servoing, obstacle avoidance) depending on the context. Our contribution is twofold. First of all, we have designed a visual servoing controller able to make the robot move over a long distance thanks to a topological map and to the choice of suitable targets. In addition, multi-camera visual servoing control laws have been built to benefit from the image data provided by the different cameras which are embedded on the Air-Cobot system. The second contribution is related to obstacle avoidance. A control law based on equiangular spirals has been designed to guarantee non collision. This control law, based on equiangular spirals, is fully sensor-based, and allows to avoid static and dynamic obstacles alike. It then provides a general solution to deal efficiently with the collision problem. Experimental results, performed both in LAAS and in Airbus hangars and runways, show the efficiency of the developed techniques.

Keywords: Visual Servoing, obstacle avoidance...

A. Robot Transformations and Jacobian Computation

Contents

A.1. Transformations concerning the stereo camera systems

The following sections will deal with the detailed translations and rotations towards both the front and the rear camera system. These are vital matrices which will find great usefulness in the later generation of velocity twist matrix and the robot Jacobian.

A.1.1. Transformation to the front cameras system

As we have explained in Section 2.2.1, the Air-Cobot system is equipped with several visual sensors. The following paragraph will provide transformations, the Jacobian and further useful information of the front camera system.

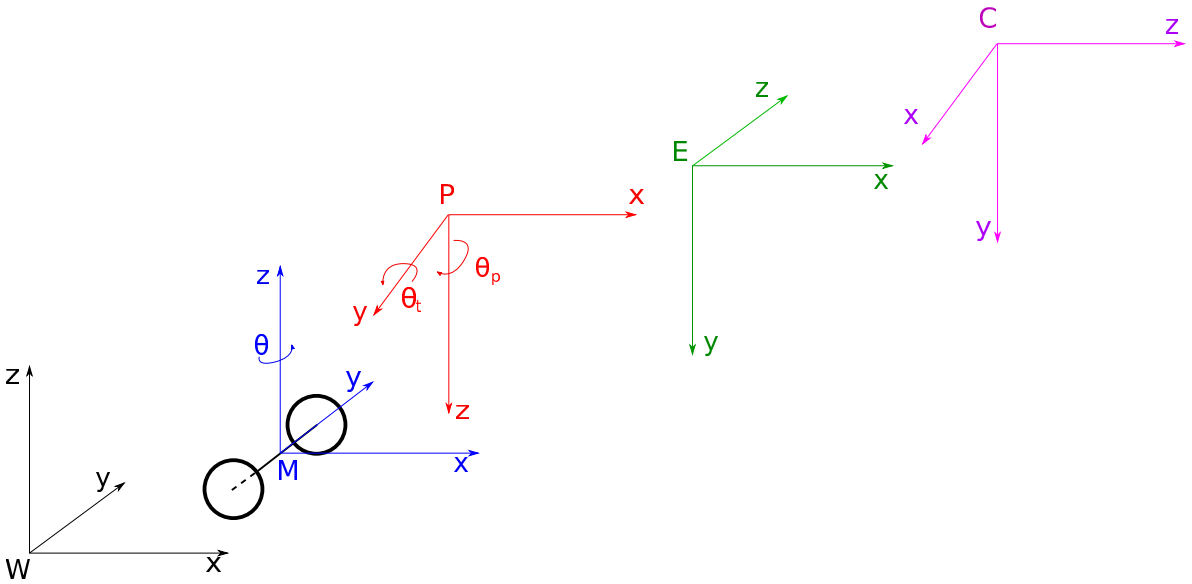


Figure A.1.: Robot coordinate system for the front cameras

As has been explained in this thesis, we make use of the ViSP library for our research. Therefore, this document some of the writing conventions differ. For example $wMm = M_{R_W/R_M}$. Let $R_i(\mu)$ be the rotation matrix of angle μ around axis i and let T_v be the translation matrix of vector v. Furthermore, θ_p will refer to the pan angle, while θ_t will designate the tilt angle of the stereo camera system from here on out. Depending of the section, the camera we are referring to is either in the front or in the back. Since we making use of a stereo camera system these matrices can refer to either the left or the right camera (determined by z_E).

$$M_{R_W/R_M} = \begin{pmatrix} \cos(\theta_{robot}) & -\sin(\theta_{robot}) & 0 & x_M \\ \sin(\theta_{robot}) & \cos(\theta_{robot}) & 0 & y_M \\ 0 & 0 & 1 & z_M \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.1}$$

$$[R_x(\pi)/T_{(x_P,y_P,z_P)}] = \begin{pmatrix} 1 & 0 & 0 & x_P \\ 0 & -1 & 0 & y_P \\ 0 & 0 & -1 & z_P \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.2}$$

$$[R_z(\theta^{cam}_{pan})] = \begin{pmatrix} \cos(\theta^{cam}_{pan}) & -\sin(\theta^{cam}_{pan}) & 0 & 0 \\ \sin(\theta^{cam}_{pan}) & \cos(\theta^{cam}_{pan}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.3}$$

$$[R_y(-\theta^{cam}_{tilt})] = \begin{pmatrix} \cos(\theta^{cam}_{tilt}) & 0 & -\sin(\theta^{cam}_{tilt}) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta^{cam}_{tilt}) & 0 & \cos(\theta^{cam}_{tilt}) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.4}$$

$$\begin{aligned} M_{R_M/R_P} &= [R_x(\pi)/T_{(x_P,y_P,z_P)}] \times [R_z(\theta^{cam}_{pan})] \times [R_y(-\theta^{cam}_{tilt})] \\ &= \begin{pmatrix} \cos(\theta^{cam}_{pan})\cos(\theta^{cam}_{tilt}) & -\sin(\theta^{cam}_{pan}) & -\cos(\theta^{cam}_{pan})\sin(\theta^{cam}_{tilt}) & x_P \\ -\sin(\theta^{cam}_{pan})\cos(\theta^{cam}_{tilt}) & -\cos(\theta^{cam}_{pan}) & \sin(\theta^{cam}_{pan})\sin(\theta^{cam}_{tilt}) & y_P \\ -\sin(\theta^{cam}_{tilt}) & 0 & -\cos(\theta^{cam}_{tilt}) & z_P \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{A.5}$$

$$M_{R_P/R_E} = [R_x\left(\frac{\pi}{2}\right)] = \begin{pmatrix} 1 & 0 & 0 & x_E \\ 0 & 0 & -1 & y_E \\ 0 & 1 & 0 & z_E \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.6}$$

$$M_{R_E/R_C} = [R_y\left(\frac{\pi}{2}\right)] = \begin{pmatrix} 0 & 0 & 1 & x_C \\ 0 & 1 & 0 & y_C \\ -1 & 0 & 0 & z_C \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.7}$$

A.1.2. Transformation to the rear cameras

$$[R_y(\pi)/T_{(x_P,y_P,z_P)}] = \begin{pmatrix} -1 & 0 & 0 & x_P \\ 0 & 1 & 0 & y_P \\ 0 & 0 & -1 & z_P \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.8}$$

$$\begin{aligned} M_{R_M/R_P} &= [R_y(\pi)/T_{(x_P,y_P,z_P)}] \times [R_z(\theta^{cam}_{pan})] \times [R_y(-\theta^{cam}_{tilt})] \\ &= \begin{pmatrix} -\cos(\theta^{cam}_{pan})\cos(\theta^{cam}_{tilt}) & \sin(\theta^{cam}_{pan}) & \cos(\theta^{cam}_{pan})\sin(\theta^{cam}_{tilt}) & x_P \\ \sin(\theta^{cam}_{pan})\cos(\theta^{cam}_{tilt}) & \cos(\theta^{cam}_{pan}) & -\sin(\theta^{cam}_{pan})\sin(\theta^{cam}_{tilt}) & y_P \\ -\sin(\theta^{cam}_{tilt}) & 0 & -\cos(\theta^{cam}_{tilt}) & z_P \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{A.9}$$

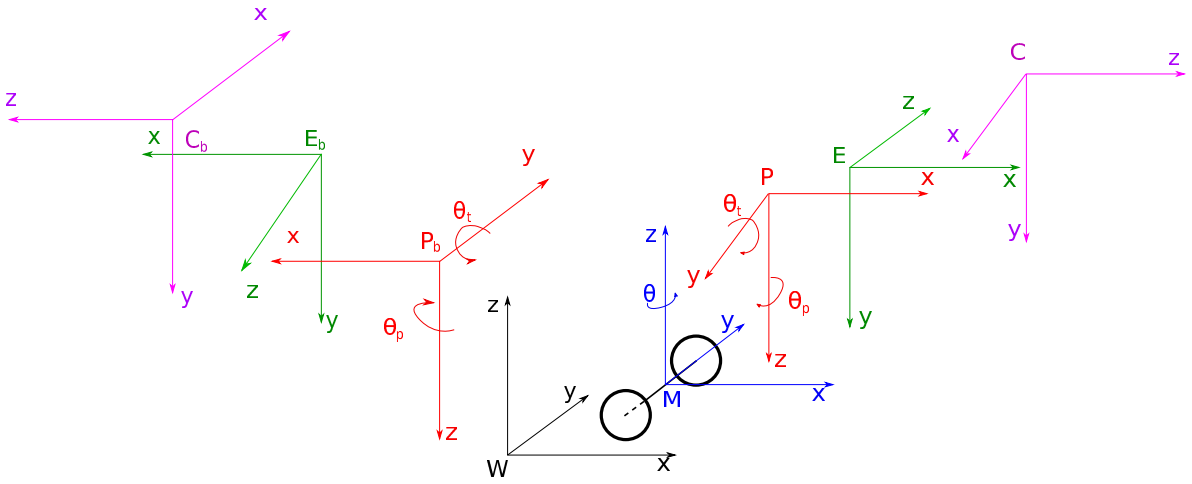


Figure A.2.: Robot coordinate system for the rear cameras

A.1.3. Additional transformations

This section presents the remaining coordinate systems. Although necessary for the robot operation, those coordinate systems are not essential for the Visual Servoing. Figure A.3 provides an extension of the robots coordinate systems, the remaining homogeneous transformations are defined in the following equations.

