UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Development of a local positioning system for builder drones with image processing

Dissertation presented by
Thibault JACQUES , Frédéric KACZYNSKI

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Pierre LATTEUR, Ramin SADRE

Reader(s)
Sébastien GOESSENS, François WIELANT

Academic year 2017-2018

Abstract

This master thesis is achieved in the context of the research led by Pr. Latteur and his team in the civil engineering department of the Université Catholique de Louvain whom are exploring the usage of UAVs in the field of the construction, which has never known a real revolution in the last decades.

After the elaboration of new structural elements "drone-compatible" and a UAV, developing a positioning system will allow the drone to be autonomous and independent to achieve specific missions. In this purpose, a general positioning system combining different solutions must be elaborated.

In this report, we will focus on the usage of the image processing for local positioning of the structural elements from which the drone will be able to detect it and to localize it in the purpose to transport it.

Acknowledgements

We would like to thank all the people who made it possible for us to carry out the project successfully:

- Pr. Pierre Latteur and Sébastien Goessens for their availability and for sharing their knowledge about civil engineering.
- Pr. Ramin Sadre for his feedbacks allowing us to step back about the work accomplished.
- The company ALX Systems for their support and their help about the drone control.
- Remy Vermeiren, Nicolas Sorensens and Zakariya Bouhaddi, with who we shared the work during this year, for sharing their knowledge about the project and their progress.
- Our families for their encouragement and their support throughout these years of study and through the process of researching and writing this thesis.

Contents

CHAPTER 1

Introduction

This master thesis is achieved in the context of the research led by Pr. Latteur and his team in the civil engineering department of the Université Catholique de Louvain whom are exploring the usage of UAVs in the field of the construction that has never known a real revolution in the last decades.

The objective is to develop a positioning system using computer vision to detect and localize the position of structural elements that the UAV must transport. The work will be integrated into a positioning system that aggregates different solutions allowing a drone to be autonomous and independent to achieve specific missions in order to prove the feasibility of the usage of autonomous drones in the construction.

The first part of the report will be an overview of the context in which the master thesis is achieved because the research about the usage of drones is in progress and in the construction field too.

After that, a chapter will be dedicated to the global positioning system. The objective is to describe the positioning framework set up and the function of the image processing. Moreover, the requirements, the constraints and the materials available will be approached.

The Chapter 4 and 5 explain the image processing techniques used during the project and how they were combined to reach the goal.

The next chapter evaluates the quality of the positioning with the advantages and the limits.

Finally, the last two chapters describe the architecture deployed for the control of the drone and the role of the company ALX Systems as well as the strategies envisioned to have an autonomous drone that handles construction missions.

CHAPTER 2

Context of the project

2.1 The project and the objectives

Pr. Latteur and his team of the Université Catholique de Louvain-La-Neuve (UCL) with the collaboration of Massachusetts Institute of Technology (MIT) has decided to consider possible solutions to revolutionize the industry of the construction and they are currently exploring the solution of the UAVs since 2015.

Indeed, the current way of building suffers from different difficulties:

- The painfulness of work that is reflected in the number of work accidents.
- Human errors due to a lack of communication between the different teams on a project or a lack of attention, leading to wasted resources and hence a loss of money.

Moreover, construction is a field that has never known a real revolution in the last decades. Indeed, the tools are more evolved but the techniques are similar haven't much changed. This is why some research is currently in development to benefit from the different advancement in the autonomous machine. Indeed, they want to develop different autonomous systems able to construct without real interaction with humans on the site and only with other systems. Firstly, it will reduce the number of work accidents but moreover, it will optimize the cost of construction on the site.

There are three axes for the research :

- Modeling the construction from a software point of view to be able to give all the information to the autonomous systems. This part is quite advanced with the concept of Building Information Modelling (BIM). As explained by the Department of Design & Construction of New York City[15], Building Information Modeling (BIM) refers to a digital collection of software applications designed to facilitate coordination and project collaboration. It is considered as a way to construct the building virtually on the computer before actually building it. Moreover, BIM is multidimensional because it doesn't only take into account the 3D modelling but also the time, the budget, etc. From this modelling and the information imported into it, a computer might also be able to optimize the logistics of the building process more easily than a human might do it.
- Developing autonomous machines capable of understanding the modelling and construct the

plan on site. It asks for more resources because it combines different fields of engineering like electromechanics, computer science and construction.

- Developing new structural elements compatible with the new autonomous machines because new constraints must be handled.

2.1.1 Advantages/Disadvantages of the UAVs in the construction

As explained previously, to revolutionize the construction, it needs to develop autonomous systems that are able to build structures. With the growth of the development of the UAVs and the better performance, it is logical to reflect on the possible uses of those in the field.

The first advantage is that the UAV doesn't take into account the nature of the land, compared to ground vehicles. Indeed, it doesn't have to adapt if it is on the tarmac or on the soil. Moreover, multiple UAVs can be deployed on the same site and take benefit of air-space and not only the space on land as a machine with wheels. Furthermore, the UAV is not limited in height. It can work correctly at different altitudes.

However, the UAV is limited by the load that it can transport and it can have problems with high precision due to climatic conditions, for example, it can be sensitive to the wind.

This is why the processes of construction have to be adapted to the UAVs in parallel of the development of UAV able to carry a huge load.

2.2 Current research and existing uses of UAV

Scientific teams from other universities have already prospected the use of UAVs. This section will explore and observe the progress and the contributions of those projects in the construction field.

2.2.1 Universität Stuttgart

MAV Assisted Fabrication Strategy for Large Scale Fiber-Composite Structures

In Stuttgart, a team[32] led a project to guide UAV using a total station for the purpose of the architectural fabrication process. They reached the conclusion that the guidance of UAVs by robotic total stations is possible and enables the exploration of on-site automated fabrication processes for architectural structures. They approved the guidance system in the context of a full-scale filament winding experiment. This is typically a similar solution that will be used in the general positioning system.

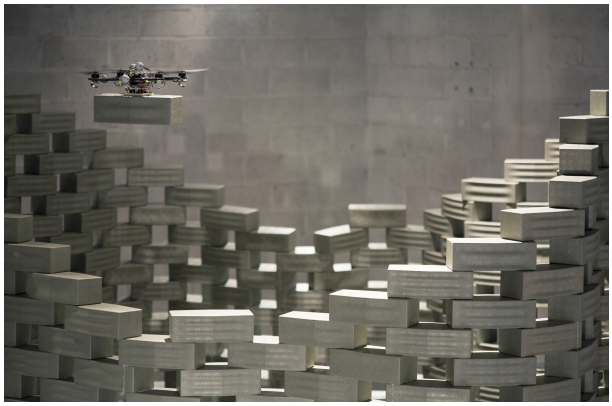
2.2.2 Ecole Polytechnique Fédérale de Zurich

The Flight Assembled Architecture Installation

The *Flight Assembled Architecture*[13] is one of the first structure built with flying vehicles. The objective was to conceive a 1:100 scale model of a 600m tall vertical village. This is why they assembled a 6m tall tower composed of foam modules constructed by four drones in 18 hours.

This work is a proof that digital technologies can be used to enhance the design and the assembly of structures. Concerning the technology used for the position, they installed a motion capture by deploying a system composed of 19 cameras to provide high-accuracy position and attitude

Figure 2.1: Example of experiments led during the project



(a) UAV building the tower.

(b) Obtained tower.

Figure 2.2: The Flight Assembled Architecture Installation

information for all quadrocopters in the space. The huge drawback is to install the system around the structure.

Building Tensile Structures with Flying Machines

The project is based on the assumption that flying machines can offer advantages compared to the traditional construction machines. For example, they can reach any point in space and fly in or around existing object.[14] However, it has drawbacks: the accuracy and the limited load may not be as good as traditional manned machines.

They led the project to construct a rope bridge that can support the weight of a person crossing it, assembled by quadcopters. The bridge is spanning 7.4 meters between two scaffolding structure. Concerning the technology for the positioning, they used the same kind of system as the previous example.

(a) UAVs building the rope bridge.

(b) Proof of load of the bridge.

Figure 2.3: Building Tensile Structures with Flying Machines

University of Pennsylvania

Assembly of Structures for Construction with Multiple Quadrotors

A team[30] from the Department of Mechanical Engineering and Applied Mathematics of the University of Pennsylvania developed a system to assemble cubic structures with a swarm of UAVs.

They showed that a team of aerial robots are able to construct structures autonomously from simple structural nodes and members using carefully-designed assembly modes compatible with the part design. They didn't use positioning system because they work relies on a static environment in which absolute positions and orientations of parts and fixtures remain unchanged so that industrial robots can be programmed to pick, place and assemble the structure.

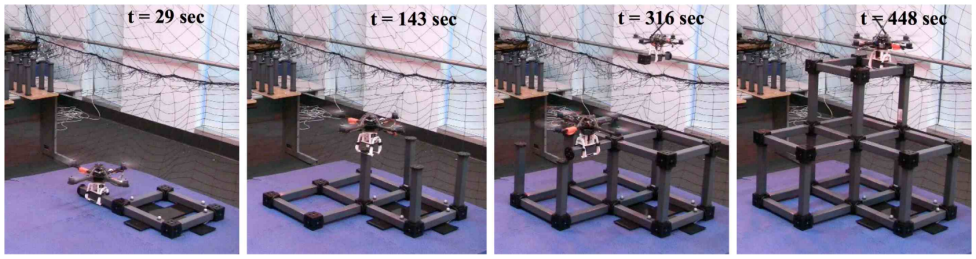


Figure 2.4: Example of UAVs building cubic structures

2.3 UAVs in the construction by UCL

Based on these observations, Pr. Latteur and his team of the Université Catholique de Louvain-La-Neuve (UCL) began working on UAVs with the goal of using them to build structures.

As explained in "Drone-Based Additive Manufacturing of Architectural Structures"[25], they are working on a new building process that would automate some of the tasks related to the design of the build, and allow the use of UAVs :

1. Designing the building by the architect and the engineer
2. Modeling the building into a BIM tool
3. Translating the BIM model into remote control instructions compatible with the drones
4. Assembling the structure with the drones

In contrast to the previous research, different projects are principally driven by new structural elements compatible with drones. Moreover, they investigated the development of an autonomous UAV able to transport these new structural elements to prove the feasibility of the solution.

2.3.1 Development of "drone-compatible" elements

In 2015, J-S. Breton and J. Leplat examined the feasibility of drone-based manufacturing of architectural structure.[29] They defined the required specifications in the context of the usage of drones in the construction.

In 2016, C. Coppieters de Gibson[19] examined the feasibility for drone-based manufacturing of architectural structure from an economic point of view. They investigated new designed as illustrated in the Figure 2.5.

In parallel, M. Reniers[37] developed a BIM interface for the construction drone-compatible to allow a easier development of structure drone-compatible

Finally, A. Naveau and A. Moncourrier[34] developed the first drone-compatible block and a catching system for the UAV. It is the premise of the blocks used in this master thesis.

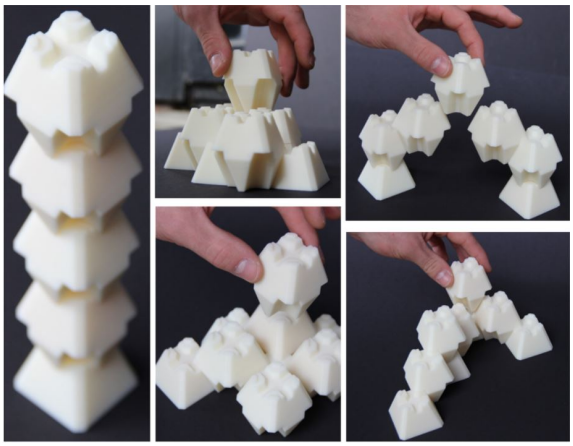


Figure 2.5: Examples of structures using drone-compatible blocks: droxels

In 2017, C. Manderliert and T. De Furstenberg[31] investigated the feasibility of drone-compatible wooden structure. Meanwhile, J-F. Leboutte and V. Parisel[27] were continuing the work about the first drone-compatible block. They developed the block with which we have worked this year and confirmed the accuracy threshold of 5 cm in each direction to place the block. It is called "Drick" and is designed for the building of single-family homes.

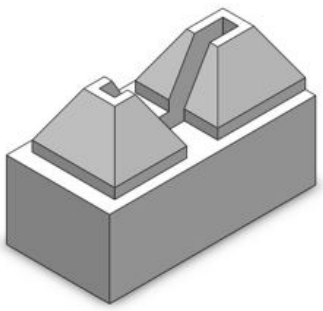


Figure 2.6: Example of drick model

This year, in parallel of our work, the research in the civil engineering department was continuing. For example, two students were exploring the parametric design of "drone-compatible" architectural timber structures[26].

As you can notice, a rich research environment is developed for the usage of drone in construction by the civil department of the Université Catholique de Louvain-La-Neuve.

2.3.2 Development of the autonomous UAV

As explained before, the civil engineer department had to design a UAV that can carry a heavy load. With an external collaboration, they conceived a drone, as illustrated on the Figure 2.7, capable of carrying a theoretical load of 80 kg. In practice, for security reasons, it can carry structural elements of 40 kg maximum. For this master thesis, it is the drone on which the positioning system has to be deployed.

Figure 2.7: Drone built during the project

Meanwhile, in the context of a master thesis, two students (N. Nehri & A. Paques) elaborated an indoor localization system based on fixed lasers on the drone[35]. While the results were persuasive, the problem of the solution is the difficulty of usage. Indeed, it is conceived only for indoor because the construction environment needed to be framed by screens to measure the distance with the lasers.

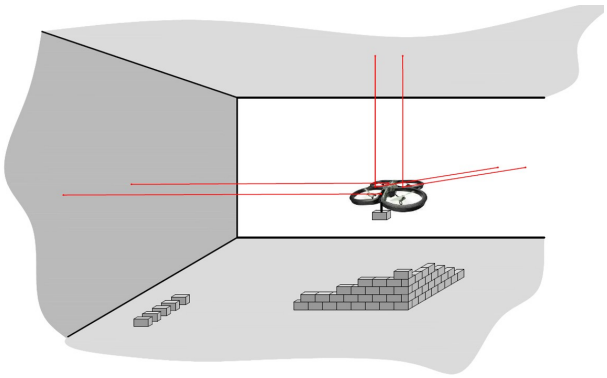


Figure 2.8: Schema of the solution using lasers

2.4 The next steps

After positive tests with a pilot and in order to prove the possible automation of the drone in the construction field, the drone needs to have a reliable positioning system easier to deploy than the solution explained in the previous section.