Transformation to the IMU

$$M_{R_M/R_{IMU}} = \begin{pmatrix} 1 & 0 & 0 & x_{IMU} \\ 0 & 1 & 0 & y_{IMU} \\ 0 & 0 & 1 & z_{IMU} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.10}$$

Transformation to the Pan-Tilt-Unit of the LASER

$$[R_z(\phi_{pan}^{laser})] = \begin{pmatrix} \cos(\phi_{pan}^{laser}) & -\sin(\phi_{pan}^{laser}) & 0 & 0 \\ \sin(\phi_{pan}^{laser}) & \cos(\phi_{pan}^{laser}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.11}$$

$$[R_y(\phi_{tilt}^{laser})] = \begin{pmatrix} \cos(\phi_{tilt}^{laser}) & 0 & \sin(\phi_{tilt}^{laser}) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi_{tilt}^{laser}) & 0 & \cos(\phi_{tilt}^{laser}) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.12}$$

$$[R_z(\phi_{pan}^{laser})]\times[R_y(\phi_{tilt}^{laser})] = \begin{pmatrix} \cos(\phi_{pan}^{laser})\cos(\phi_{tilt}^{laser}) & -\sin(\phi_{pan}^{laser}) & \cos(\phi_{pan}^{laser})\sin(\phi_{tilt}^{laser}) & 0 \\ \sin(\phi_{pan}^{laser})\cos(\phi_{tilt}^{laser}) & \cos(\phi_{pan}^{laser}) & \sin(\phi_{pan}^{laser})\sin(\phi_{tilt}^{laser}) & 0 \\ -\sin(\phi_{tilt}^{laser}) & 0 & \cos(\phi_{tilt}^{laser}) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.13}$$

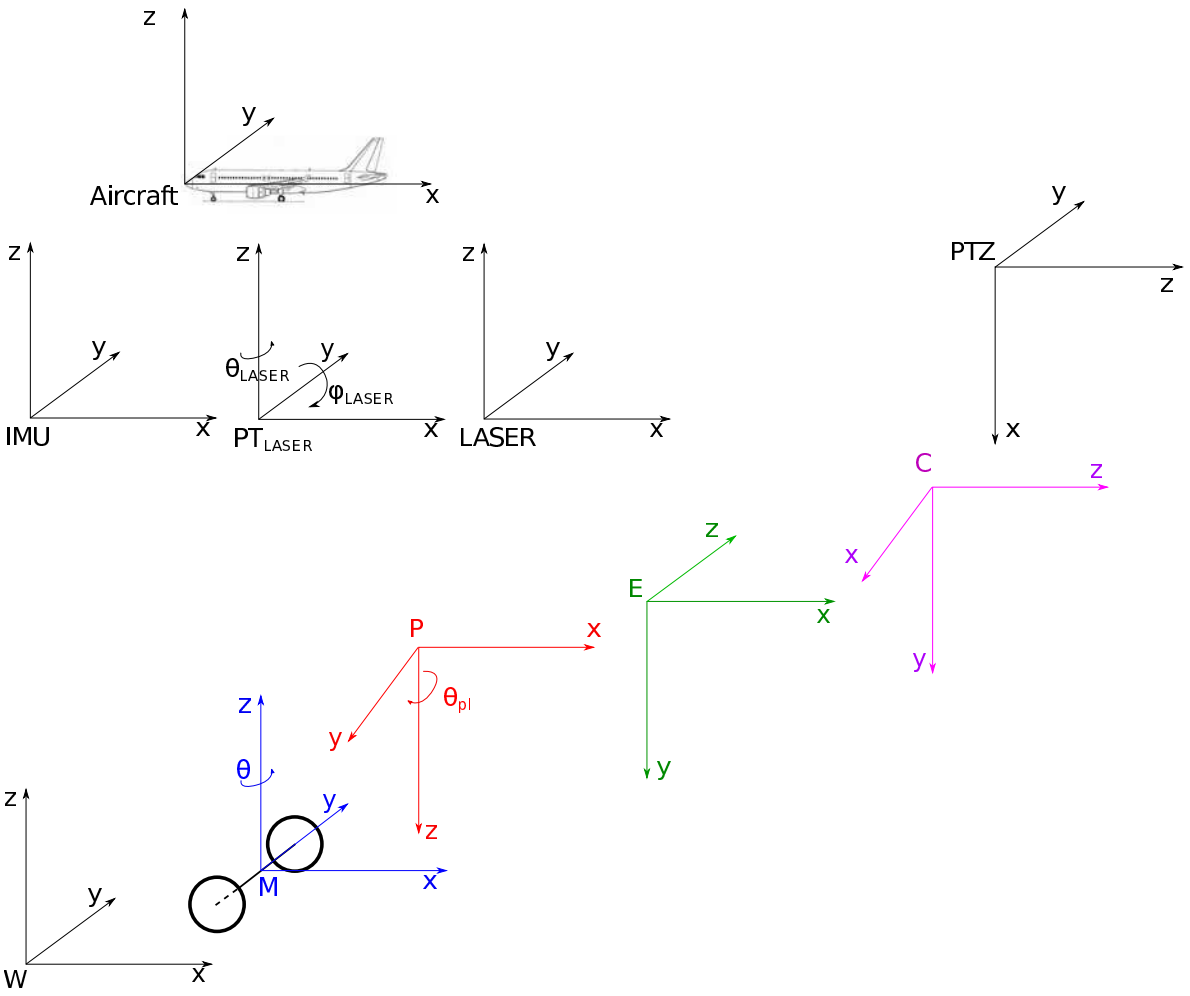


Figure A.3.: Robot Coordinate System

$$M_{R_M/R_{PT_{LASER}}} = \begin{pmatrix} \cos(\phi_{pan}^{laser})\cos(\phi_{tilt}^{laser}) & -\sin(\phi_{pan}^{laser}) & \cos(\phi_{pan}^{laser})\sin(\phi_{tilt}^{laser}) & x_{PT_{LASER}} \\ \sin(\phi_{pan}^{laser})\cos(\phi_{tilt}^{laser}) & \cos(\phi_{pan}^{laser}) & \sin(\phi_{pan}^{laser})\sin(\phi_{tilt}^{laser}) & y_{PT_{LASER}} \\ -\sin(\phi_{tilt}^{laser}) & 0 & \cos(\phi_{tilt}^{laser}) & z_{PT_{LASER}} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.14}$$

Transformation from the LASER PTU to the LASER

$$M_{R_{PT_{LASER}}/R_{LASER}} = \begin{pmatrix} 1 & 0 & 0 & x_{LASER} \\ 0 & 1 & 0 & y_{LASER} \\ 0 & 0 & 1 & z_{LASER} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.15}$$

Transformation to the PTZ camera

This transformation does not contain the pan and tilt agnles of the PTZ camera.

$$M_{R_M/R_{PTZ}} = \begin{pmatrix} 0 & 0 & 1 & x_{PTZ} \\ 0 & 1 & 0 & y_{PTZ} \\ -1 & 0 & 0 & z_{PTZ} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.16}$$

Transformation to the Aircraft frame

The homogeneous transformation between robot and aircraft will finally be provided by the localization (RT-SLAM).

$$M_{R_M/R_{Aircraft}} = \begin{pmatrix} 1 & 0 & 0 & x_{Aircraft} \\ 0 & 1 & 0 & y_{Aircraft} \\ 0 & 0 & 1 & z_{Aircraft} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.17}$$

A.2. Kinematic Screw

In the upcoming section we will derive the robot kinematic screw and the velocity twist matrix. This section will follow the reasoning of [Cadenat, 1999], hence all bold, bracketed written section numbers (e.g.: **[11.1]**) will refer to equations in this document. Keep in mind that $T^{R_j}_{A/R_i}$ is the kinematic screw at the point A relative to the frame R_i, but expressed in the frame R_j. This should explain the general notation convention that was taken for the upcoming derivation of the kinematic screw:

[11.1]:

$$T^{R_E}_{E/R_W} = \begin{pmatrix} V^{R_E}_{E/R_M} + V^{R_E}_{M/R_W} + \Omega^{R_E}_{R_M/R_W} \wedge ME_{R_E} \\ \Omega^{R_E}_{R_E/R_M} + \Omega^{R_E}_{R_M/R_W} \end{pmatrix} \tag{A.18}$$

In order to simplify part of the calculation, the robot frame R_M is chosen for now:

$$T^{R_M}_{E/R_W} = \begin{pmatrix} V^{R_M}_{E/R_M} + V^{R_M}_{M/R_W} + \Omega^{R_M}_{R_M/R_W} \wedge ME_{R_M} \\ \Omega^{R_M}_{R_E/R_M} + \Omega^{R_M}_{R_M/R_W} \end{pmatrix} \tag{A.19}$$

[11.2]:

$$\begin{cases} \dot{x} = v\cos(\theta_{robot}) \\ \dot{y} = v\sin(\theta_{robot}) \\ \dot{\theta} = \omega \end{cases} \tag{A.20}$$

[11.3]:

$$\begin{cases} V^{R_W}_{M/R_W} = (v\cos(\theta_{robot}),\, v\sin(\theta_{robot}),\, 0)^T \\ \Omega^{R_W}_{R_M/R_W} = (0,\, 0,\, \omega)^T \end{cases} \tag{A.21}$$

[**11.4**]:

$$\begin{cases} V_{M/W}^{R_M} = (v,\, 0,\, 0) \\ \Omega_{R_M/W}^{R_M} = \left(\Omega_{R_M/R_W}^{R_W}\right) \end{cases} \tag{A.22}$$

[**11.5**]:

$$T_{E/R_M}^{R_M} = \begin{pmatrix} V_{E/R_M}^{R_M} \\ \Omega_{R_E/R_M}^{R_M} \end{pmatrix} = \begin{pmatrix} V_{E/R_P}^{R_M} + V_{P/R_M}^{R_M} + \Omega_{R_P/R_M}^{R_M} \wedge PE^{R_M} \\ \Omega_{R_E/R_P}^{R_M} + \Omega_{R_P/R_M}^{R_M} \end{pmatrix} \tag{A.23}$$

because $V_{E/R_P} = dPE/dt$ and $V_{P/R_M} = dMP/dt = 0$ because there the transformation between P and E is rigid. Furthermore, since R_E is fixed to R_P, Ω_{R_E/R_P} will also be zero.

[**11.6**]:

$$T_{E/R_M} = \begin{pmatrix} \Omega_{R_P/R_M} \wedge PE \\ \Omega_{R_P/R_M} \end{pmatrix} \tag{A.24}$$

PE is given by (x_E, y_E, z_E)

To simplify the notations, let c_p, s_p, c_t and s_t be respectively $\cos(\theta_{pan}^{cam})$, $\sin(\theta_{pan}^{cam})$, $\cos(\theta_{tilt}^{cam})$ and $\sin(\theta_{tilt}^{cam})$.

[**11.7**]:

$$R_{R_M/R_P} = \begin{pmatrix} c_p\, c_t & -s_p & -c_p\, s_t \\ -s_p\, c_t & -c_p & s_p\, s_t \\ -s_t & 0 & -c_t \end{pmatrix} \tag{A.25}$$

[**11.8**]:

$$t_{P/E}^{R_M} = R_{R_M/R_P} \times t_{P/E}^{R_P} \tag{A.26}$$

to [**11.8**]

$$t_{P/E}^{R_M} = \begin{pmatrix} x_E\, c_p\, c_t & - & y_E\, s_p & - & z_E\, c_p\, s_t \\ -x_E\, s_p\, c_t & - & y_E\, c_p & + & z_E\, s_p\, s_t \\ -x_E\, s_t & & & - & z_E\, c_t \end{pmatrix} \tag{A.27}$$

$$\begin{aligned} & \Omega_{R_P/R_M} \wedge t_{P/E}^{R_M} \\ &= \begin{pmatrix} 0 \\ \dot{\theta}_t \\ -\dot{\theta}_p \end{pmatrix} \wedge \begin{pmatrix} x_E\, c_p\, c_t & - & y_E\, s_p & - & z_E\, c_p\, s_t \\ -x_E\, s_p\, c_t & - & y_E\, c_p & + & z_E\, s_p\, s_t \\ -x_E\, s_t & & & - & z_E\, c_t \end{pmatrix} \\ &= \begin{pmatrix} -x_E\, s_p\, c_t - y_E\, c_p + z_E\, s_p\, s_t \\ -x_E\, c_p\, c_t + y_E\, s_p + z_E\, c_p\, s_t \\ 0 \end{pmatrix} \times \dot{\theta}_p + \begin{pmatrix} -x_E\, s_t - z_E\, c_t \\ 0 \\ -x_E\, c_p\, c_t + y_E\, s_p + z_E\, c_p\, s_t \end{pmatrix} \times \dot{\theta}_t \end{aligned} \tag{A.28}$$

negative θ_p because in the frame R_M.

[**11.9**]:

$$T^{R_M}_{E/R_M} = \begin{pmatrix} V^{R_M}_{E/R_M} \\ \Omega^{R_M}_{R_E/R_M} \end{pmatrix} = \begin{pmatrix} -x_E\, s_p\, c_t - y_E\, c_p + z_E\, s_p\, s_t \\ -x_E\, c_p\, c_t + y_E\, s_p + z_E\, c_p\, s_t \\ 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} \times \dot{\theta}_p \tag{A.29}$$

[**11.9**]:

$$T^{R_M}_{E/R_M} = \begin{pmatrix} V^{R_M}_{E/R_M} \\ \Omega^{R_M}_{R_E/R_M} \end{pmatrix} = \begin{pmatrix} -x_E\, s_p\, c_t - y_E\, c_p + z_E\, s_p\, s_t \\ -x_E\, c_p\, c_t + y_E\, s_p + z_E\, c_p\, s_t \\ 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} \times \dot{\theta}_p + \begin{pmatrix} -x_E\, s_t - z_E\, c_t \\ 0 \\ -x_E\, c_p\, c_t + y_E\, s_p + z_E\, c_p\, s_t \\ 0 \\ 1 \\ 0 \end{pmatrix} \times \dot{\theta}_t \tag{A.30}$$

$t_{M/E} = t_{M/P} + t_{P/E}$
[**11.10**]:

$$t^{RM}_{M/E} = \begin{pmatrix} x_p & + & x_E\, c_p\, c_t & - & y_E\, s_p & - & z_E\, c_p\, s_t \\ y_p & - & x_E\, s_p\, c_t & - & y_E\, c_p & + & z_E\, s_p\, s_t \\ z_p & - & x_E\, s_t & & & - & z_E\, c_t \end{pmatrix} \tag{A.31}$$

for [**11.11**]:

$$\dot{q} = \begin{pmatrix} v \\ \dot{\theta} \\ \dot{\theta}_p \\ \dot{\theta}_t \end{pmatrix} \tag{A.32}$$

for [**11.11**]:

$$\Omega^{R_M}_{R_E/W} = \begin{pmatrix} 0 \\ 0 \\ \dot{\theta} \end{pmatrix} + \begin{pmatrix} 0 \\ \dot{\theta}_t \\ -\dot{\theta}_p \end{pmatrix} \tag{A.33}$$

for [**11.11**]: concerning $\dot{\theta}$

$$velocity = rotation \wedge position(center\, of\, rotation) \tag{A.34}$$

$$\dot{\theta}_{partofthescrew} = \begin{pmatrix} 0 \\ 0 \\ \dot{\theta} \end{pmatrix} \wedge t^{RM}_{M/E} = \begin{pmatrix} -y_p + x_E\, s_p\, c_t + y_E\, c_p - z_E\, s_p\, s_t \\ x_p + x_E\, c_p\, c_t - y_E\, s_p - z_E\, c_p\, s_t \\ 0 \end{pmatrix} \times \dot{\theta} \tag{A.35}$$

[**11.11**]:

$$T^{R_M}_{E/W} = \begin{pmatrix} [J_1] & [J_2] & [J_3] & [J_4] \end{pmatrix} \times \dot{q} \tag{A.36}$$

$$[J_1][J_2] = \begin{pmatrix} 1 & -y_P + x_E\, s_p\, c_t + y_E\, c_p - z_E\, s_p\, s_t \\ 0 & x_P + x_E\, c_p\, c_t - y_E\, s_p - z_E\, c_p\, s_t \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{A.37}$$

$$[J_3][J_4] = \begin{pmatrix} -x_E\, s_p\, c_t - y_E\, c_p + z_E\, s_p\, s_t & -x_E\, s_t - z_E\, c_t \\ -x_E\, c_p\, c_t + y_E\, s_p + z_E\, c_p\, s_t & 0 \\ 0 & -x_E\, c_p\, c_t + y_E\, s_p + z_E\, c_p\, s_t \\ 0 & 0 \\ 0 & 1 \\ -1 & 0 \end{pmatrix} \tag{A.38}$$

[**11.12**]:

$$R_{R_M/R_E} = R_{R_M/R_P} \times R_{R_P/R_E} \tag{A.39}$$

[**11.13**]:

$$R_{R_P/R_E} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \tag{A.40}$$

to [**11.14**]:

$$R_{R_M/R_E} = R_{R_M/R_P} \times R_{R_P/R_E} = \begin{pmatrix} c_p\, c_t & -s_p & -c_p\, s_t \\ -s_p\, c_t & -c_p & s_p\, s_t \\ -s_t & 0 & -c_t \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} c_p\, c_t & -c_p\, s_t & s_p \\ -s_p\, c_t & s_p\, s_t & c_p \\ -s_t & -c_t & 0 \end{pmatrix} \tag{A.41}$$

to [**11.14**]:

$$R^T_{R_M/R_E} = \begin{pmatrix} c_p\, c_t & -s_p\, c_t & -s_t \\ -c_p\, s_t & s_p\, s_t & -c_t \\ s_p & c_p & 0 \end{pmatrix} \tag{A.42}$$