Therefore, we worked this year on a general positioning system that combines different system in order to guide the drone to realize the task it has been attributed.

The following chapter will present the different positioning systems developed this year and the different constraints and resources linked to the project.

CHAPTER 3

Positioning system overview

3.1 Development of a positioning framework

This year, our work is integrated into a research about methods of localization usable by the UAV builder. The purpose is to obtain an autonomous drone that can construct a simple structure with drone-compatible elements from the previous research. An interesting way to have a robust positioning framework is to combine different positioning system to benefits of the advantages of each.

To have a real interest, the positioning system needs to respect a constraint of 5cm of accuracy in each direction required by structural elements. Moreover, the flight tests are effectuated in a dedicated zone ("DroneZone") of the university. So, the system has to work in an indoor environment as well as an outdoor environment.

Currently, three systems can meet the expectations:

- global localization with a total station
- global localization with Ultra Wideband (UWB) system
- local localization with image processing

Before developing more precisely the solution of local positioning with image processing, it is necessary to explain quickly how the different solutions work, their advantages and disadvantages.

Global localization with a total station

The total station is an instrument that can measure the angles and the distance between itself and a particular point. The accuracy is estimated at less than 1 cm. In general, it is used to measure distance on site. In the context of the master thesis, a prism is locked on the UAV and the information of the position of the UAV is sent to it.

The huge advantage of the solution is the accuracy. However, the system can't give the position of the drone if one object is obstructing the view between the total station and the drone. Moreover, if the total station is not fast enough to follow the drone, it can lose its tracking and stop providing positioning information to the drone. While this system is really powerful, it is still sensible to disturbances.

Figure 3.1: Example of total station: Leica Viva TS12

Furthermore, with only one prism, the system can't give information about the orientation of the UAV. It has to refer to another system or to the sensors of the UAV.

Global localization with Ultra Wideband (UWB) system

The ultra wideband system is an indoor positioning system based on ultra-wideband (UWB) radio signals. After installing different transmitter (called anchors), the localization of the tag is computed from the triangulation of the position with the different anchors. This distance is obtained from the time the signal takes to arrive at the tag and return to the anchor.

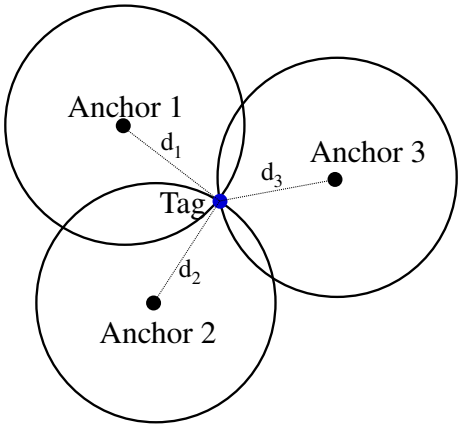


Figure 3.2: Example of UWB principle

One major advantage is the robustness of the solution because, in contrast with the total station, it doesn't require a line of sight between the tag and the anchors. However, the accuracy is influenced by the environment and the materials around. Furthermore, the accuracy is around 10 cm, which is above our requirement of 5cm. It can be a good solution for long flights that doesn't need an excellent accuracy.

Once again, this system is not able to give the orientation of the UAV with respect to the brick.

Local localization with image processing

The solution with image processing is restricted to a local localization. Indeed, it doesn't correspond to a global position as the simultaneous localization and mapping (SLAM) solution where the robot can create a map of its environment from the images of the cameras and so be autonomous in its environment to avoid obstacles.

Here, the objective is to localize the brick around the UAV to be able to give the instructions in order to catch it and deposit it at the right place. So, the major disadvantage is the fact that the solution can't work alone. Indeed, without first seeing a brick in the image feed, the system can't give instruction to the drone.

However, the solution can give a good feedback of the accuracy of the two other solutions during the phase of the catch and the deposit. Moreover, with the image processing, the drone will be able to orient itself with respect to the brick, something that wasn't possible with the other systems.

3.2 Constraints

During the project, we had to take into account some constraints to have the best work that matches with the expectations of the environment of the project.

- **Accuracy :** the maximal error accepted was evaluated to 5 cm in each direction in previous works as explained in the subsection 2.3.1. So, the positioning system has to respect this requirement. Otherwise, the structures developed in the civil engineering department will not be compatible with the drone.
- **Adaptability :** the positioning system must be adaptable to other kinds of materials, even by a person with little knowledge in computer science. The purpose is to have a user-friendly solution for civil engineers who are developing the structures without having to reconsider the positioning system of the drone.
- **Least restrictive :** the positioning system must try to be the less restrictive as possible from the point of view of the elements of construction. The best solution for the image processing part is a solution without modification of the elements of construction for example.

It is interesting to see the antagonism between the two last constraints. A less restrictive solution can lead to a decrease in the adaptability of the project. Indeed, if we have to detect diamond but also rectangles, we can't use the specific properties of each but we can use the mutual properties of the quadrilaterals. This is why it would be interesting to find a good comprise between the two constraints and not be restricted to one structural element.

3.3 Resources

During the project, we had access to different resources, both from a hardware and software point of view. In some cases, they influenced our solutions and this influence will be explained during the report.

3.3.1 Company collaboration

ALX Systems

The project is realized in collaboration with the Belgian company ALX Systems.[1] Created in 2016, this young start-up has for objective to build a framework able to easily control a drone in different situations and for different purposes. They provide expertise in drone control but also in drone building and computer vision. Indeed, they are developing a secure operating system

for drones called ALX Operating System and a solution of computer vision called ALX Vision. Thanks to the collaboration, we can benefit from their knowledge in this new environment. Moreover, it allows us to focus on the positioning system, since the control of the flight is ensured by their system. However, it brings some constraints because we have to use the PixHawk flight controller and the computer vision part has to be coded in C++ in order to be compatible with their framework.

3.3.2 Hardware

DroneZone

A dedicated zone is established at the university for every drone flights. It is called the "DroneZone" and it allows to test safely the drone thanks to the safety net surrounding it. Moreover, this space can be used to verify the quality of the positioning system because it is a real flight environment. The zone offers a surface of 12x15m approximately.

Dricks

As explained before, obtaining an accurate positioning system working with elements "drone compatible" is the main objective. For this purpose, we are using, as an example of a construction element, the "Dricks" (cf. Figure 2.6) designed by previous civil engineer students.[27] They can be assembled in the same manner as the famous Lego® bricks. However, they are adapted for the drone with a threshold of the accuracy of approximately 5cm and they can be caught by the drone using an electromagnetic system. Moreover, different flights with a pilot were done with these blocks and the results were convincing as you can observe on the Figure 3.3

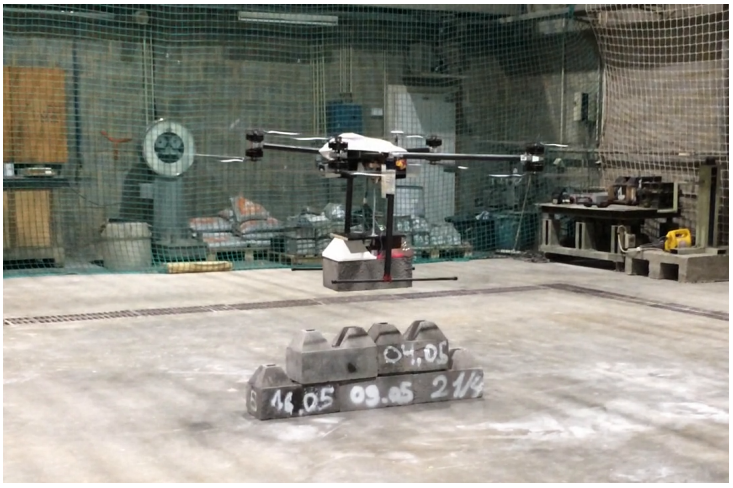


Figure 3.3: Drone building wall of dricks

PixHawk

As explained before, we have to use the flight controller PixHawk to be compatible with the framework developed by ALX Systems. The PixHawk flight controller has the advantage of being open source and so modifications to the code can be quickly effected if necessary. Moreover, a large community contributes to the project, which means that resources on the project can easily be found on the internet, for example.

Odroid XU4

Besides the flight controller, we also embed a single-board computer on the drone to perform the image processing algorithms. While it could be possible to run those algorithms on a ground station and transfer the images from the drone to the station through wireless communication. The delay it would add to the whole pipeline is not desirable as the drone would receive the positioning information with an even bigger delay. In our case, we use the same embedded computer as the previous work: the Odroid XU4.

More powerful than a Raspberry but still at an affordable price, the Odroid XU4 is a good compromise for our project. Its 8-core CPU allows us to exploit the parallel nature of some of the algorithm we used. A comparison with the well-known Raspberry Pi is effectuated in the Appendix A.

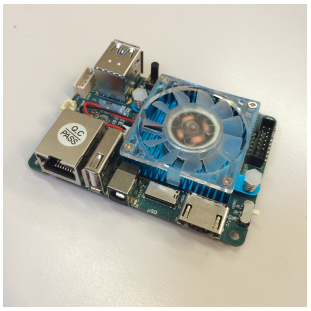


Figure 3.4: Odroid XU4

Drones

For the drones, we worked on different models available in the civil engineering department. They are the fruit of the work of the previous year in the context of the usage of the drones in the construction.

3 different drones have been developed :

- **A small drone** with 6 rotors, illustrated at the Figure 3.5a. We can implement the positioning system (Odroid XU4 + camera + prism + UWB tag) on it but it is not able to carry a brick because it would too heavy. It still allows us to test the positioning system easily.
- **A medium drone** with 8 rotors, illustrated at the Figure 3.5b. As the small drone, it can support the positioning system and while no carrying system has been installed on it, it has enough power to carry a heavier load and possibly a lighter version of the current brick.
- **A big drone** with 8 rotors. This drone is the targeted drone for the positioning system because it is developed to carry structural elements of 40 kg as explained in Section 2.3.2 and illustrated at the Figure 2.7. Moreover, catching system is already installed on it.

Cameras

For this project, we used a Logitech HD Webcam C270 as illustrated in Figure 3.6. Our first proofs-of-concept showed that we didn't necessarily need a better resolution than what regular webcams can offer. They also have the advantage of being simple to connect (USB) to the Odroid and easy to use with OpenCV.

(a) Small drone (b) Medium drone

Figure 3.5: Illustration of the drones of the civil engineering department

Figure 3.6: Logitech HD Webcam C270[8]

This webcam has a picture resolution of 1080p, a field-of-view of 60° and offers a maximum framerate of 30fps at 480p.[9]

3.3.3 Software

OpenCV

Concerning the image processing, we used a well-known library in the field: OpenCV[11] (Open Source Computer Vision Library). It is a released under a BSD license and hence it's free for both academic and commercial use. Moreover, it supports different languages such as C++, Python and Java interfaces and platforms such as Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Moreover, the library is adopted all around the world by more than 47 thousand people. The large community allows us to benefit well-explained tutorials and a detailed documentation.

Drone control system of ALX Systems (ALX Operating System)

The company ALX Systems is developing a control system for drones above the flight controller Pixhawk. The objective is to propose an easy framework for the control of the drones, able to be deployed in different situations and environment. In our case, they will handle the control of the drone and thus we can focus our work on the positioning system that will give input to the system. It will be more described in the dedicated Chapter 7.

3.4 Structure of the development of a local positioning by image processing

To summarize, the objective of this master thesis is to use image processing in order to localize the drick and give the required information to the drone to build a wall of dricks. Note that the first-floor wall of dricks is already installed as illustrated in the Figure 3.7. The system conceived during this master thesis can be split into different subtasks :

- Detection of the bricks
- Compute the position of the bricks with respect to the UAV
- Determine the strategy the drone should follow to pick up and drop the brick.

Moreover, the major constraint to take into account is the accuracy. The drone-compatible structure allows for a maximal accuracy error around 5 cm. So, aside from making sure the system doesn't detect a nonexistent brick, the system also must be accurate when measuring the position of a brick.

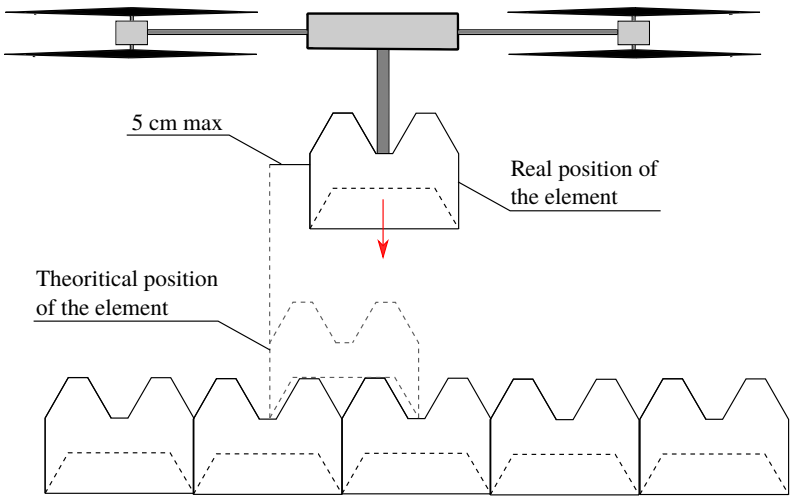


Figure 3.7: Illustration of the situation

CHAPTER 4

Detection

4.1 Objectives

The objectives of this sub-task are to extract the list of bricks present in the image. After investigating different solutions, two methods have been selected to accomplish the task:

- A solution based on color points placed on the bricks.
- A solution based on ArUco code.

These two solutions work differently, but both yields a list of bricks, composed of a list of extracted "feature" points that are used by the positioning phase to properly locate the bricks in the 3D space.

In this chapter, the constraints and the hypotheses will be approached before to explain the investigating solutions. Finally, the two selected techniques will be explained more deeply

4.2 Constraints and hypotheses

In order to develop a solution for the detection task, we must face some constraints and take some hypotheses.

Firstly, the output of the detection will be used by the positioning of the different bricks. During the positioning step, as it will be explained later in the Chapter 5, the algorithm will match the information of the 2D images with the known 3D model of the brick and, thus, estimate the position with respect to the camera. For this purpose, the output of the detection task must be accurate to have a correct positioning.

Moreover, the camera will be embedded on the drone. It means that the image can be blurred by the quick movement of the drone or because of the vibrations of the motors. In our case, we have taken the hypothesis that the drone will be in a suitable environment, without wind, and that the perturbations don't impact greatly the quality of the image. Indoor tests with a camera on the drone confirmed this hypothesis.

On top of that, in the context of an autonomous drone, the speed of it must be low enough to not cause any motion blur.

Furthermore, another hypothesis we have taken is the fact that tilting the camera doesn't impact the detection phase. Indeed, the catching system on the drone can obstruct the video feed taken by the camera. Because of this, we can't point the camera directly to the ground, as the low field-of-view would restrict the visible area. To solve this, the camera can be placed with a tilt on the drone.

Finally, during the detection phase, we focus our investigation on a solution for the drick. Therefore, once a solution is found, it has to be evaluated about its adaptability to other elements.

4.3 Investigated solutions

The solutions for the detection can be divided into two classes of solutions. The first class of solutions are the ones without modifying the drick, thus less restrictive whereas the other class are the solutions with modification of the drick by adding patterns on it for example. However, we have to keep in mind the discussion about the antagonism in the Section 3.2 about the constraints of the project between the adaptability of the system and the fact that is the less restrictive.

4.3.1 Features detection

One option to detect the known object is to compare one image of it with what is seen in the frame. The comparison from an image processing point of view is based on salient points. They are singularities that can be detected repeatable under different points of view. Note that the salients points detected by image processing are not directly related to visual perception.

Concerning the detection, it will try to match the salients point from the training image with these of the frame to find the drick.

However, in our case, the technique suffers from two huge problems. Firstly, the dricks are composed of concrete and don't have interesting salient points or they are details of the dricks due to imperfections and they are not present in others. Indeed, the different dricks are not uniform and not perfectly the same, causing problems for the matching.

Furthermore, the ground of the DroneZone is composed of concrete too. Salient points of the drick can thus be similar to blemishes of the ground.

This solution can be interesting only if particular patterns are present on top of the construction elements for example.

4.3.2 Training a brick detector

A common solution for the detection task means training an algorithm to detect if an object is present or not on a frame. Nowadays, the most famous technique is certainly the deep learning with the convolutional neural network (CNN).

Without entering into the details of implementation, the convolutional neural network is a variation of multilayer perceptrons using image processing techniques. While the principle of the neural network is not new, it only saw a resurgence in the image processing world in 2012 when Krizhevsky, Sutskever and Hinton used a similar network to classify images amongst a thousand categories in the ImageNet LSVRC-2010 contest[24], achieving an error rate of only 16% (from 25%). From a simplified point of view, the algorithm is fed a huge number of images with objects that you want to detect as well as whether a brick is appearing in each image or not. After the

training phase, you can evaluate new images and the algorithm will be able to detect if they contain wanted objects. To apply that to the project, a dataset of the drick from different points of view and in different environments must be created.

However, this solution has different problems. Firstly, it will be overkill and difficult to set up. Indeed, if the system takes into account only the drick, it will have only one class of detection: "It is a drick" or "It is not a drick". Moreover, in the future, if new construction element must be added, a new training set must be produced, the algorithm must be trained again and adapted to have a correct behavior.

Furthermore, the most important problem is the accuracy. Indeed, normally, the algorithm will evaluate if a drick is present or not but it will not localize it on the image. To obtain the information, the current algorithms localize the object in a window as you can observe on the Figure 4.1 but it will not be enough accurate for the second phase, the positioning phase, what is more, if the camera is tilted.

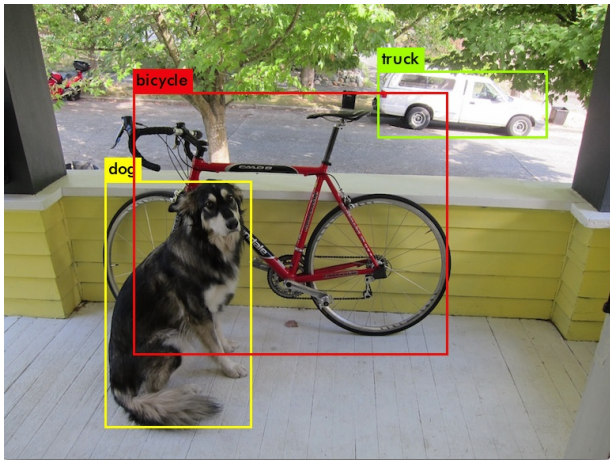


Figure 4.1: Example of detection with the CNN YOLOV3[36].

4.3.3 Edge detector

Another solution is to determine the positioning of the brick by detecting the edges of the drick. A well-known algorithm is the Canny edge detector[18] developed by John F. Canny in 1986. Unfortunately, after some tests, we quickly faced some problems: the drick and the ground are made of concrete, with very little discernible differences which makes it harder to properly detect the edges of the drick, as shown in Figure 4.2. Moreover, the shadow of the drick can have an impact on the outline detected by the edge detector. On top of that, if the camera is tilted, the shape of the drick won't be recognizable as with a camera that is directly perpendicular to the ground.

4.3.4 Color points on the brick

In the same idea of the features detection in the Subsection 4.3.1, salient points can be added on the drick at fixed and known positions. One way to achieve that is adding color points on the object. These points can be chosen to be easily detected and with interesting positions for the purpose of facilitating the positioning step.

This solution, although it requires color spots on the brick, was investigated and used during the project. It will be described more carefully in the Section 4.4

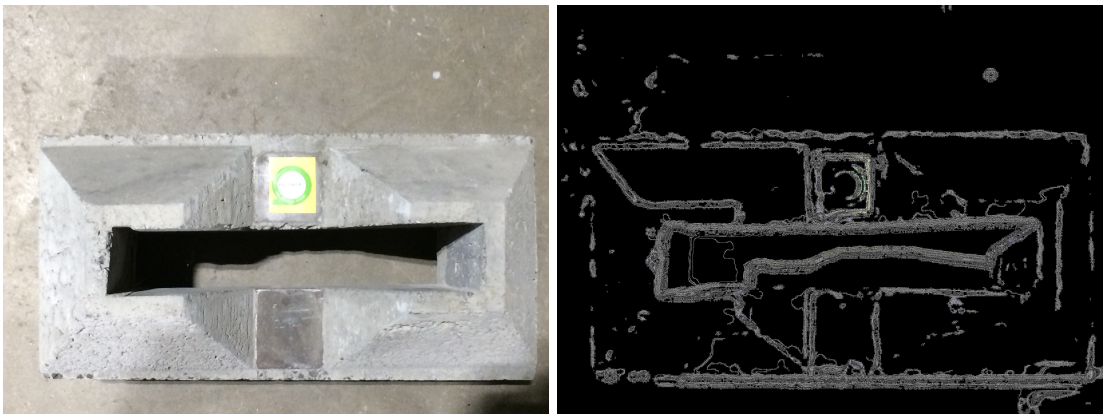


Figure 4.2: Edge detection with drick.

4.3.5 Code on the brick

Instead of detecting directly the bricks on the image, it can be envisioned to detect distinctive patterns on the brick at a fixed position.

The reliability of the solution is one advantage. Indeed, robust patterns with error correction principle can be used to facilitate the detection. Moreover, these patterns can embed information about the dricks. It is an advantage if we determine which dricks must be taken in an exact order but the information can also allow the drone to identify different dricks such as these forming a corner. Furthermore, with some patterns, the orientation can be easily deduced.

QR code

The most well-known code easily readable by a machine is the QR code (Quick Response Code). It is a two-dimensional barcode designed initially for the automobile industry in Japan to track components during manufacturing, it has since been standardized in ISO18004[23].

Figure 4.3: Example QR code containing the address of the UCL website

Using a popular solution is an advantage because it already exists robust libraries for example, in C++, ZBar[12]. However, in practice, we noticed a poor performance at a distance higher than 70 cm. Indeed, as you can observe on the Figure 4.3, the QR code is composed by small squares black or white, the distance and the blurring impact the algorithm because it is difficult to distinguish the different squares.

AR marker

Solutions similar to QR code exists for augmented reality (AR) purpose. Indeed, the pose estimation of these markers can be easily computed and augmented reality can be developed thanks to that. You can observe a simple example on the Figure 4.4.

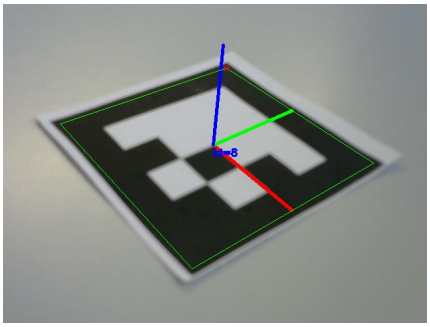


Figure 4.4: Simple example of augmented reality based on ArUco marker.

Moreover, these markers are often used in the robotics industry for the pose estimation such as by Boston Dynamics as you can observe on the Figure 4.5.

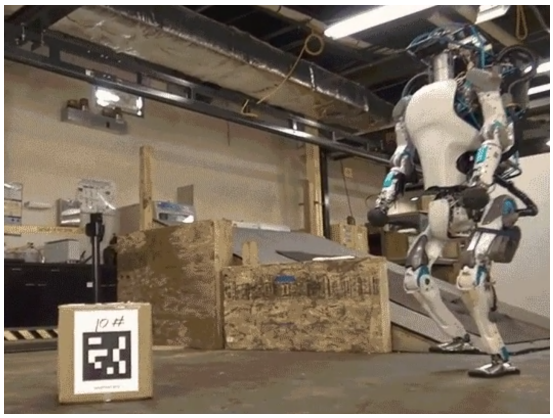


Figure 4.5: Robot of Boston Dynamics[2] using AR marker.

We have chosen this solution because the performances were positive during the first tests. Indeed, the squares are bigger than for the QR code and so more robust to the distance and the blurring. In practice, the chosen AR marker is the ArUco marker[22] because its usage is generalized and a library for the detection and decoding is developed in OpenCV. A more detailed explanation of the principles will be talked in a dedicated section.

4.4 Color points

In this section, we discuss the solution using color points on the drick. The detection of the drick can be divided into different tasks:

- Pre-processing of the image
- Color detection
- Morphological operation
- Blob detection

- Shape matching

4.4.1 Pre-processing

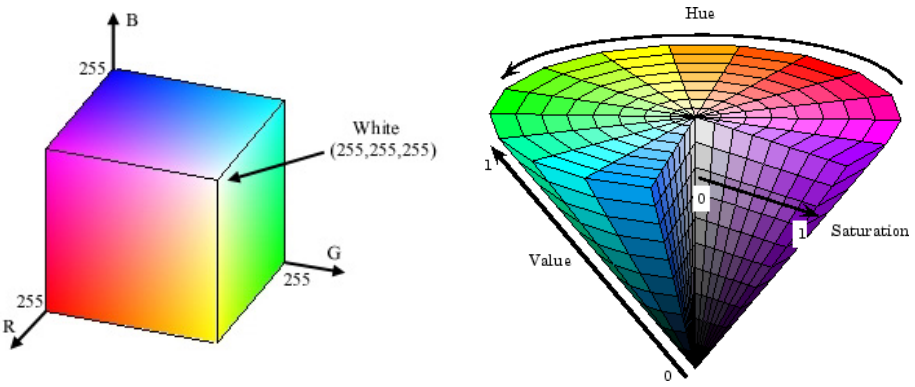


Figure 4.6: The RGB and HSV model[5][7]

The first operation done is to convert the image from RGB (`red`, `green`, `blue`) values to HSV (`hue`, `saturation` and `value`) values. It is more efficient to work in this model since the `Hue` value, defined by the CIECAM02 as the "degree to which a stimulus can be described as similar to a color described as red, green, blue and yellow"[33], roughly corresponds to the color of the pixel and is thus the most important one when trying to detect colors in an image. Red pixels, for example, typically falls in the 170°-10° range (`hue` values wrap around 180°) while blue pixels falls in the 100°-120° range. The `saturation` and `value` both roughly correlates with the environment in which the image is taken. Figure 4.6 shows the role of the different parameters on the final color, for the HSV and RGB models.

The algorithm then makes some modifications to the image to increase the efficiency of the detection in the later steps. One of the main goals is to make the algorithm independent of the environment lighting by maximizing the contrast of the image. For this, we apply CLAHE[39] to equalize the `lightness` values over the image.

CLAHE is a technique that relies on Histogram Equalization. Histogram Equalization is a technique that modifies the lightness values of the pixels of an image to increase the contrast. It tries to linearize the cumulative distribution function of the lightness values over the image, thus ensuring that the full range of lightness values (which are defined between 0 and 255) are used in the image, instead of possibly only a subset of these values.

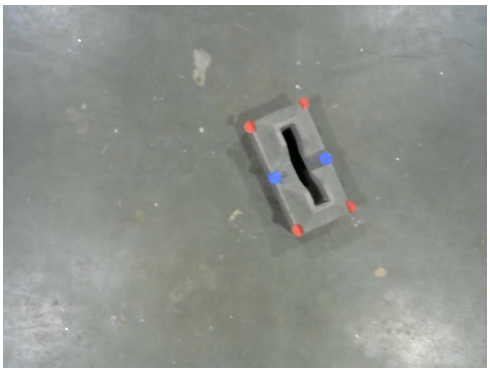
As multiple parts of the image can have different lighting conditions, CLAHE first divides the image into tiles and then applies Histogram Equalization on each tile. In our solution, we apply CLAHE on the `value`

This results in an image where all values of `value` are used, thus allowing the algorithm to be less dependent on the general lightness environment of the UAV.

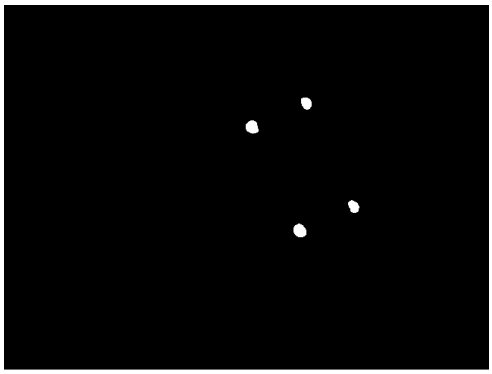
4.4.2 Color detection

Thresholding

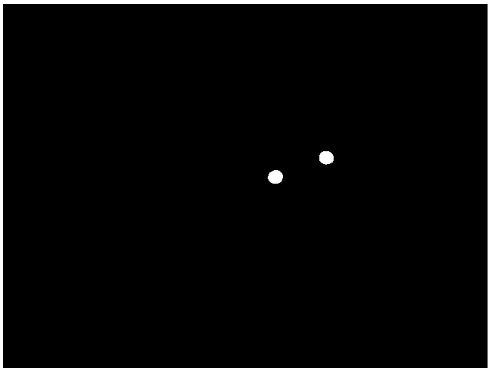
The algorithm first thresholds the image by selecting the pixels that belong in a certain interval of HSV values. The interval is chosen empirically, based on the color put on the brick. Typical intervals used for our application is given in Table 4.1. Note that the value a determine principally by the `hue` value and it shows the advantage to using the HSV model again the RGB model.