[**11.14**]:

$$T^{R_E}_{E/W} = \begin{pmatrix} R^T_{R_M/R_E} & 0 \\ 0 & R^T_{R_M/R_E} \end{pmatrix} \times T^{R_M}_{E/W} \tag{A.43}$$

[**11.15**]:

$$T^{R_E}_{E/W} = \begin{pmatrix} c_p\, c_t & -y_P\, c_p\, c_t - x_P\, s_p\, c_t + y_E\, c_t & -y_E\, c_t & -y_E\, s_p\, s_t - z_E\, c_p \\ -c_p\, s_t & y_P\, c_p\, s_t + x_P\, s_p\, s_t - y_E\, s_t & -y_E\, s_t & x_E\, c_p - y_E\, s_p\, c_t \\ s_p & -y_P\, s_p + x_P\, c_p + x_E\, c_t - z_E s_t & -x_E\, c_t + z_E\, s_t & -x_E\, s_t\, s_p - z_E\, c_t\, s_p \\ 0 & -s_t & s_t & -s_p\, c_t \\ 0 & -c_t & c_t & s_p\, s_t \\ 0 & 0 & 0 & c_p \end{pmatrix} \times \dot{q} \tag{A.44}$$

For the back cameras, the equations are slightly different.

$$T_{E/W}^{R_E} = \begin{pmatrix} -c_p\,c_t & y_P\,c_p\,c_t + x_P\,s_p\,c_t + y_E\,c_t & -y_E\,c_t & y_E\,s_p\,s_t + z_E\,c_p \\ c_p\,s_t & -y_P\,c_p\,s_t - x_P\,s_p\,s_t - y_E\,s_t & -y_E\,s_t & -x_E\,c_p + y_E\,s_p\,c_t \\ -s_p & y_P\,s_p - x_P\,c_p + x_E\,c_t - z_E s_t & -x_E\,c_t + z_E\,s_t & x_E\,s_t\,s_p + z_E\,c_t\,s_p \\ 0 & -s_t & s_t & s_p\,c_t \\ 0 & -c_t & c_t & -s_p\,s_t \\ 0 & 0 & 0 & -c_p \end{pmatrix} \times \dot{q} \tag{A.45}$$

A.3. Velocity screw

The following section will present the derivation of the velocity twist matrix. Velocity twist matrix is a regular term in the ViSP framework and describes a matrix that allows the transformation of the velocity screw between the camera frame,C, and the end-effector frame, E,. During our derivation we have found that there are 2 possible ways of deriving that matrix. Both solutions will be presented in the upcoming section.

A.3.1. Solution "1" of deriving the velocity screw matrix

For this approach we will change the application point before changing the base.

A.3.2. a

$$\vec{V}_{C/W} = \vec{V}_{E/W} + \hat{t}_{C/E} \wedge \Omega_{R_E/R_W} \tag{A.46}$$

$$\vec{V}_{C/W}^{R_E} = \vec{V}_{E/W}^{R_E} + t_{C/E}^{R_E} \wedge \Omega_{R_E/R_W}^{R_E} = \begin{pmatrix} I & \hat{t}_{C/E}^{R_E} \end{pmatrix} \times T_{E/W}^{R_E} \tag{A.47}$$

$$T_{C/W}^{R_E} = \begin{pmatrix} I & \hat{t}_{C/E}^{R_E} \\ 0 & I \end{pmatrix} \times T_{E/W}^{R_E} \tag{A.48}$$

b

$$\begin{aligned} T_{C/W}^{R_C} &= \begin{pmatrix} R_{R_C/R_E} & 0 \\ 0 & R_{R_C/R_E} \end{pmatrix} \times \begin{pmatrix} I & \hat{t}_{C/E}^{R_E} \\ 0 & I \end{pmatrix} \times T_{E/W}^{R_E} \\ &= \begin{pmatrix} R_{R_C/R_E} & R_{R_C/R_E} \times \hat{t}_{C/E}^{R_E} \\ 0 & R_{R_C/R_E} \end{pmatrix} \times T_{E/W}^{R_E} \end{aligned} \tag{A.49}$$

A.3.3. Solution "2" of deriving the velocity screw matrix

Contrary to the first solution, we will now change the base before we deal with the point of application.

A.3.4. a

$$T^{R_C}_{E/R_W} = \begin{pmatrix} R_{R_C/R_E} & 0 \\ 0 & R_{R_C/R_E} \end{pmatrix} T^{R_E}_{E/W} \tag{A.50}$$

b

$$\vec{V}^{R_C}_{C/W} = \vec{V}^{R_C}_{E/W} + t^{R_C}_{C/E} \wedge \Omega^{R_C}_{R_E/R_W} \tag{A.51}$$

$$\begin{aligned} T^{R_C}_{C/W} &= \begin{pmatrix} I & \hat{t}^{R_C}_{C/E} \\ 0 & I \end{pmatrix} T^{R_C}_{E/R_W} \\ &= \begin{pmatrix} I & \hat{t}^{R_C}_{C/E} \\ 0 & I \end{pmatrix} \begin{pmatrix} R_{R_C/R_E} & 0 \\ 0 & R_{R_C/R_E} \end{pmatrix} T^{R_E}_{E/R_W} \\ &= \begin{pmatrix} R_{R_C/R_E} & \hat{t}^{R_C}_{C/E} \times R_{R_C/R_E} \\ 0 & R_{R_C/R_E} \end{pmatrix} T^{R_E}_{E/R_W} \end{aligned} \tag{A.52}$$

$$\vec{V}^{R_C}_{C/W} = R_{R_C/R_E} \times \vec{V}^{R_E}_{E/R_W} + \hat{t}^{R_C}_{C/E} \times R_{R_C/R_E} \times \Omega^{R_E}_{R_E/R_W} \tag{A.53}$$

and

$$R_{R_C/R_E} \times \Omega^{R_E}_{R_E/R_W} = \Omega^{R_C}_{R_E/R_W} \tag{A.54}$$

A.3.5. Applying solution "2" to our problem

In order to change the kinematic screw in such a way that it can be utilized for the Air-Cobot visual navigation modules one of our solution needs to be applied to the kinematic screw.

$${}^{c}v_e = \begin{pmatrix} R_{R_C/R_E} & \hat{t}^{R_C}_{C/E} \times R_{R_C/R_E} \\ 0_3 & R_{R_C/R_E} \end{pmatrix} \tag{A.55}$$

$$M_{R_E/R_C} = \begin{pmatrix} 0 & 0 & 1 & x_C \\ 0 & 1 & 0 & y_C \\ -1 & 0 & 0 & z_C \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.56}$$

but we need either only the rotation or translation:

$$R_{R_E/R_C} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} \tag{A.57}$$

$$t^{R_E}_{E/C} = \begin{pmatrix} x_C \\ y_C \\ z_C \end{pmatrix} \tag{A.58}$$

In order to be able to use this the inverse needs to be found:

$$R_{R_C/R_E} = \left(R_{R_E/R_C}\right)^{-1} = \left(R_{R_E/R_C}\right)^T = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{A.59}$$

the translation matrix is expressed in the frame of the end effector

$$t_{C/E}^{R_E} = \begin{pmatrix} -x_C \\ -y_C \\ -z_C \end{pmatrix} \tag{A.60}$$

$$t_{R_C/R_E}^{R_C} = R_{R_C/R_E} \times t_{C/E}^{R_E} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} -x_C \\ -y_C \\ -z_C \end{pmatrix} = \begin{pmatrix} z_C \\ -y_C \\ -x_C \end{pmatrix} \tag{A.61}$$

$$\hat{t}_{C/E}^{R_C} = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & x_C & -y_C \\ -x_C & 0 & -z_C \\ y_C & z_C & 0 \end{pmatrix} \tag{A.62}$$

$$\hat{t}_{C/E}^{R_C} \times R_{R_C/R_E} = \begin{pmatrix} 0 & x_C & -y_C \\ -x_C & 0 & -z_C \\ y_C & z_C & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -y_C & x_C & 0 \\ -z_C & 0 & x_C \\ 0 & z_C & -y_C \end{pmatrix} \tag{A.63}$$

so now we can put together the velocity twist matrix

$${}^c v_e = \begin{pmatrix} 0 & 0 & -1 & -y_C & x_C & 0 \\ 0 & 1 & 0 & -z_C & 0 & x_C \\ 1 & 0 & 0 & 0 & z_C & -y_C \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{A.64}$$

Which will allow us to build $T_{C/R_W}^{R_C}$ which is the Jacobian as we need it:

$$T_{C/R_W}^{R_C} = {}^c\, v_e \times T_{E/R_W}^{R_E} \tag{A.65}$$

$$\begin{aligned} T_{C/R_W}^{R_C} &= \begin{pmatrix} -s_p & y_P\, s_p - x_P\, c_p - (x_E + x_C)\, c_t + (z_E + y_C)\, s_t & x_E + x_C & 0 \\ -c_p\, s_t & y_P\, c_p\, s_t + x_P\, s_p\, s_t + (z_C - y_E)\, s_t & 0 & x_E + x_C \\ c_p\, c_t & -y_P\, c_p\, c_t - x_P\, s_p\, c_t - (z_C - y_E)\, c_t & -y_E + z_C & -z_E - y_C \\ 0 & 0 & 0 & -1 \\ 0 & -c_t & 1 & 0 \\ 0 & -s_t & 0 & 0 \end{pmatrix} \times \dot{q} \\ &= J \times \dot{q} \end{aligned} \tag{A.66}$$

A.4. Interaction matrix

The interaction matrix used is a 8×6 matrix. Below an example for one feature point (i) (in total there are four) is provided:

$$L_i = \begin{pmatrix} \frac{-1}{z_i} & 0 & \frac{X_i}{z_i} & X_i Y_i & -(1+X_i^2) & Y_i \\ 0 & \frac{-1}{z_i} & \frac{Y_i}{z_i} & 1+Y_i^2 & -X_i Y_i & -X_i \end{pmatrix} \tag{A.67}$$

$$L_{i_{red}} = \begin{pmatrix} \frac{-1}{z_i} & \frac{X_i}{z_i} & -(1+X_i^2) \\ 0 & \frac{Y_i}{z_i} & -X_i Y_i \end{pmatrix} \tag{A.68}$$

$$L = \begin{pmatrix} L_{1_{red}} \\ L_{2_{red}} \\ L_{3_{red}} \\ L_{4_{red}} \end{pmatrix} \tag{A.69}$$

the big letters resemble to points in the image plane and small letters describe 3d points

A.5. Jacobian and Velocity Twist Matrix of Air-Cobot

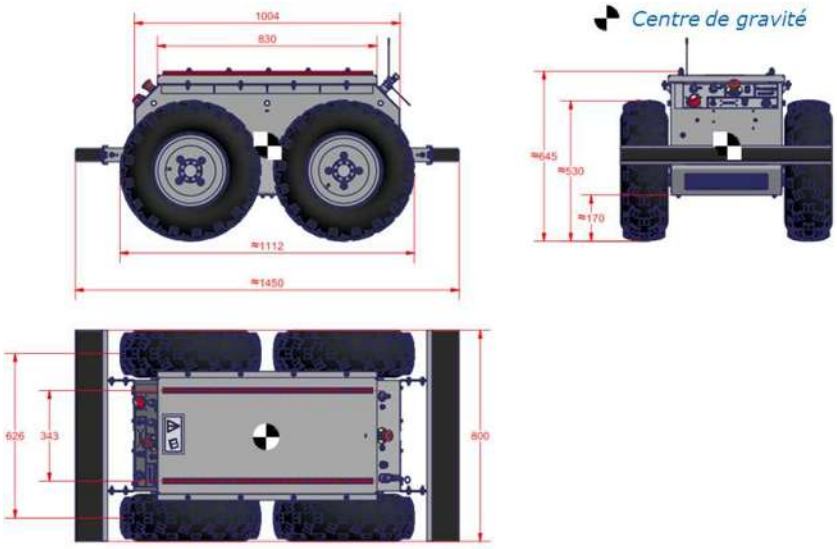


Figure A.4.: Aircobot Dimensions

Basically we will have to provide values for all points, starting form $M(x_M, y_M, z_M)$ until $C(x_C, y_C, z_C)$.

$$\begin{cases} x_M \rightarrow \text{provided by robot localization method} \\ y_M \rightarrow \text{provided by robot localization method} \\ z_M = 640 \end{cases} \tag{A.70}$$

$$\begin{cases} x_P = 505 \\ y_P = 0 \\ z_P = 50 \end{cases} \tag{A.71}$$

$$\begin{cases} x_E = 0 \\ y_E = \pm 150.00 (\text{depending on right}(+)/\text{left}(-) \text{ camera}) \\ z_E = 0 \end{cases} \tag{A.72}$$

$$\begin{cases} x_C = 0 \\ y_C = -25 \\ z_C = 0 \end{cases} \tag{A.73}$$

A.5.1. extended

$$(x_{IMU},\, y_{IMU},\, z_{IMU}) = (0,\, 0,\, 870) \tag{A.74}$$

$$(x_{PTZ},\, y_{PTZ},\, z_{PTZ}) = (594,\, 0,\, 340) \tag{A.75}$$

$$(x_{PT_{LASER}},\, y_{PT_{LASER}},\, z_{PT_{LASER}}) = (623,\, 0,\, -225) \tag{A.76}$$

$$(x_{LASER},\, y_{LASER},\, z_{LASER}) = (0,\, 0,\, 110) \tag{A.77}$$

A.6. Air-Cobot Jacobian and velocity twist matrix

A.6.1. Jacobian - eJe

$$T_{E/W}^{R_E} = \begin{pmatrix} \cos(\theta_{pan}^{cam}) & y_E - 505\sin(\theta_{pan}^{cam}) & -y_E \\ 0 & 0 & 0 \\ \sin(\theta_{pan}^{cam}) & 505\cos(\theta_{pan}^{cam}) & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \times \dot{q} \tag{A.78}$$

A.6.2. Jacobian- eJe reduced

$$T_{E/W}^{R_E} = \begin{pmatrix} \cos(\theta_{pan}^{cam}) & y_E - 505\sin(\theta_{pan}^{cam}) & -y_E \\ \sin(\theta_{pan}^{cam}) & 505\cos(\theta_{pan}^{cam}) & 0 \\ 0 & -1 & 1 \end{pmatrix} \times \dot{q} \tag{A.79}$$

When looking at the camera coordinate system it becomes obvious that linear velocities are only possible along the x and z direction. Furthermore, the only possible rotational velocity can be found around the y axis. Thus, reducing the Jacobian as done above is valid.

A.6.3. Velocity twist matrix- cVe

$${}^{c}v_e = \begin{pmatrix} 0 & 0 & -1 & 25 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 25 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{A.80}$$

A.7. Analysis of the Robots Skid Steering Drive on a flat Surface with high Traction

In this short section the control type of the robot will be discussed. On Air-Cobot and in the Gazebo simulations a skid steering drive controller is used to move the robot. As it has already been pointed out, this can have some advantages (simple command input, easy to test and no moving axis), but especially for the robot localization this can be extremely disadvantageous. Previously, the localization with wheel odometry has already been discussed. Thus, in this section the focus will be more on the other sensors (cameras for SLAM, visual odometry and Visual Servoing and LASERS). What all these methods have in common is that they rely on tracking previously detected/ identified key features. Tracking can be a very delicate and sensible task. The smaller the features get the more likely the tracker will loose them during an unexpected movement. To achieve a better baseline for discussion the following paragraph will contain an experiment in which the robot is:

1. Moved back and fourth with the standard PID gains provided by the manufacturer (fast response (P))
2. Moved back and fourth with the integral heavy PID gains (smoother behavior)
3. Turning in place with the standard PID gains provided by the manufacturer (fast response (P))
4. Turning in place with the integral heavy PID gains (smoother behavior)

During the experiments the robot speed will be recorded directly from the CAN as well as the accelerations and velocities in the approximate center of gravity of the robot provided by the IMU.

A.8. ROS-based software architecture in at the end of the project

The following figure will provide a detailed overview of the executed software architecture during a navigation mission of Air-Cobot.