(a) Original image



(b) Thresholding the red pixels



(c) Thresholding the blue pixels

Figure 4.7: Example of thresholding

	Red pixels		Blue pixels	
	Minimum	Maximum	Minimum	Maximum
Hue	170	10	80	115
Saturation	80	255	85	255
Value	70	255	75	255

Table 4.1: Example of an interval to detect red and blue pixels in *normal* lighting conditions

The output of this step is a black-and-white image where white pixels correspond to detected color pixels.

This step is done for each color that the algorithm needs to detect and can be parallelized. The programs allow for any pattern of colors as long as it forms 2 lines of any number of points of any different colors. In this thesis, we use 2 lines of 3 points (a red one, a blue one and a red one). Figure 4.7 shows the output of this step on an image taken by a drone with colors spots on the brick.

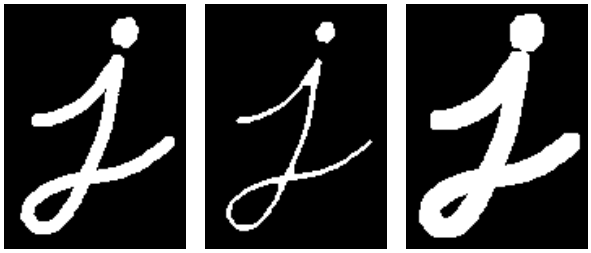
Morphological operations

To remove the white noise (isolated white pixels outside of a white spot) and black noise (black pixels inside a white spot), we apply a morphological opening and closing, which relies on morphological dilation and erosion.

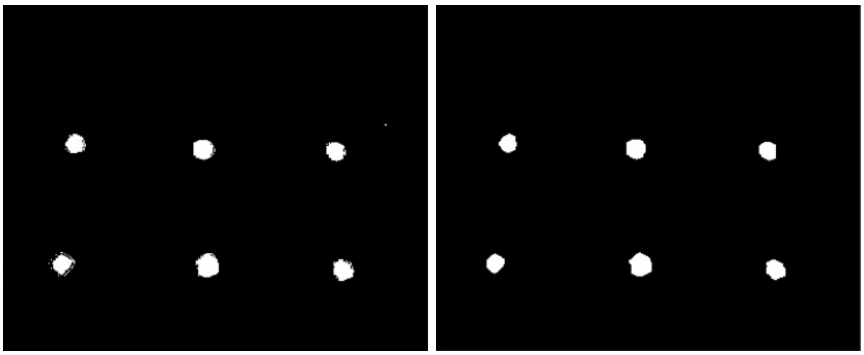
A morphological dilation is the process of applying a structuring element of white pixels, usually a square or a circle, onto every white pixel on the image to "grow" the image out of the white

pixels boundaries, as shown in Figure 4.8c. Erosion is the complement of dilation. It removes a structuring element of black pixels for every black pixel in the image, as shown in Figure 4.8b.

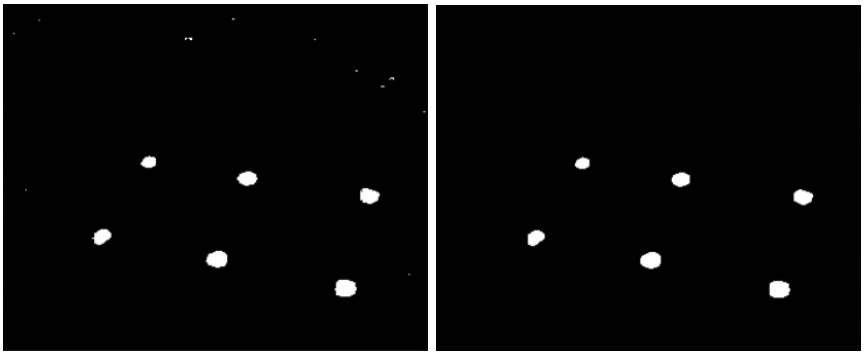
A morphological opening corresponds to an erosion followed by a dilation (both with the same structuring element) while a morphological closing corresponds to a dilation followed by an erosion (again, with the same structuring element). Figure 4.8e shows how the opening is used to remove isolated white pixels from the image, reducing the risk of misdetection in the following steps of the algorithm. Conversely, closing will remove black pixels inside patches of white pixels, as shown in Figure 4.8d.



(a) Original[16] (b) Erosion[16] (c) Dilation[16]



(d) Morphological closing: note how the holes in the white blobs have disappeared after the closing



(e) Morphological opening: note how the isolated pixels outside the blobs have disappeared after the opening

Figure 4.8: Morphological operations

4.4.3 Blob detector

Once a cleaned black-and-white image is produced, we try to detect "blob" of pixels out of the white pixels of the image. To extract those blobs, the algorithm simply finds groups of pixels that are connected. Once a list of blobs (each composed of a group of pixels) is established, it can be filtered to rule out unwanted blobs, using multiple criteria:

- **Size** filtering simply rules out blobs that are composed of not enough or too many pixels.
- **Circularity** computes a measure of how the blob resembles a "circle". From that, the algorithm can rule out blobs that are not circular enough.
- **Concavity** measures how "empty" the blob is. The algorithm can rule out blobs that are too much sparse, that contains too many black pixels inside them.

The result of this step is a filtered list of blobs of white pixels.

4.4.4 Shape matching

Once we have a list of color points or blobs, we are interested in trying to know which color points belongs to which bricks. Indeed, if the image contains multiple bricks, we'll have to differentiate from which bricks the multiple color points come from.

Our solution first tries to find lines of aligned equidistant n points by taking all combinations of n points among the list of found color points and verifying that. To make sure they are aligned, we simply check if the angle between two consecutive points is the same (up to a defined threshold). To make sure they are equidistant, we check if the distance between two consecutive points is the same (again, up to a defined threshold), as described in Algorithm 1.

Algorithm 1 Line matching algorithm

```
1:  procedure LineMatching(points)                          ▷ Finds lines of point
2:      lines ← []
3:      for line ← combination of n points in l do
4:          accepted = True
5:          for p, q, r ← three consecutive points in line do
6:              if norm(p − q) − norm(q − r) > distance_threshold then
7:                  accepted = False
8:              if angle(p, q) − angle(q, r) > angle_threshold then
9:                  accepted = False
10:         if accepted then
11:             lines.append(l)
12:     return l
```

Algorithm 2 Shape matching algorithm

```
1:  procedure ShapeMatching(lines)                          ▷ Finds shapes outs of lines
2:      shapes ← []
3:      for line_1, line_2 ← any pair of two lines in lines do
4:          accepted = True
5:          if mean_distance(line_1) − mean_distance(line_2) > distance_threshold then
6:              accepted = False
7:          if mean_angle(line_1) − mean_angle(line_2) > angle_threshold then
8:              accepted = False
9:          if accepted then
10:             shapes.append(line_1, line_2)
11:     return shapes
```

Once we have a collection of lines, the algorithm tries to group pairs of lines together, to finally form the brick. In the same vein as the previous procedure, we try to match pairs of lines that are

approximately parallel and have approximately the same distance between points, as described in Algorithm 2.

4.5 ArUco codes

Figure 4.9: ArUco square markers used for detection, with a resolution of 5x5, 6x6 and 8x8 respectively[22]

ArUco is an algorithm developed by Garrido-Jurado, Muñoz-Salinas et al[22] that can detect and estimate the camera pose using squared markers such as those shown in Figure 4.9.

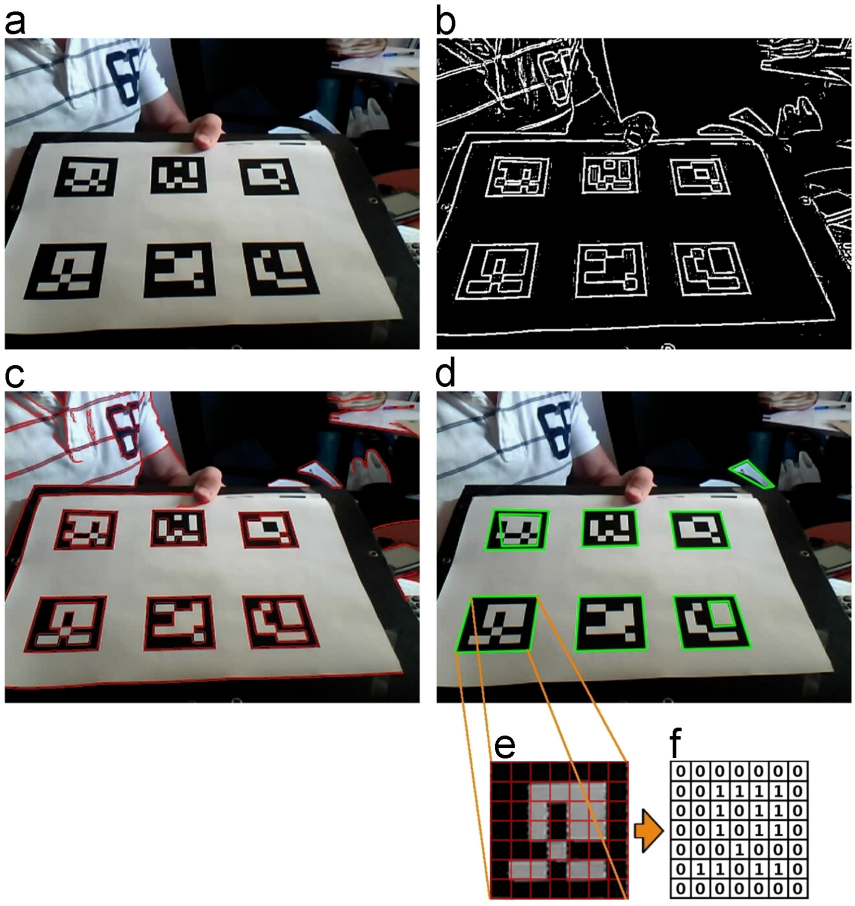


Figure 4.10: ArUco algorithm for automatic marker detection[22]

The algorithm works by extracting contours in the image using local adaptive thresholding (Figure 4.10(b)) and contour detection (Figure 4.10(c)), and then filtering out contours that are not 4-sided polygons (Figure 4.10(d)). For each polygon found, it uses perspective projection to turn the polygon into a square (Figure 4.10(e)), and then tries to read the data inside the marker by tiling the image and checking if the majority of the pixels in a tile is white or black

(Figure 4.10(f)). To filter out 4-sided polygons that weren't markers, the algorithm checks if the border tiles all contain 0 (that is, that they are filled with black pixels).

What results is a list of detected markers with the data the algorithm has parsed. The method handles any resolution of markers, as shown in Figure 4.9 which allow the developer to do a trade-off between data and robustness. Indeed, a higher resolution will contain more bits of data but will reduce the maximum distance of detection.

The article by Garrido-Jurado, Muñoz-Salinas et al[22] also details a method to generate a fault-resistant dictionary. Indeed, if a 5x5 marker is used (allowing 25 bits of information, and 2^{25} different values), but the application only needs to recognize four different values, the mapping between the values and the markers can be chosen in a way that maximizes the difference between the four different markers chosen to represent the four values. Such a process is equivalent to an error-correcting code used in other computer science domains.

4.5.1 Subpixel refinement of corners

Once ArUco found a squared marker and correctly parsed it, the algorithm can use the position of the bits (and especially the corners) inside the marker to correct a posteriori the initial position of the corners detected in the image. This process, called subpixel refinement, improves the precision of the corner, and ultimately the positioning parameters. Though, it introduces a computational cost to the overall detection.

4.6 False positives vs false negatives

Due to the strong constraints, the two methods enforce good performance. The detection step rarely recognizes markers that aren't true markers (false positives), and is actually more likely to reject a true marker if it isn't perfectly recognized as such (false negatives).

Indeed, the color markers detection algorithm only recognizes a set of points as a brick if they are aligned. The use of multiple colors alternately also reduces the chances of false positives. With the same objective in mind, ArUco has been designed to only recognize a square marker if it can correctly parse it.

This property allows the next step of the algorithm to be less concerned about whether a set of points is truly a brick, or if some of the points of the set are outliers.

CHAPTER 5

Positioning

5.1 Objectives

Once the detection phase yielded a list of detected bricks on the image, we are interested in trying to find the relative position of the bricks to the camera and indirectly to the drone.

For this purpose, we estimate the position and the rotation of a known 3D model given a set of n 3D points and their corresponding 2D projections on an image. Before any processing, we calibrate the camera in order to obtain a relation between the camera's units (pixels) and the physical word units (meters for example).

5.2 Calibration

5.2.1 Pinhole camera model

From a simple point of view, a camera saves an image in two dimensions from a three-dimensional scene.

The simplest model is the pinhole camera model, it consists of a pinhole in a wall where every point of the physical world are projected onto the image plane through the hole. As you can observe the Figure 5.1, the image is rotated of 180° on the image plane but we can imagine a virtual image plane located in front of the pinhole instead of behind.

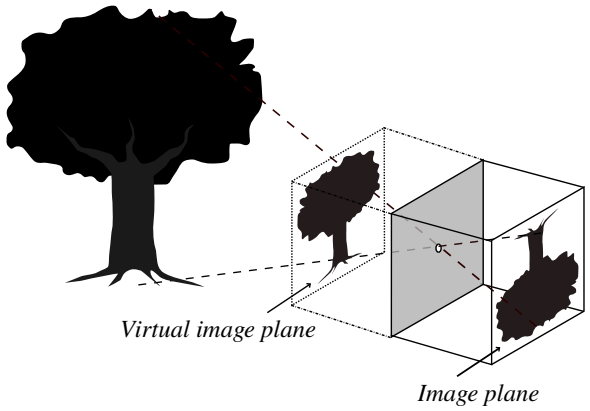


Figure 5.1: Representation of pinhole camera model

From the model, we can determine the parameters that determine the projection of the real world to the image plane: the intrinsic and extrinsic parameters.