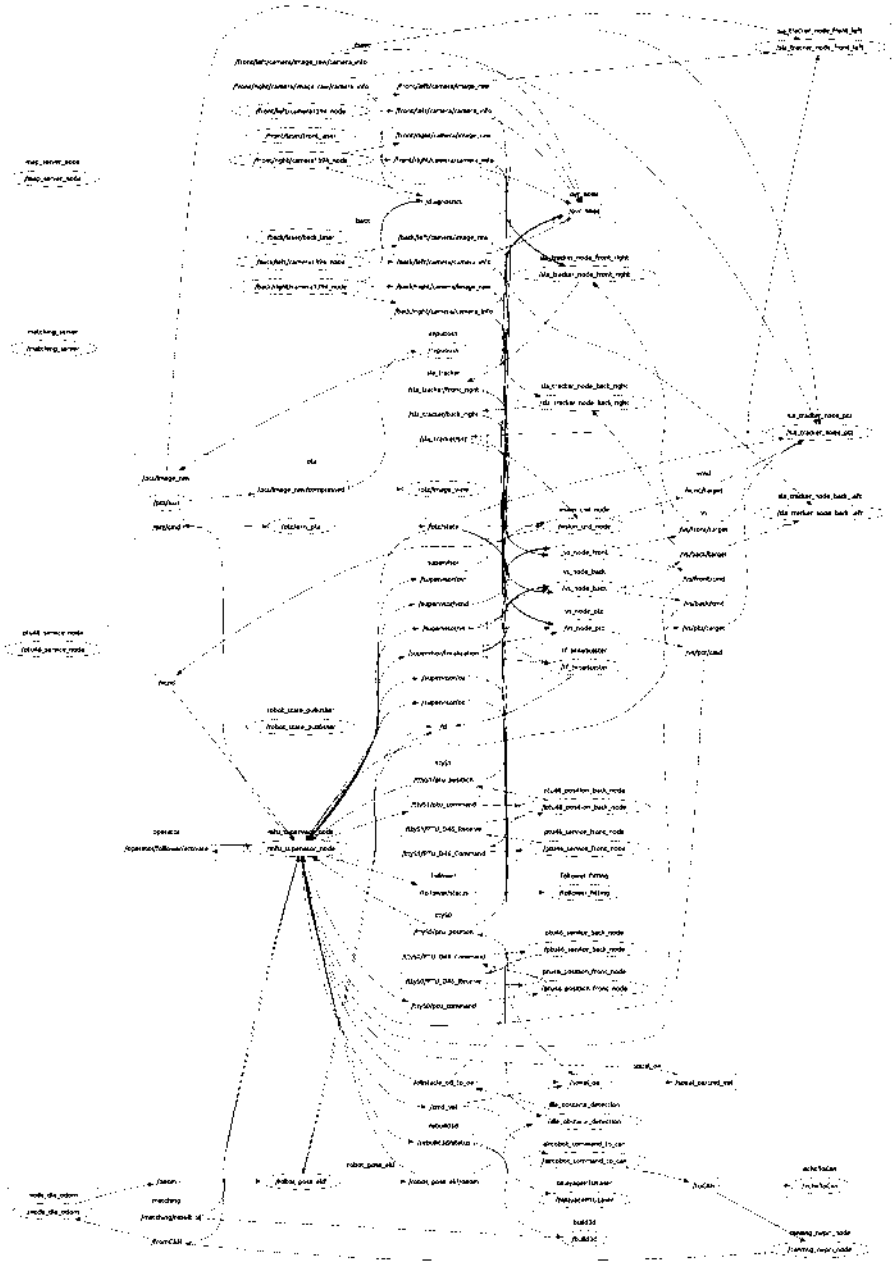
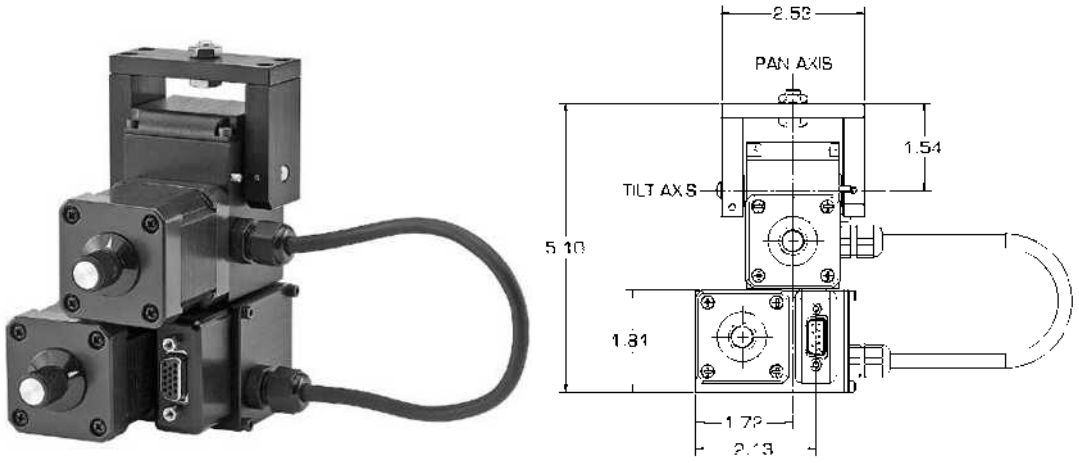


Figure A.5.: Graph showing all nodes that run during a navigation mission



(a) PTU (b) Cross section of the PTU

Figure A.6.: Extract of PTU datasheet [man, 2014]

Bibliography

[web,] Irisa webiste - lagadic - visp - tutorials. http://www.irisa.fr/lagadic/visp/documentation/visp-2.10.0/tutorial-boost-vs.html#adaptive_gain. Accessed: 2015-10-31.

[mus,] U.s., china, russia, elon musk: Entrepreneur's "insane" vision becomes reality. http://www.cbsnews.com/news/us-china-russia-elon-musk-entrepreneurs-insane-vision-becomes-reality/. Accessed: 2017-03-4.

[man, 2014] (2014). Ptu-d46.

[Allibert et al., 2010] Allibert, G., Courtial, E., and Chaumette, F. (2010). Predictive control for constrained image-based visual servoing. *IEEE Transactions on Robotics*, 26(5):933–939.

[Alper et al., 2004] Alper, B. S., Hand, J. A., Elliott, S. G., Kinkade, S., Hauan, M. J., Onion, D. K., and Sklar, B. M. (2004). How much effort is needed to keep up with the literature relevant for primary care? *Journal of the Medical Library association*, 92(4):429.

[Asimov, 1950] Asimov, I. (1950). *I, Robot*.

[Balch et al., 2007] Balch, T., Christensen, H., Camp, V., Logston, D., Collins, T., Powers, M., Egerstedt, M., Schafrik, R., Hoffman, M., Witter, J., et al. (2007). Darpa urban challenge.

[Bellot and Danes, 2001] Bellot, D. and Danes, P. (2001). Towards an lmi approach to multiobjective visual servoing. In *European Control Conference 2001*, Porto (Portugal).

[Bernstein and Raman, 2015] Bernstein, A. and Raman, A. (2015). The great decoupling: An interview with erik brynjolfsson and andrew mcafee. *Harvard Business Review*, 93:66–74.

[Besl and McKay, 1992] Besl, P. and McKay, N. D. (1992). A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256.

[Bonin-Font et al., 2008] Bonin-Font, F., Ortiz, F., and Oliver, G. (2008). Visual navigation for mobile robots : a survey. *Journal of intelligent and robotic systems*, 53(3):263.

[Borenstein and Koren, 1991] Borenstein, J. and Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, 7(3):278–288.

[Boyadzhiev, 1999] Boyadzhiev, K. N. (1999). Spirals and conchospirals in the flight of insects. *The College Mathematics Journal*, 30(1):pp. 23–31.

[Breitenstein et al., 2009] Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., and Van Gool, L. (2009). Markovian tracking-by-detection from a single, uncalibrated camera.

[Brynjolfsson and McAfee, 2014] Brynjolfsson, E. and McAfee, A. (2014). *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies*. W. W. Norton.

[Cadenat, 1999] Cadenat, V. (1999). *Commande référencée multi-capteurs pour la navigation d'un robot mobile*. Theses, Université Paul Sabatier - Toulouse III.

[Cai et al., 2014] Cai, C., Dean-León, E., Somani, N., and Knoll, A. (2014). 6d image-based visual servoing for robot manipulators with uncalibrated stereo cameras. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 736–742. IEEE.

[Čapek, 2004] Čapek, K. (2004). *RUR (Rossum's universal robots)*. Penguin.

[Chaumette, 1998] Chaumette, F. (1998). Potential problems of stability and convergence in image-based and position-based visual servoing. In *The confluence of vision and control*, pages 66–78. Springer.

[Chaumette, 2004] Chaumette, F. (2004). Image moments: a general and useful set of features for visual servoing. *Robotics, IEEE Transactions on*, 20(4):713 – 723.

[Chaumette and Hutchinson, 2006] Chaumette, F. and Hutchinson, S. (2006). Visual servo control. i. basic approaches. *Robotics & Automation Magazine, IEEE*, 13(4):82–90.

[Chaumette and Hutchinson, 2007] Chaumette, F. and Hutchinson, S. (2007). Visual servo control. ii. advanced approaches [tutorial]. *Robotics Automation Magazine, IEEE*, 14(1):109–118.

[Chen and Medioni, 1991] Chen, Y. and Medioni, G. (1991). Object modeling by registration of multiple range images. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 2724–2729 vol.3.

[Cheng et al., 2006] Cheng, Y., Maimone, M., and Matthies, L. (2006). Visual odometry on the mars exploration rovers - a tool to ensure accurate driving and science imaging. *Robotics Automation Magazine, IEEE*, 13(2):54 –62.

[Cherubini et al., 2013] Cherubini, A., Grechanichenko, B., Spindler, F., and Chaumette, F. (2013). Avoiding moving obstacles during visual navigation. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3069–3074.

[Cherubini et al., 2014] Cherubini, A., Spindler, F., and Chaumette, F. (2014). Autonomous visual navigation and laser-based moving obstacle avoidance. *Intelligent Transportation Systems, IEEE Transactions on*, 15(5):2101–2110.

[Chesi et al., 2004] Chesi, G., Hashimoto, K., Prattichizzo, D., and Vicino, A. (2004). Keeping features in the field of view in eye-in-hand visual servoing: a switching approach. *IEEE Transactions on Robotics*, 20(5):908–914.

[Chin, 2000] Chin, G. J. (2000). Flying along a logarithmic spiral. *Science*, 290(5498):1857.

[Choset et al., 2005] Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S. (2005). *Principles of Robot Motion*. MIT Press, Boston.

[Cobzas and Zhang, 2001] Cobzas, D. and Zhang, H. (2001). Mobile robot localization using planar patches and a stereo panoramic model. In *Vision Interface*, pages 04–99, Ottawa, Canada.

[Comport et al., 2010] Comport, A., Malis, E., and Rives, P. (2010). Real-time quadrifocal visual odometry. *International Journal of Robotics Research, Special issue on Robot Vision*, 29.

[Comport et al., 2004] Comport, A. I., Marchand, E., and Chaumette, F. (2004). Robust model-based tracking for robot vision. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 1, pages 692–697 vol.1.

[Corke and Hutchinson, 2001] Corke, P. I. and Hutchinson, S. A. (2001). A new partitioned approach to image-based visual servo control. *IEEE Transactions on Robotics and Automation*, 17(4):507–515.

[Crick et al., 2011] Crick, C., Jay, G., Osentoski, S., Pitzer, B., and Jenkins, O. C. (2011). Rosbridge: Ros for non-ros users. In *Proceedings of the 15th International Symposium on Robotics Research*.

[Damas and Santos-Victor, 2009] Damas, B. and Santos-Victor, J. (2009). Avoiding moving obstacles: the forbidden velocity map. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4393–4398.

[Danès et al., 2010] Danès, P., Coutinho, D. F., and Durola, S. (2010). *Multicriteria Analysis of Visual Servos through Rational Systems, Biquadratic Lyapunov Functions, and LMIs*, pages 169–188. Springer London, London.

[de la Croix, 2013] de la Croix, J.-P. (2013). *Describtion of the Sim I Am robotics simulator*.

[Degani and Wiener, 1991] Degani, A. and Wiener, E. L. (1991). Human factors of flight-deck checklists: the normal checklist.

[Demissy et al., 2016] Demissy, T., Dujols, L., Lafforgue, G., and Terrier, Y. (2016). Projet long report - visual servoing.

[Dick, 1982] Dick, P. K. (1982). Blade runner (do androids dream of electric sheep). 1968. *NewYork: Del Rey-Ballantine*.

[Dijkstra, 1971] Dijkstra, E. W. (1971). A short introduction to the art of programming.

[Drury, 2001] Drury, C. G. (2001). Human factors in aircraft maintenance. Technical report, DTIC Document.

[Eledath et al., 2007] Eledath, J., Bansal, M., Kreutzer, G., Das, A., Naroditsky, O., and English, W. (2007). Darpa urban challenge.

[Ferrucci et al., 2013] Ferrucci, D. A., Levas, A., Bagchi, S., Gondek, D., and Mueller, E. T. (2013). Watson: beyond jeopardy! *Artif. Intell.*, 199:93–105.

[Fiorini and Shiller, 1998] Fiorini, P. and Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772.

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

[Fleurmond, 2015] Fleurmond, R. (2015). *Asservissement visuel coordonné de deux bras manipulateurs*. PhD thesis, University of Toulouse, France.

[Folio, 2007] Folio, D. (2007). *Stratégies de commande référencées multi-capteurs et gestion de la perte du signal visuel pour la navigation d'un robot mobile*. PhD thesis, University of Toulouse, France.

[Folio and Cadenat, 2005] Folio, D. and Cadenat, V. (2005). A controller to avoid both occlusions and obstacles during a vision-based navigation task in a cluttered environment. In *European Control Conference*, pages 3898–3903, Seville, Espagne.

[Ford, 2015] Ford, M. (2015). *Rise of the Robots: Technology and the Threat of a Jobless Future*. Basic Books.

[Forsberg and Mooz, 1991] Forsberg, K. and Mooz, H. (1991). The relationship of system engineering to the project cycle. In *INCOSE International Symposium*, volume 1, pages 57–65. Wiley Online Library.

[Forster et al., 2014] Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE.

[Fox et al., 1997] Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33.

[Frey and Osborne, 2013] Frey, C. B. and Osborne, M. A. (2013). The future of employment: how susceptible are jobs to computerisation. *Retrieved September*, 7:2013.

[Futterlieb et al., 2014] Futterlieb, M., Cadenat, V., and Sentenac, T. (2014). A navigational framework combining visual servoing and spiral obstacle avoidance techniques. In *Informatics in Control, Automation and Robotics (ICINCO), 2014 11th International Conference on*, volume 02, pages 57–64.

[G. Palmieri and Callegari, 2012] G. Palmieri, M. Palpacelli, M. B. and Callegari, M. (2012). A comparison between position-based and image-based dynamic visual servoings in the control of a translating parallel manipulator.

[Gans and Hutchinson, 2007] Gans, N. R. and Hutchinson, S. A. (2007). Stable visual servoing through hybrid switched-system control. *IEEE Transactions on Robotics*, 23(3):530–540.

[Geraerts and Overmars, 2002] Geraerts, R. and Overmars, M. H. (2002). A comparative study of probabilistic roadmap planners. In *Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02)*, pages 43–57, Nice, France.

[Gerkey and Konolige, 2008] Gerkey, B. P. and Konolige, K. (2008). Planning and control in unstructured terrain. In *ICRA Workshop on Path Planning on Costmaps*.

[Grey@youtube.com, 2014] Grey@youtube.com, C. (2014). Humans need not apply.

[Hafez and Jawahar, 2007] Hafez, A. H. A. and Jawahar, C. V. (2007). Visual servoing by optimization of a 2d/3d hybrid objective function. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1691–1696.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination od minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107.

[Himmelman, 2002] Himmelman, J. (2002). *Discovering Moths: Nightime Jewels in Your Own Backyard*. Down East Books.

[Hutchinson et al., 1996a] Hutchinson, S., Hager, G., and Corke, P. (1996a). A tutorial on visual servo control. *IEEE Trans. on Rob. and Automation*, 12(5):651–670.

[Hutchinson et al., 1996b] Hutchinson, S., Hager, G., and Corke, P. (1996b). A tutorial on visual servo control. *Robotics and Automation, IEEE Transactions on*, 12(5):651–670.

[Ilon, 1975] Ilon, B. (1975). Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base. US Patent 3,876,255.

[Indiveri, 2009] Indiveri, G. (2009). Swedish wheeled omnidirectional mobile robots: Kinematics analysis and control. *Trans. Rob.*, 25(1):164–171.

[Jackson et al., 2004] Jackson, J. D., Yezzi, A. J., and Soatto, S. (2004). Tracking deformable moving objects under severe occlusions. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, volume 3, pages 2990–2995 Vol.3.

[Kavraki et al., 1996] Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566 –580.

[Kazemi et al., 2010] Kazemi, M., Gupta, K., and Mehrandezh, M. (2010). *Path-Planning for Visual Servoing: A Review and Issues*, pages 189–207. Springer London, London.

[Kelly, 1994] Kelly, A. (1994). An intelligent predictive controller for autonomous vehicles. Technical report, DTIC Document.

[Kelly, 2013] Kelly, A. (2013). *Mobile Robotics: Mathematics, Models, and Methods*. Cambridge University Press.

[Kermorgant and Chaumette, 2011] Kermorgant, O. and Chaumette, F. (2011). Combining ibvs and pbvs to ensure the visibility constraint. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2849–2854.

[Khatib et al., 1997a] Khatib, Jaouni, Chatila, and Laumond (1997a). Dynamic path modification for car-like nonholonomoic mobile robots. In *IEEE Int. Conf. on Robotics and Automation*, pages 490–496.

[Khatib et al., 1997b] Khatib, M., Jaouni, H., Chatila, R., and Laumond, J. P. (1997b). Dynamic path modification for car-like nonholonomic mobile robots. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 2920–2925 vol.4.

[Khatib, 1985] Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 500–505.

[Koenig and Likhachev, 2005] Koenig, S. and Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *Robotics, IEEE Transactions on*, 21(3):354–363.

[Koren and Borenstein, 1991] Koren, Y. and Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404 vol.2.

[Krösea et al., 2001] Krösea, B., Vlassisa, N., Bunschotena, R., and Motomura, Y. (2001). A probabilistic model for appearance-based robot localization. *Image and Vision Computing*, 19:381–391.

[Kurzweil, 2004] Kurzweil, R. (2004). The law of accelerating returns. In *Alan Turing: Life and Legacy of a Great Thinker*, pages 381–416. Springer.

[Kurzweil, 2005] Kurzweil, R. (2005). *The singularity is near: When humans transcend biology*. Penguin.

[Kurzweil, 2016] Kurzweil, R. (2016). The singularity is near1. *Ethics and Emerging Technologies*, page 393.

[Lamiraux et al., 2004a] Lamiraux, F., Bonnafous, D., and Lefebvre, O. (2004a). Reactive path deformation for nonholonomic mobile robots. *Robotics, IEEE Transactions on*, 20(6):967 – 977.

[Lamiraux et al., 2004b] Lamiraux, F., Bonnafous, D., and Lefebvre, O. (2004b). Reactive path deformation for nonholonomic mobile robots. *IEEE Transactions on Robotics*, 20(6):967–977.

[Latorella and Prabhu, 2000] Latorella, K. A. and Prabhu, P. V. (2000). A review of human error in aviation maintenance and inspection. *International Journal of Industrial Ergonomics*, 26(2):133–161.

[LaValle, 1998] LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. *In*, TR 98-11(98-11):98–11.

[Lem, 1973] Lem, S. (1973). *The Invincible: science fiction*. Sidgwick & Jackson.

[Lepetit and Fua, 2006] Lepetit, V. and Fua, P. (2006). Monocular based-model 3d tracking of rigid objects. found. *Trends. Comput. Graph. Vis*, 1.

[Levine, 1996] Levine, W. S. (1996). *The Control Handbook*. CRC Press Handbook.

[Li, 2009] Li, K.-C. (2009). *Handbook of Research on Scalable Computing Technologies*, volume 1. IGI Global.

[Mabrouk and McInnes, 2008a] Mabrouk, M. and McInnes, C. (2008a). Solving the potential field local minimum problem using internal agent states. *Robotics and Autonomous Systems*, 56(12):1050–1060.

[Mabrouk and McInnes, 2008b] Mabrouk, M. H. and McInnes, C. R. (2008b). An emergent wall following behaviour to escape local minima for swarms of agents. *International Journal of Computer*, 35(4):463–476.