5.2.2 Intrinsic camera parameters

The intrinsic camera parameters depend on the camera setup like the model and the resolution used. They allow the mapping between the camera units and the real world metrics. These are five intrinsic parameters:

- Vertical and horizontal focal length f_x and f_y are the distance between the pinhole and the image plane. In the pinhole camera model, the two values are the same but in practice due to the limit of the model they can be different.
- Principal point c_x and c_y is the intersection between the image plane and the axis perpendicular to the image plane that passes through the pinhole. It is usually the image center.
- Skew factor s corresponds to a shear distortion in the projected image.

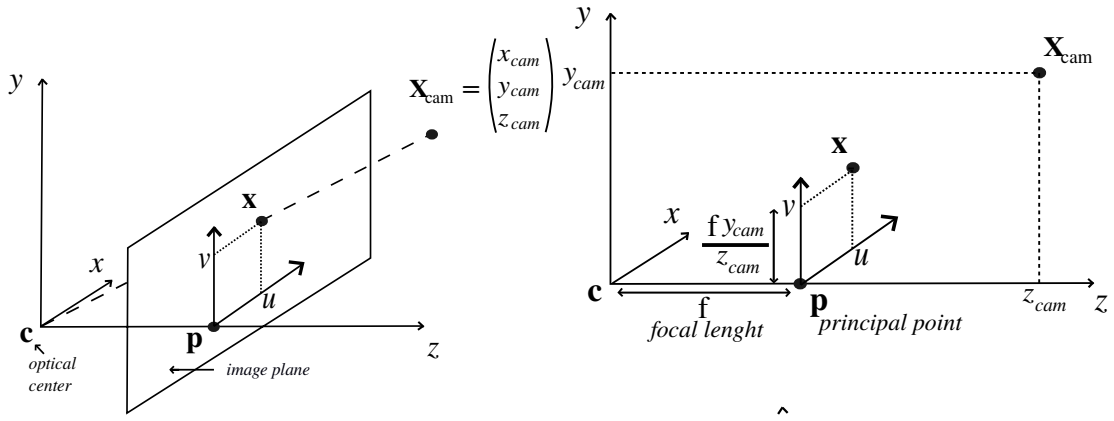


Figure 5.2: Representation of intrinsic parameters from the pinhole model

We can express the point of an image from the real world as:

$$z_{cam}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_x & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix}$$

Where the camera matrix K including every intrinsic parameters is:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_x & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

In practice, this matrix is estimated by a calibration process. Note that any point on the same ray of light will be projected to the same position on the image plane. Thus, every point in the same ray of light from $\mathbf{c}$ to $\mathbf{X}_{\text{cam}}$ will be expressed with the same image coordinates.

5.2.3 Extrinsic camera parameters

The extrinsic camera parameters are external to the camera and depend on the world configuration.

Indeed, the camera can only express the real world in a coordinate relative to itself. However, it can be interesting to express the measurement in another coordinate. For example, in our project, we want to express the measurement with respect to the UAV coordinate instead of the camera coordinate.

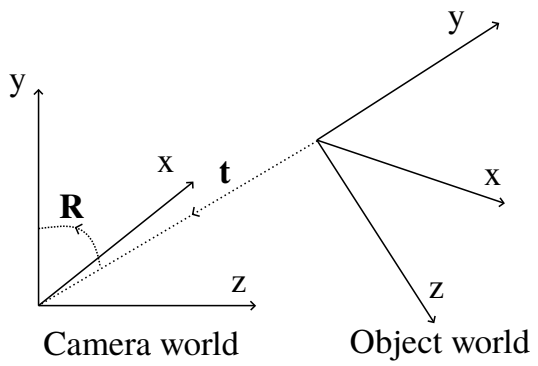


Figure 5.3: Representation of extrinsic parameters

In this purpose, the extrinsic parameters formulate the position and the orientation of the camera with respect to a world reference.

This transformation is defined by:

- 3x1 translation vector, t
- 3x3 rotation matrix, R

Knowing the camera's position and orientation in the coordinate system, we compute the transformation from one to another. Let us define the point in the world coordinate as $p_w = (x_w, y_w, z_w)^t f$ and the point in the camera coordinate as $p_c = (x_c, y_c, z_c)^t$, we obtain:

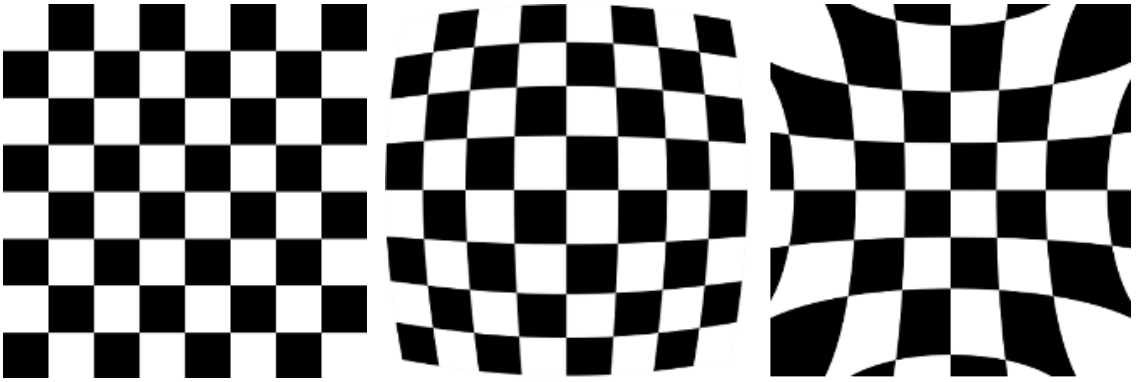
$$p_c = R * p_w + t$$

We can write it in a matrix format with homogeneous coordinates:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \Big[R|t\Big] * \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Finally, from the intrinsic and extrinsic camera parameters, we can express, thanks to the pinhole camera model, a real world point in the image plane:

$$z_{cam} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \Big[R|t\Big] * \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$



(a) No distortion (b) Positive radial distortion (Barrel distortion) (c) Negative radial distortion (Pincushion distortion)

Figure 5.4: The different types of distortions and their impacts camera[4]

5.2.4 Lens distortion

The pinhole camera model approximates correctly the situation but the model can be limited in practice by the distortion caused by the lens. Indeed, real camera use lenses and the lenses deflect the ray of light passing through them. This is why it is an approximation to consider that the rays of light are straight from the real world to the image plane through the pinhole.

These are two kinds of distortion: radial and tangential distortion. The radial distortion causes the straight line to appear curved, as shown in Figure 5.4. Moreover, this effect is intensified at the edge of the image.

The radial distortion is represented by OpenCV[10] thanks to the model of Brown[17] as followed:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

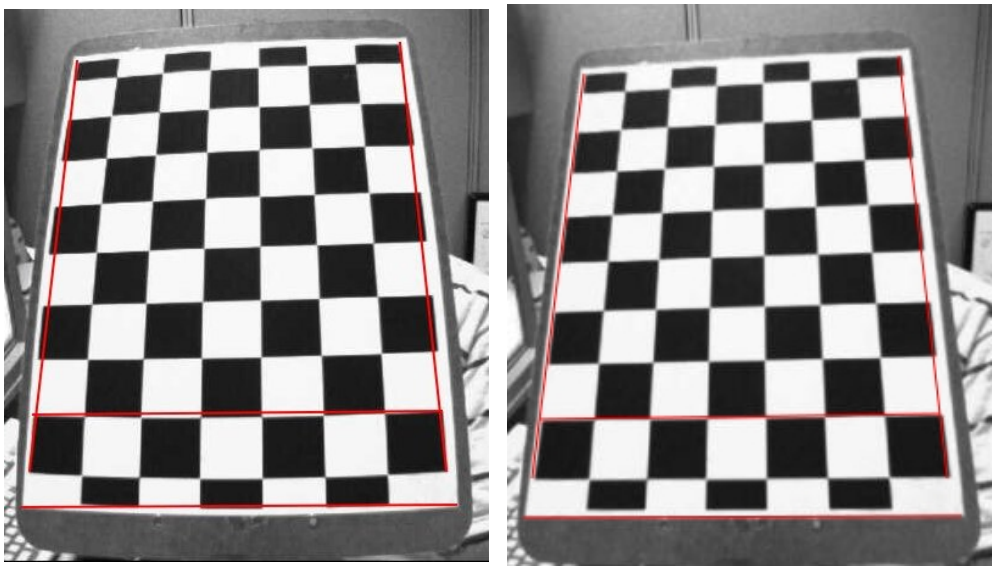
Concerning the tangential distortion, it can occur if the lens is not exactly parallel to the image plane and it can be represented as followed:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

To conclude for the lens distortion, the calibration process must estimate 5 different factors for the distortion.

5.2.5 Calibration in practice

In practice, the library OpenCV offers a calibration process of the camera to estimate the camera matrix and the distortion matrix by taking pictures from different points of view of a regular pattern such as a chessboard.



(a) Before (b) After

Figure 5.5: Representation of the effect of a calibration of a camera[3]

It is important to be very careful with the calibration because it will determine the relation between the camera's units and the physical world as explained before and therefore the accuracy of the positioning system. Indeed, we faced measurement errors due to a poor calibration. The main problem was to use images taken at the same distance. Moreover, the camera was not stable during the process, creating a degradation of the quality of the pictures and thus of the calibration of the camera.

To obtain a correct calibration, different steps must be respected:

- To be sure that the camera or the chessboard is not moving during the capture of the picture to avoid motion blurring.
- To take pictures changing the different parameters. For example, modifying the distance between the camera and the chessboard without changing the orientation or keeping the same distance but moving the camera/chessboard to be on a side of the image and not the center.

5.3 Positioning: Perspective-n-Point

As explained before, the problem at hand is to estimate the position and the rotation of a 3D model given a set of n 3D points and their corresponding 2D projections on an image, as shown in Figure 5.6. For example, if we used the color detection strategy in the previous step, we'd use the 2D positions of the detected color markers and their corresponding position on a 3D model of the brick. The algorithm would then give us the position and the orientation of such a 3D model such that the projection of the position of the color markers on the camera plane corresponds to the given 2D points.

With the ArUco codes, the four corners of the squared marker are used as 2D points.

This problem, known as the Perspective-n-Point problem, has been heavily studied and multiple algorithms exist in the literature to solve it. This section details the basis of the most used methods with their advantages and disadvantages.

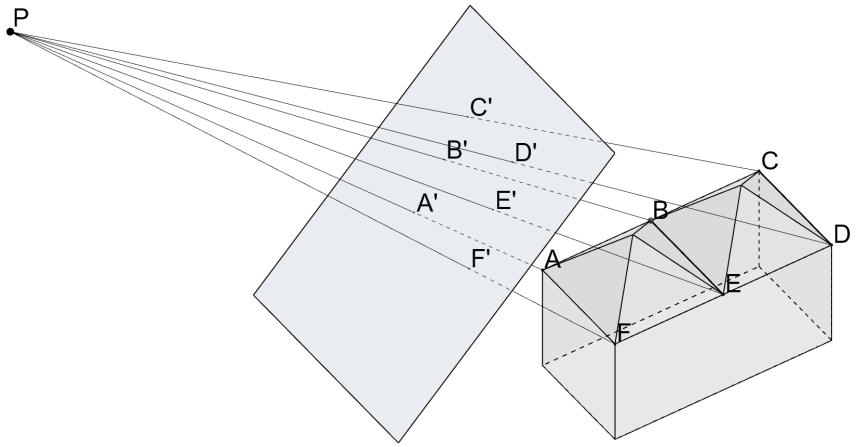


Figure 5.6: Representation of the PnP problem. The blue plane represents the camera plane while P represents the focal centre of the camera. Points of the 3D model (A to B) are projected to the camera plane (A' to F')

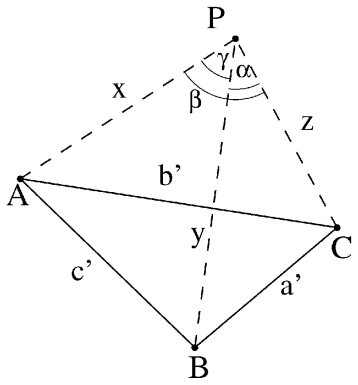


Figure 5.7: Representation of the Perspective-3-Point[21]

5.3.1 Perspective-3-Point

To find the six degrees of freedom (three for the translation and three for the rotation), a minimum of three points are needed. The problem, in this particular setup, is called the Perspective-3-points and an algorithm to solve it is detailed in the article by Gao et al.[21]

Given three known 2D points (A, B and C) and the center of our pinhole camera, we can define three triangles, all binding P and two other 2D points, as represented in 5.7. From these triangles, we define the distances $|AB| = c'$, $|BC| = a'$, $|CA| = b'$, $|PA| = X$, $|PB| = Y$, $|PC| = Z$ and the angles $\widehat{APB} = \alpha$, $\widehat{BPC} = \beta$, $\widehat{CPA} = \gamma$. The cosine law can then be used to obtain the following system of equations:

$$
\begin{aligned}
X^2 + Y^2 - XY \cdot 2cos(\alpha) - c'^2 = 0 \\
Y^2 + Z^2 - YZ \cdot 2cos(\beta) - a'^2 = 0 \\
Z^2 + X^2 - ZX \cdot 2cos(\gamma) - b'^2 = 0
\end{aligned}
$$

In those equations, X, Y and Z are the unknowns. Resolving these equations yield many solutions due to the geometrical nature of the problem. In practice, a fourth point is used to choose between them.

The main disadvantage of this problem is that we can't use additional points to obtain a better estimation of the solution. Since we use six color marks on a brick, it means we would need to ignore three detected points on our image. Algorithms capable of taking into account any number of points exist, as described below.

5.3.2 Efficient Perspective-n-Point (EPnP)

EPnP is an algorithm described in the article by Lepetit, Moreno-Noguer and Fua[28].

This solution has multiple advantages, two of which are crucial for our application. Firstly, the algorithm can take any number of projected 2D points, which allows us to get a better precision by increasing the number of color marks on the brick. Secondly, the algorithm has a linear time complexity with the number of points given. This allows us to increase the number of color marks on the brick without increasing too much the time taken to scan an image.

It works by expressing the n points as a weighted sum of four controls points and solving the PnP problem where the unknowns become the coordinates of these controls points.