[Maintenance,] Maintenance, A. Meeting 11: Human error in aviation maintenance (1997).

[Malin, 2013] Malin, J. L. (2013). Envisioning watson as a rapid-learning system for oncology. *Journal of Oncology Practice*, 9(3):155–157.

[Malis, 2008] Malis, E. (2008). Méthodologies d'estimation et de commandea partir d'un systeme de vision. *Habilitation. Nice-Sophia Antipolis*.

[Malis et al., 1999] Malis, E., Chaumette, F., and Boudet, S. (1999). 21/2d visual servoing. *Robotics and Automation, IEEE Transactions on*, 15(2):238–250.

[Mantegh et al., 2010] Mantegh, I., Jenkin, M. R., and Goldenberg, A. A. (2010). Path planning for autonomous mobile robots using the boundary integral equation method. *Journal of Intelligent & Robotic Systems*, 59(2):191–220.

[Marder-Eppstein et al., 2010] Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., and Konolige, K. (2010). The office marathon: Robust navigation in an indoor office environment. In *International Conference on Robotics and Automation*.

[Marquez-Gamez, 2012] Marquez-Gamez, D. (2012). *Towards visual navigation in dynamic and unknown environment: trajectory learning and following, with detection and tracking of moving objects*. PhD thesis, Toulouse University.

[Martinez and Fernández, 2013] Martinez, A. and Fernández, E. (2013). *Learning ROS for robotics programming*. Packt Publishing Ltd.

[McBride, 2007] McBride, J. (2007). Darpa urban challenge.

[Mcfadyen et al., 2012a] Mcfadyen, A., Corke, P., and Mejias, L. (2012a). Rotorcraft collision avoidance using spherical image-based visual servoing and single point features. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1199–1205.

[Mcfadyen et al., 2012b] Mcfadyen, A., Mejias, L., and Corke, P. (2012b). Visual servoing approach to collision avoidance for aircraft. In *28th Congress of the International Council of the Aeronautical Sciences 2012*, Brisbane Convention & Exhibition Centre, Brisbane, QLD.

[Mebarki et al., 2010] Mebarki, R., Krupa, A., and Chaumette, F. (2010). 2-d ultrasound probe complete guidance by visual servoing using image moments. *IEEE Transactions on Robotics*, 26(2):296–306.

[Mezouar and Chaumette, 2002] Mezouar, Y. and Chaumette, F. (2002). Avoiding self-occlusions and preserving visibility by path planning in the image. *Robotics and Autonomous Systems*, 41(2–3):77 – 87. Ninth International Symposium on Intelligent Robotic Systems.

[Moore, 1965] Moore, G. E. (1965). Cramming more components onto integrated circuits, electronics, volume 38, number 8, april 19, 1965. *Also available online from ftp://download. intel. com/research/silicon/moorespaper. pdf*.

[Mur-Artal et al., 2015] Mur-Artal, R., Montiel, J. M. M., and Tardós, J. D. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system. *CoRR*, abs/1502.00956.

[Nister et al., 2004] Nister, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–652 – I–659 Vol.1.

[Okabe et al., 2000] Okabe, A., Boots, B., Sugihara, K., and Chiu, S. N. (2000). *Spatial Tessellations - Concepts and Applications of Voronoi Diagrams*. John Wiley.

[O'Kane, 2014] O'Kane, J. M. (2014). A gentle introduction to ros.

[Online, 2016] Online, S. S. (2016). Wage statistics for 2014.

[Owen and Montano, 2005] Owen, E. and Montano, L. (2005). Motion planning in dynamic environments using the velocity space. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2833–2838.

[Paletta et al., 2001] Paletta, L., Frintrop, S., and Hertzberg, J. (2001). Robust localization using context in omnidirectional imaging. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 2072 – 2077 vol.2.

[Pérez-D'Arpino et al., 2008] Pérez-D'Arpino, C., Medina-Meléndez, W., Fermín, L., Guzmán, J., Fernández-López, G., and Grieco, J. C. (2008). Dynamic velocity field angle generation for obstacle avoidance in mobile robots using hydrodynamics. In *Advances in Artificial Intelligence–IBERAMIA 2008*, pages 372–381. Springer.

[Petiteville, 2012] Petiteville, A. D. (2012). *Navigation référencée multi-capteurs d'un robot mobile en environnement encombré*. PhD thesis, University of Toulouse, France.

[Petiteville et al., 2013] Petiteville, A. D., Durola, S., Cadenat, V., and Courdesses, M. (2013). Management of visual signal loss during image based visual servoing. Zurich, Switzerland.

[Quigley et al., 2009] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5.

[Rasmussen and Vicente, 1989] Rasmussen, J. and Vicente, K. J. (1989). Coping with human errors through system design: implications for ecological interface design. *International Journal of Man-Machine Studies*, 31(5):517–534.

[Rösmann et al., 2012] Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F., and Bertram, T. (2012). Trajectory modification considering dynamic constraints of autonomous robots. In *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, pages 1–6. VDE.

[Rosmann et al., 2013] Rosmann, C., Feiten, W., Wosch, T., Hoffmann, F., and Bertram, T. (2013). Efficient trajectory optimization using a sparse model. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 138–143. IEEE.

[Royer et al., 2007] Royer, E., Lhuillier, M., Dhome, M., and Lavest, J.-M. (2007). Monocular vision for mobile robot localization and autonomous navigation. *International Journal of Computer Vision*, 74(3):237–260.

[Samson et al., 1991] Samson, C., Espiau, B., and Borgne, M. L. (1991). *Robot control: the task function approach*. Oxford University Press.

[Segvic et al., 2009] Segvic, S., Remazeilles, A., Diosi, A., and Chaumette, F. (2009). A mapping and localization framework for scalable appearance-based navigation. *Computer Vision and Image Understanding*, 113(2):172–187.

[Shappell and Wiegmann, 2012] Shappell, S. A. and Wiegmann, D. A. (2012). *A human error approach to aviation accident analysis: The human factors analysis and classification system*. Ashgate Publishing, Ltd.

[Sharma and Sutanto, 1997] Sharma, R. and Sutanto, H. (1997). A framework for robot motion planning with sensor constraints. *IEEE Transactions on Robotics and Automation*, 13(1):61–73.

[Shin and Ramanathan, 1994] Shin, K. G. and Ramanathan, P. (1994). Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24.

[Shirai and Inoue, 1973] Shirai, Y. and Inoue, H. (1973). Guiding a robot by visual feedback in assembling tasks. *Pattern recognition*, 5(2):99IN3107–106108.

[Siegwart and Nourbakhsh, 2004] Siegwart, R. and Nourbakhsh, I. (2004). *Introduction to autonomous mobile robots*. A bradford book, Intelligent robotics and autonomous agents series. The MIT Press.

[Sim and Dudek, 1999] Sim, R. and Dudek, G. (1999). Learning visual landmarks for pose estimation. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 3, pages 1972 –1978 vol.3.

[Sterela and Desfosses,] Sterela and Desfosses, A. Sterela.

[Strickland and Guy, 2013] Strickland, E. and Guy, E. (2013). Watson goes to med school [2013 tech to watch]. *Spectrum, IEEE*, 50(1):42–45.

[Tamadazte et al., 2012] Tamadazte, B., Piat, N. L. F., and Marchand, E. (2012). A direct visual servoing scheme for automatic nanopositioning. *IEEE/ASME Transactions on Mechatronics*, 17(4):728–736.

[Teimoori and Savkin, 2008] Teimoori, H. and Savkin, A. (2008). A method for collision-free navigation of a wheeled mobile robot using the range-only information. In *American Control Conference, 2008*, pages 1418–1423.

[Thrun et al., 2000] Thrun, S., Burgard, W., and Fox, D. (2000). A real time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In *IEEE International Conference on Robotics and Automation*, San Francisco, CA, USA.

[Ulrich and Borenstein, 1998] Ulrich, I. and Borenstein, J. (1998). Vfh+: Reliable obstacle avoidance for fast mobile robots. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1572–1577. IEEE.

[Ulrich and Borenstein, 2000] Ulrich, I. and Borenstein, J. (2000). Vfh*: local obstacle avoidance with look-ahead verification. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 3, pages 2505–2511 vol.3.

[van den Berg and Overmars, 2005] van den Berg, J. P. and Overmars, M. H. (2005). Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897.

[Victorino and Rives, 2004] Victorino, A. and Rives, P. (2004). An hybrid representation well-adapted to the exploration of large scale indoors environments. In *IEEE International Conference on Robotics and Automation*, pages 2930–2935, New Orleans, USA.

[Wikipedia, 2015] Wikipedia (2015). Watson (computer) — wikipedia, the free encyclopedia. [Online; accessed 24-October-2015].

[Wolf et al., 2002] Wolf, J., Burgard, W., and Burkhardt, H. (2002). Robust vision-based localization for mobile robots using an image retrieval system based on invariant features. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 1, pages 359 – 365 vol.1.

[Zhang, 1994] Zhang, Z. (1994). Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vision*, 13(2):119–152.

[Zhang et al., 2012] Zhang, Z.-h., Xiong, Y.-s., Fan, P., Liu, Y., and Cheng, M. (2012). Real-time navigation of nonholonomic mobile robots under velocity vector control. *International Journal of Advanced Robotic Systems*, 9.