5.3.3 Random sample consensus (RANSAC)

The presence of outliers in our data can worsen the quality of the result produced by EPnP. [20] Fischler and Bolles devised a meta-algorithm, RANSAC[20], that iteratively generates multiple solutions of EPnP using different subsets of the original n points, each time trying to reject the noisy samples of the dataset that deviates too much from the fitted model.

Due to the strong constraints, our two detection strategies enforces in the previous step, we have fewer chances of having outliers in the set of 2D points. Thus, RANSAC is of little use in our case and hasn't been implemented in our solution.

5.3.4 Axes system

As said before, the output of the positioning is a translation vector and a rotation vector. Care must be taken when dealing with the rotation vector as multiple different formalisms exist to express a rotation. OpenCV uses by default an axis-angle formalism, which is explained in Figure 5.8. In this notation, the direction of the vector represents the axis around which the rotation will be applied. The norm of the vector is then the angle θ of the said rotation.

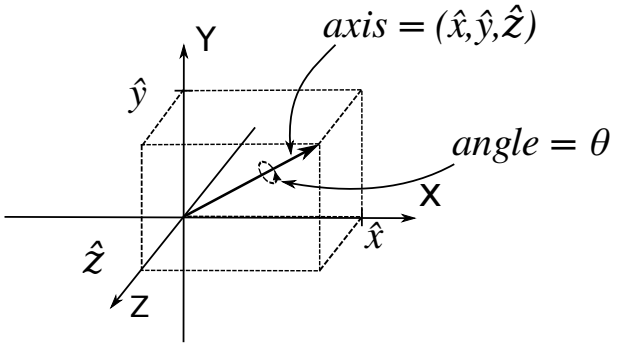
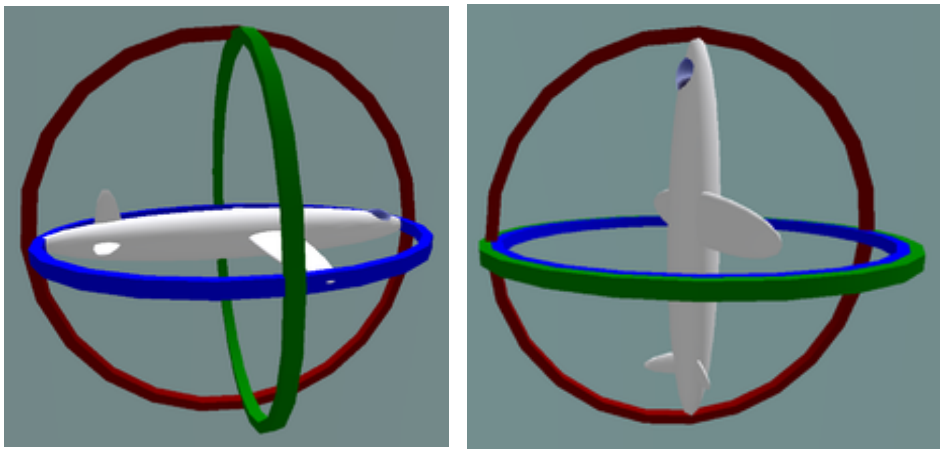


Figure 5.8: Representation of the axis-angle formalism[38]

We can convert those angles to Euler angles to get a more intuitive representation of the rotation when debugging. Euler angles represent a succession of three rotations along the three axes. This

formalism suffers from some problems, the main one being the Gimbal Lock effect which is why it less usable in mathematical computations.

The Gimbal Lock effect is the loss of a degree of freedom in specific configurations of the angles, most particularly when one axis has a rotation of 90°. Figure 5.9 shows an illustration of this effect. In Figure 5.9a, the three axes can be used to induce a rotation of the plane along the three axes, but in Figure 5.9b shows that a rotation of the plane at 90° aligned two axes, thus removing one degree of freedom in the system.



(a) Normal situation (b) Gimbal locked situation

Figure 5.9: Illustration of the Gimbal Lock effect[6]

Euler angles can still be used to get a quick interpretation of the result of the algorithm (such as when debugging it or validating the results, for example).

CHAPTER 6

Evaluation

This chapter focuses on the evaluation of the performances of the different algorithms used for the final solution.

All the videos used for this section are available in the folder `detector/videos` of the archive of the source code.

6.1 Positioning

For the positioning part, we are interested in the different metrics that are outputted by the positioning algorithm, that is, the translation vector (three components) and the rotation vector (three components).

We evaluate these six values independently taking a short video (~30 seconds) for each possible distances and rotations between the camera and the markers. We then applied our algorithm to this video, saved the result, and represented them in the box plots below. Both solutions (Color Markers and ArUco) were tested at the same time.

The placement of the axis is shown in Figure 6.1. The z-axis is the axis perpendicular to the markers, and going through the camera. The x-axis and the y-axis are the axes in the plane of the board holding the markers.

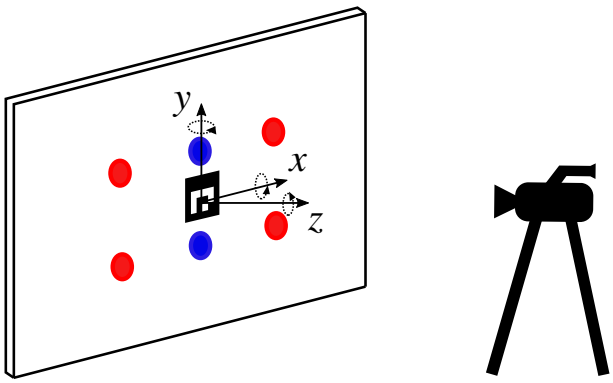


Figure 6.1: Representation of method of evaluation

The camera used for the measurement is the Logitech HD Webcam C270 presented in Section

3.3.2. Due to the field-of-view of the webcam, we weren't able to measure with a distance between the camera and the camera lower than 30 cm as the markers were no longer visible at this distance.

Since we were only interested in the measurement for this part, we only took into account the frames where the algorithm did detect a brick and removed those where nothing was detected.

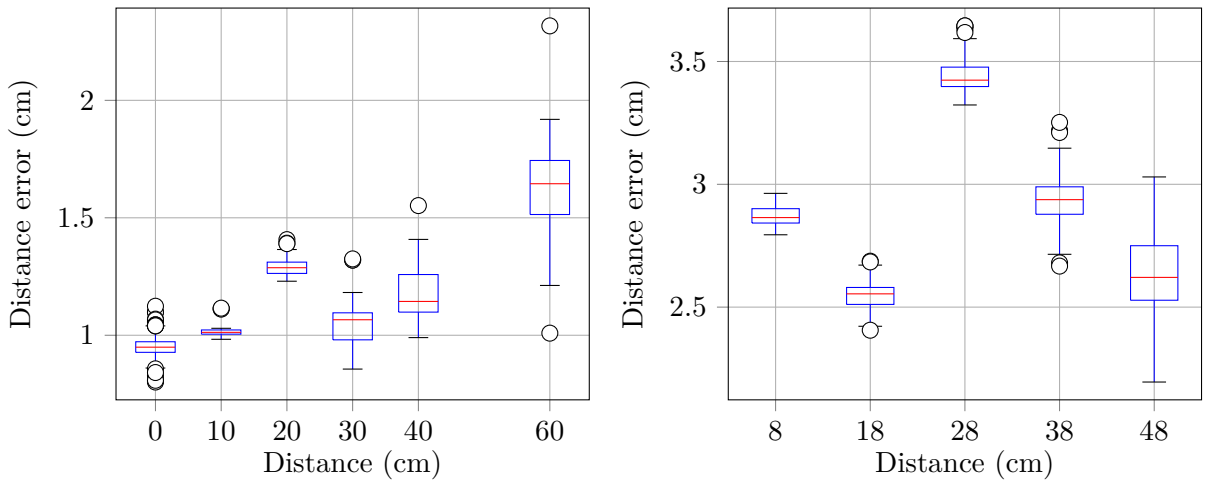
We measured and reported the three rotation parameters, but in our situation, only the rotation along the z-axis is important. Indeed if the camera is directly perpendicular to the ground, and since the drone is always approximately parallel to the ground, there should never be a rotation along the x-axis or the y-axis (or at least, not a big one). The only rotation that interests us in this situation is the one along the z-axis, as it corresponds to the alignment of the brick relative to the drone.

6.1.1 Color detection

We first focus on the results of the color detection algorithm. We used 6 color markers (2-by-3) with a distance of 10 cm between the markers.

Translation

Figure 6.2a (Figure 6.2b) shows the measured error for a translation along the x-axis (resp. y-axis). These measurements were taken with a camera placed 1.90 m away from the marker. We can see that the errors along those two axes are contained below 4 cm and that the variance, in general, increases as we move the marker away from the center of the image. This error in the translation parameters can be caused by an error in the algorithm, but also by an error during the calibration process of the camera. Having a good camera matrix and a distortion vector is important for the positioning phase of the algorithm, as these will have a direct impact on the result of the solvePnP algorithm.



(a) X-axis

(b) Y-axis

Figure 6.2: Translation error along the camera-plane axis using the color detection algorithm

Figure 6.3 shows the error (in percentage) for a translation along the z-axis. We can see that the margin of error is generally around 7%, and always below 10%. We can notice that the variance also increases as we move further away from the marker, which is explained by the difficulty the

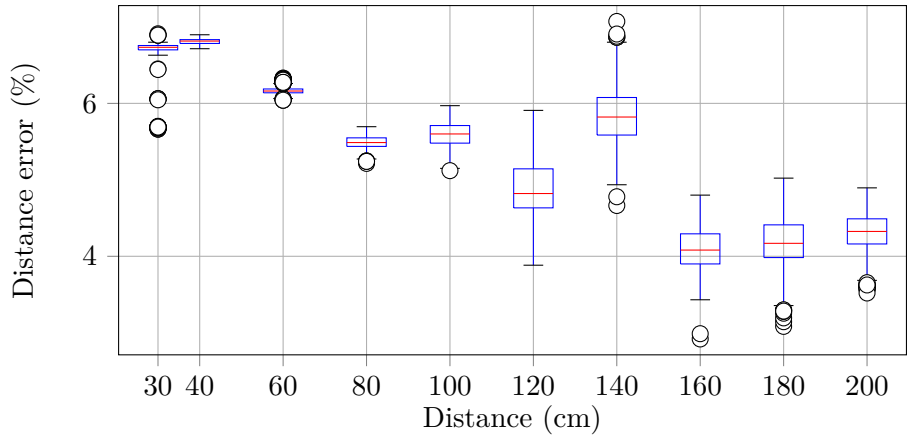


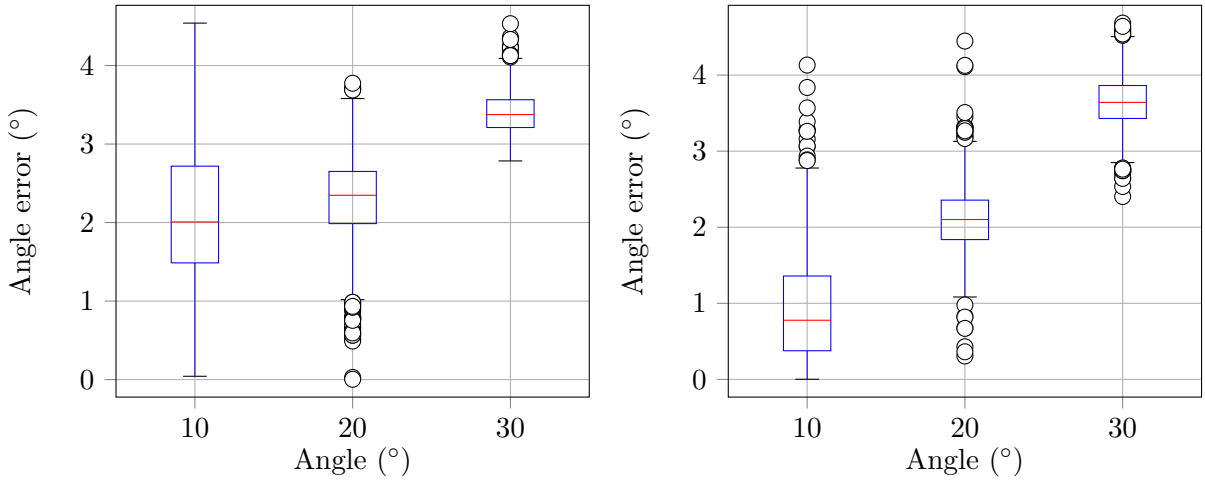
Figure 6.3: Translation error (in percentage) along the z-axis using the color detection algorithm

algorithm will have to clearly distinguish the color markers. It is important to notice that the error in centimetres will decrease when the drone approach the object. Moreover, under 1 meter of distance, the constraint of 5 cm is respected.

Rotation

To measure the detected rotation, the camera was placed 1m away from the marker.

Figure 6.4a (Figure 6.4a) shows the measured error for a rotation along the x-axis (resp. y-axis). We can see that the margin of error is always below 5°.



(a) X-axis

(b) Y-axis

Figure 6.4: Rotation error along the camera-plane axis using the color detection algorithm

Figure 6.5 shows the measured error along the z-axis. Our solution achieves an error margin consistently below 3°. This means that if the drone tries to align itself with the brick using the result of the computer vision algorithm, it will deviate with an error of 3°. This deviation will directly produce a displacement of the notch that should fit together with the small pyramid of the brick. A deviation of 3° for a brick of length 20 cm will approximately produce a displacement of $tan(3°) * 20cm = 1cm$. This displacement will add itself to the translation error along the x-axis or the y-axis, and the total should be inside the 5 cm constraint.

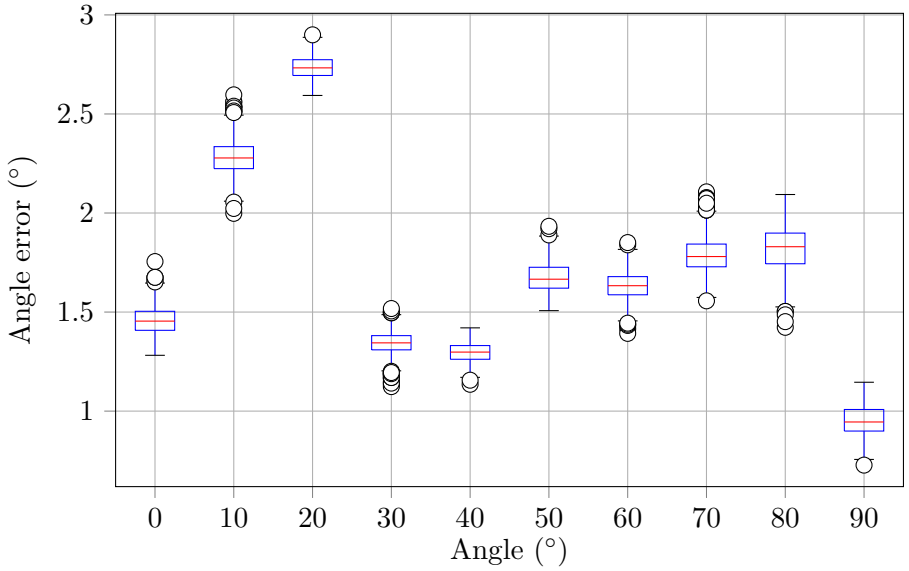


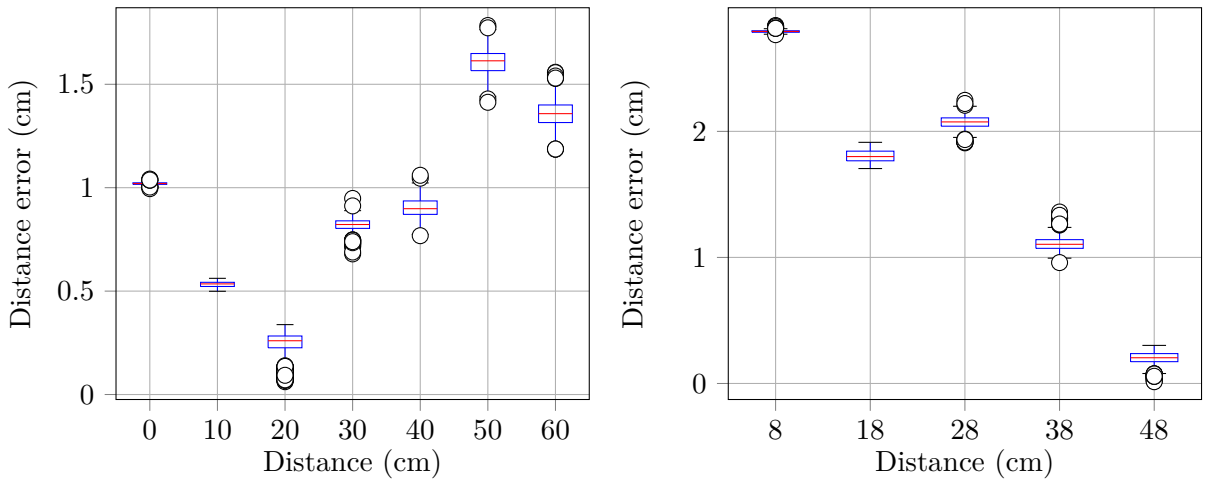
Figure 6.5: Rotation error along the z-axis using the color detection algorithm

6.1.2 ArUco positioning

We used a 6.5cm-by-6.5cm ArUco marker with a resolution of 4x4.

Translation

Figure 6.6a (Figure 6.6b) shows the measured error along the x-axis (resp. y-axis). The measurements were taken with a camera placed 1m90 away from the marker. We can see that the margin of error is below 3 cm, which fits the 5 cm constraint of the problem.



(a) Translation X

(b) Translation Y

Figure 6.6: Translation error along the camera-plane axis using the ArUco algorithm

Figure 6.7 shows the error (in percentage) measured along the z-axis. We can see that the margin of error goes up to 9% when the camera is around 2m away from the marker, but that it decreases as we move the camera closer and closer. The error, with a distance of 20 cm, is

reduced to 6%, which represents an error of about 1 cm.

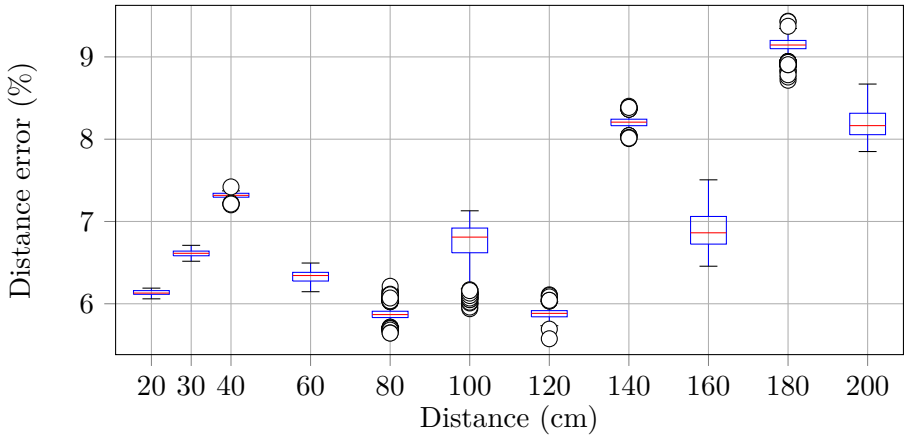


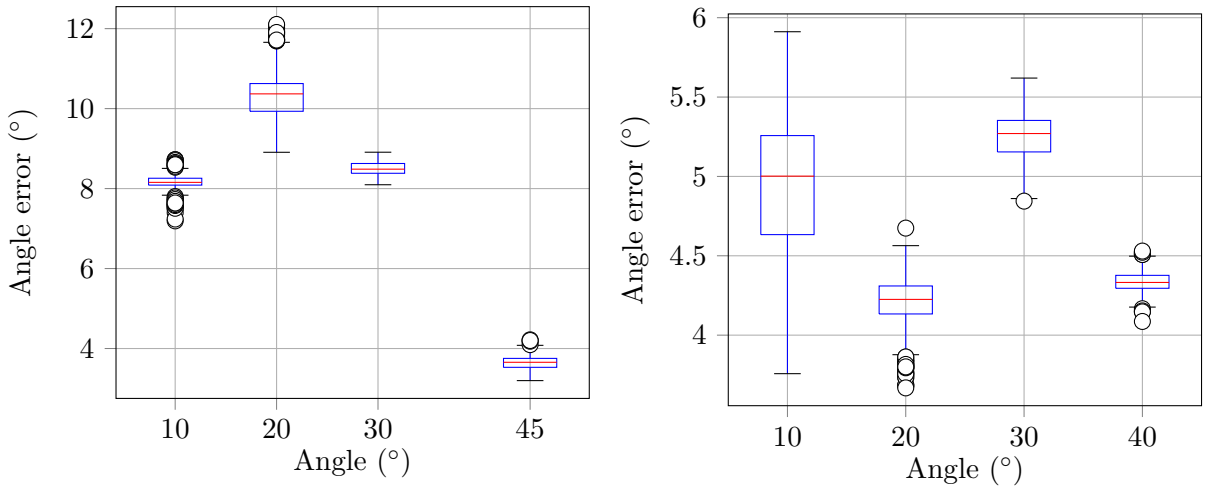
Figure 6.7: Translation error (in percentage) along the z-axis using the ArUco algorithm

Rotation

As with the color detection algorithm, the angle was measured with a camera placed 1m away from the target.

Figure 6.8a (Figure 6.8b) shows the measured error along the x-axis (resp. y-axis). The error in the measurement is quite noticeable, with differences going up to 10%.

This can be explained by the coplanar nature of the points used in the solve PnP, that is, the fact that all points are constrained in the same plane. This has a big impact on the computation of the rotation along the planar axis. A small misdetection of the distances between the points will make the algorithm believe that the object is tilted along the x-axis or the y-axis.



(a) Rotation X

(b) Rotation Y

Figure 6.8: Rotation error along the camera-plane axis using the ArUco algorithm

Figure 6.9 shows the measured error for a rotation along the z-axis. The error is around 2° for all the tested angles. As explained earlier, this error will induce a translation error due to the misplacement of the notch that should fit on the pyramid of the brick. The induced error will

here be less than 1 cm which, when added to the translation error detailed above, is still below 5 cm.

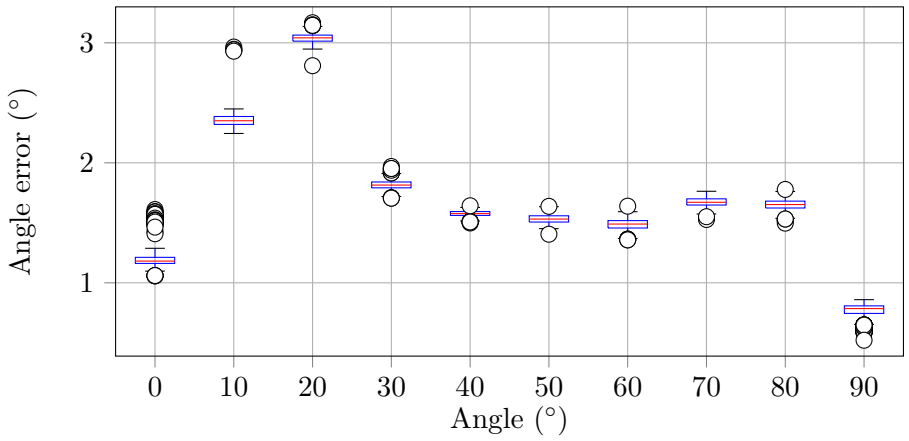


Figure 6.9: Rotation error along the z-axis using the ArUco algorithm

6.2 Frame rate

The time to process a frame has a big impact on the usefulness of our solution. Indeed, the drone needs to have access to the positioning information as fast as possible if it needs to correct its trajectory while in flight. The following measurements were done on the Odroid, which specifications are described 3.3.2.

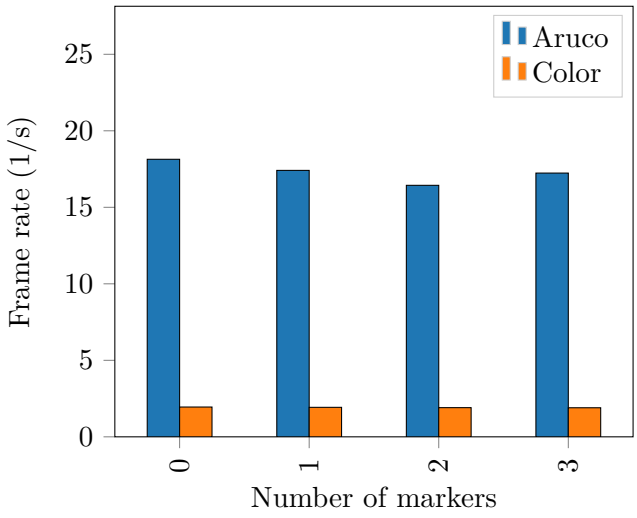


Figure 6.10: Frame rates of the two solutions depending on the number of markers

Figure 6.10 shows the results of a measurement. The algorithm analyzed a video of about 10 seconds. The framerate is calculated by taking the average time to process a frame, only on the frames where the correct amount of markers were found.

We can first see that the ArUco algorithm achieves a significantly better frame rate than the Color Detection technique.

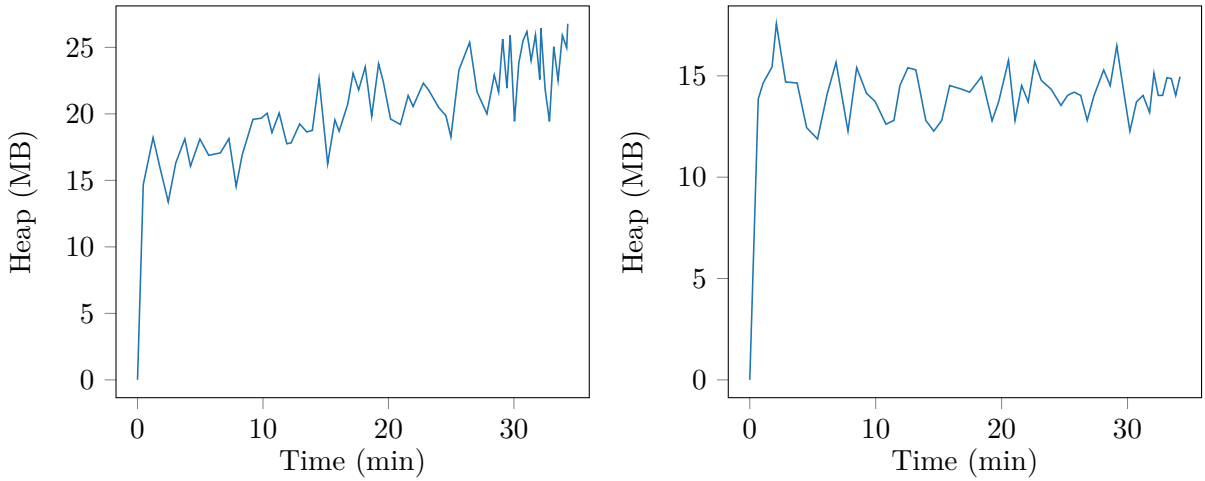
We can also notice that the drop in framerate as we add more and more markers is not as big as we may have thought. This effect is explained by the fact that most of the time spent on

processing a frame is done on the first phases: thresholding and blob detection for the Color Detection algorithm, and doing edge detection for ArUco. The workload needed to do these steps are independent of the number of markers.

6.3 Memory consumption

We used Valgrind to measure the memory consumption of the two algorithms, and more specifically Memory Check to spot the memory leaks, and Massif to profile the memory used over time.

Figure 6.11 has been generated by profiling two programs (one for each solution) that continuously scan the same video of 20 seconds over more than 30 minutes (looping back when the video finishes).



(a) Color detection

(b) ArUco

Figure 6.11: Memory consumption of the two detection algorithms

We can see that the memory consumption of ArUco is constant, but the increase seen for the Color Detection indicates that there is a small memory leak present in this algorithm. Unfortunately, even after extended analysis of the output of Valgrind, we weren't able to fully remove the memory leaks present in the algorithm.

Though, after 30 minutes of execution, the leak accumulated only around 10MB of memory. To fully saturate the 1GB of RAM available on the Odroid, it would take around 50 hours of execution. However, it is not a huge problem due to the fact that the flights don't attain this duration.

6.4 Comparison

For the positioning, the main goal of the solution is to have a precision of 5 cm along the x-axis and the y-axis so that the drone can take a brick. We can see that both the color detection algorithm and the ArUco algorithm fits this requirement, as both have a precision of 3 cm along those two axes.

The second goal was to have a relatively precise measure of the rotation along the z-axis. Indeed, the computer vision component of the drone is the only one being able to detect the rotation of the brick. Both algorithms yield a measurement with a margin of error below 3°.

The ArUco algorithm yields a slightly more precise measure for the translation along the three axes. This is due to the refined corner detection ArUco is able to do once it detected a code (as explained in Section 4.5.1). Later in this report, in Section 8.4.2, we discuss the possibility of putting two ArUco codes on the brick to solve a specific issue. Using two ArUco code would also improve the precision of the algorithm, by how much specifically could be the subject of further research.

The color detection algorithm, though, yields better measurements for the rotation along the two camera-plane axes. This is explained by the fact that the size of the ArUco marker is only 6.5cm-by-6.5cm, while the distance between two color points is 10 cm, for a total rectangle of 20cm-by-10cm. This bigger scale helps the positioning algorithm correct the possible error in the distance between the points, and prevent this error from impacting too much the detected rotation.

A possible improvement could be to combine the measurement of the two methods to obtain a more precise measure of the different parameters.

For the frame rate, we can see a clear advantage for the ArUco algorithm. Research could be done to profile and optimize the algorithm in order to get a better frame rate. The use of a GPU, a component more tailored for this kind of computation, could also be considered and explored.

With ArUco, we also have the advantage of being able to bundle an id to the marker that can be detected by the algorithm. This allows us to mark differently different kind of bricks (if a different brick is needed for the angle of a wall, for example), or to better handle the case where multiple bricks are seen on the image. Moreover, it can be more robust when we know which marker we want to detect by identifying the id.

Both methods allow for different types of object to be marked. For the color detection, a rectangle of markers must be placed on the object (With any number of markers of any color). For ArUco, a single marker must be placed at the middle of the object. This means that both algorithms can easily be modified to adapt to a new kind of brick if the design of the building blocks came to change in the future.

CHAPTER 7

Control System

In this chapter, we introduce the architecture of the system for the drone with the different position system integrated into it. Moreover, we will briefly explain the framework provided by the company ALX Systems, how it helps us control the drone and how we adapted our work into it.

7.1 General architecture

As described in Chapter 3 about the overview of the project, the system can be split into three different blocks :

- **Flight controller hardware**, handled by the PixHawk autopilot, it manages the motor controllers in function of the input received by the remote control or by the flight controller software.
- **Flight controller software**, handled by the system of the company ALX Systems, it gives inputs to the PixHawk in function of other blocks. The system manages the aggregation of the different data available.
- **Positioning components**, each system of positioning will feed the flight controller software the readings they all individually collect. There are three positioning systems :
    - Global positioning with total station
    - Global positioning with UWB tag
    - Local positioning with image processing

The next sections will describe how the different systems interact with each other and how they are deployed in practice. However, they will not describe the flight controller hardware block as we didn't put any work into it during the project. Indeed, from our point of view, the interfacing with the control of the drone is done by the ALX Systems software. So, we didn't examine in-depth how works the PixHawk controller.

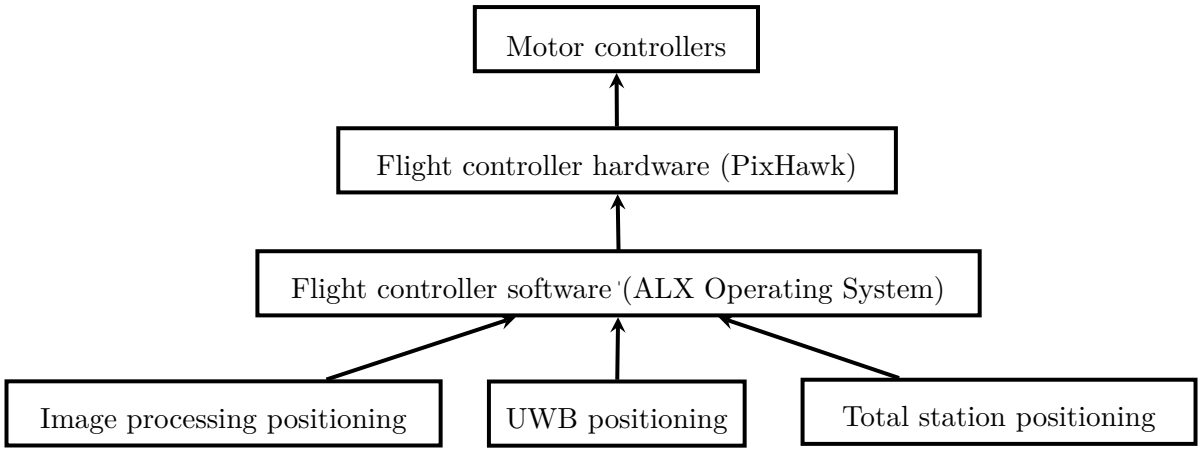


Figure 7.1: Big overview of the system

7.2 ALX Systems framework

This section will explain how the ALX Systems software works and the architecture often deployed by them during their projects. First, the architecture is divided into four distinct blocks as illustrated in the Figure 7.2.

- The **Ground station** is the user interface for the control of the drone. It doesn't need heavy computations. As such, a computer tablet is enough to host it. From this station, the user can manage the mission of the drones and give them different instructions. The information is passed through the ALX Server before going to the drone by internet protocol.
- The **ALX Server** is the connection between the ground station and the UAV. This block can be deployed locally on the ground station or remotely on a physical server. Indeed, the communication system will change. For example, the company can use radio connection between the ALX Server and the UAV but also wireless mobile telecommunications technology such as the 3G.
- The **Onboard ALX Operating System** is the core of the control embedded on the drone. It receives the instructions from the ALX Server, information from the flight controller and from the onboard computer vision. After processing them, it gives the right instruction to the flight controller depending on the objective it needs to accomplish. The instructions for the positioning can be relative or global because they are transformed to become GPS positions for the PixHawk.
- The **Onboard ALX Vision** is the image processing block. It will provide information about the environment of the drone to the ALX Operating System. This treatment can be done on the same physical device, or on another embedded device. The second solution is preferred by the company because using another one allows the second device to use optimized platform for the task such as a Graphics Processing Unit (GPU). Moreover, if a crash happens during the processing of the images, it will not cause problems for the whole ALX Operating System.

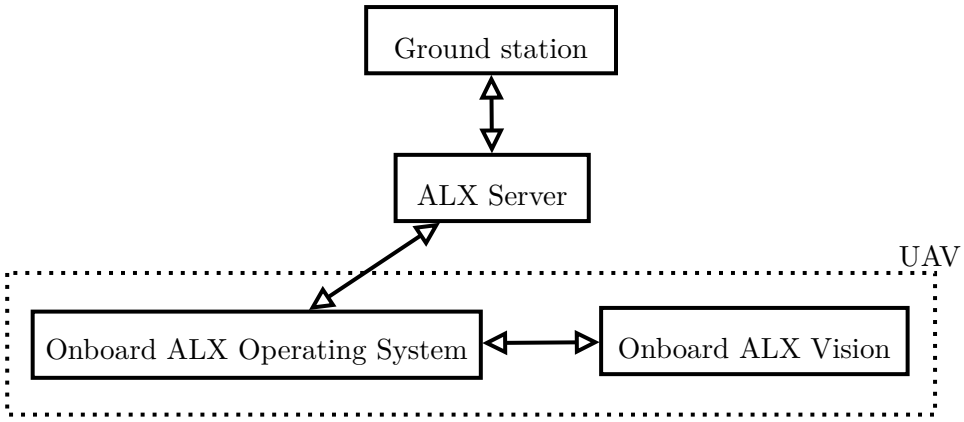


Figure 7.2: Resume of the basic architecture deployed by ALX Systems

7.3 Architecture of the solution

After explaining the general architecture used by ALX Systems, we describe the one used during the project, illustrated on the Figure 7.3.

Concerning the ground station and the ALX Server, they are deployed both on the same computer named ground computer station. This ground computer station is the user interface and it is connected to the total station and recuperates the positioning information providing from it thanks to the prism pose on the drone.

The positioning providing by the total station is sent by internet protocol through Wi-Fi router to the Odroid on the drone with the ALX control system. This one is connected too with the UWB tag that provides its position to the system. Furthermore, the control system receives the information about the detected objects from the computer vision working on a dedicated Odroid.

Finally, the different positioning readings are aggregated and provided as instructions to the PixHawk that manages the motors by controlling the electronic speed control.

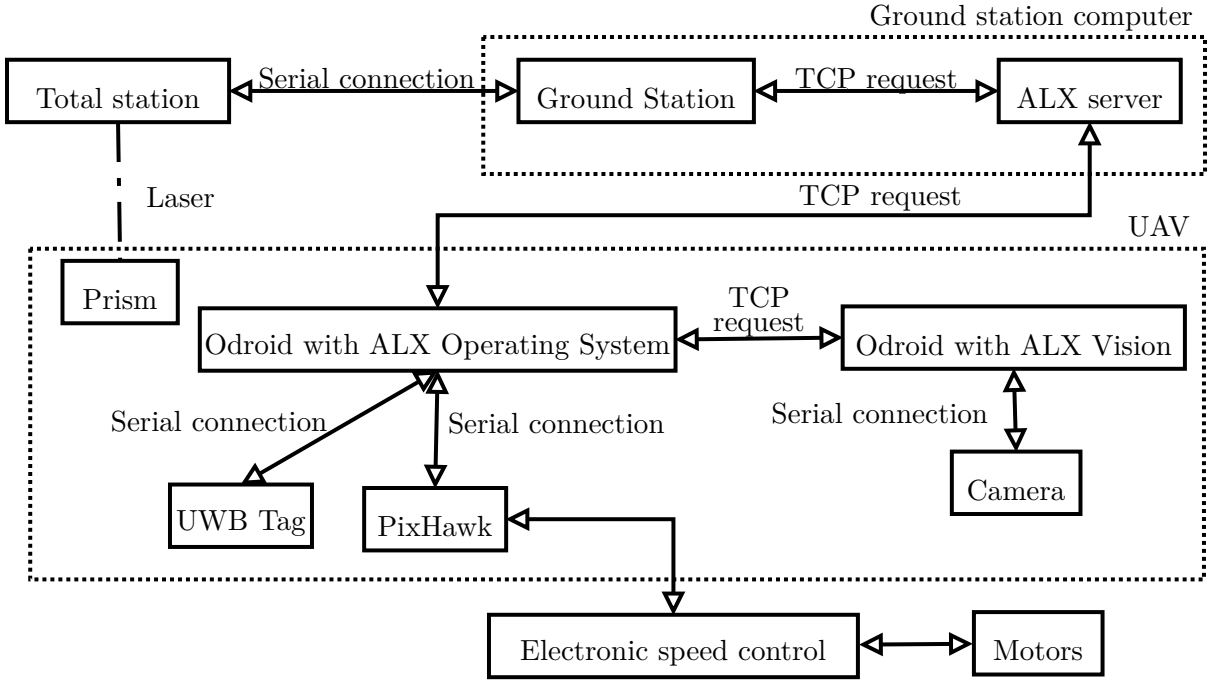


Figure 7.3: Overview of general architecture the system

7.4 Integration of the image processing

Concerning the system of computer vision conceived by the company, the architecture is formed by a chain of modules. The advantage is the modularity of the system. Indeed, one module implemented for one project can be easily used to another one. In this purpose, we have integrated our work in their system by creating a module called "Drovisio". The module takes as input a video stream and outputs the information about objects detected with the translation and rotation vectors.

About the decision-making for the instructions of the drone, it is carried out in the ALX Operating System by aggregating the different positioning system. The strategy during the flight will be discussed in the next chapter.

CHAPTER 8

Strategy

While the previous chapter discussed in details the architecture deployed, this chapter focuses on the strategies the drone can follow, using our solution, in order to accomplish the task.

Unfortunately, we were never able to properly test the quality of the strategies during a real-world test of the drone because we weren't able to find enough time with ALX Systems to install the system of the drone and perform a full-chain test of our solution. Indeed some unexpected difficulties were encountered when they were available for us. For example, the small drone was not enough stable for their system. This chapter explains the issues we ran into when testing theoretical scenarios, and what we changed in our solution to accommodate those issues. These techniques and scenarios can be used and improved upon in further research.

8.1 Configuration of the markers on the brick

Different configurations have been considered for the positions of the markers on the brick.

8.1.1 Color markers

For the color markers, we first simply placed 6 red markers in a 2-by-3 rectangle shape configuration, with equal distance between each point, as shown in Figure 8.1.

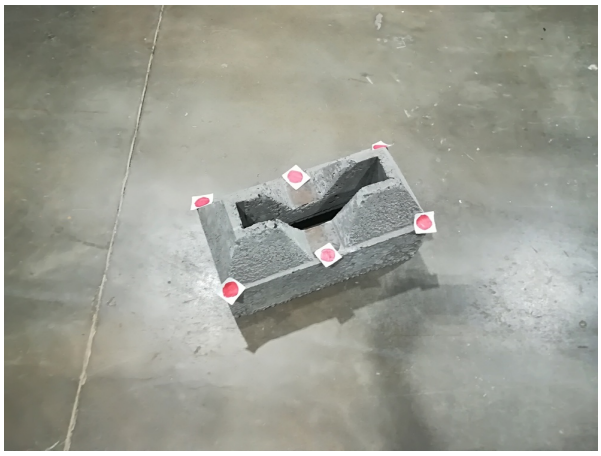


Figure 8.1: Configuration of 6 red colors spots in a rectangular shape

To make the algorithm more robust, and to avoid false positives, we decided to change the color of the middle points, as shown in Figure 8.2. Indeed, if other red spots were present in the picture, they could sometime interfere with the color marks on the brick, creating false bricks.

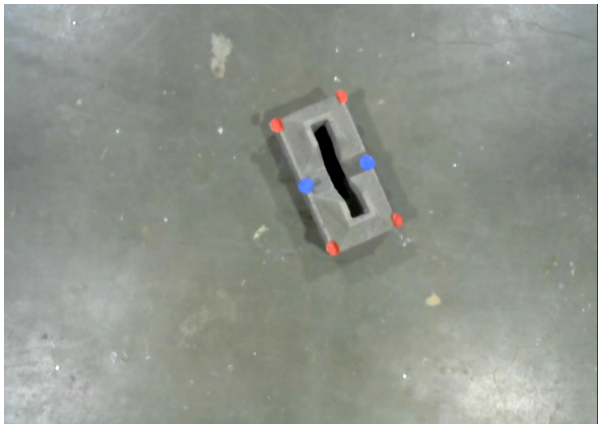
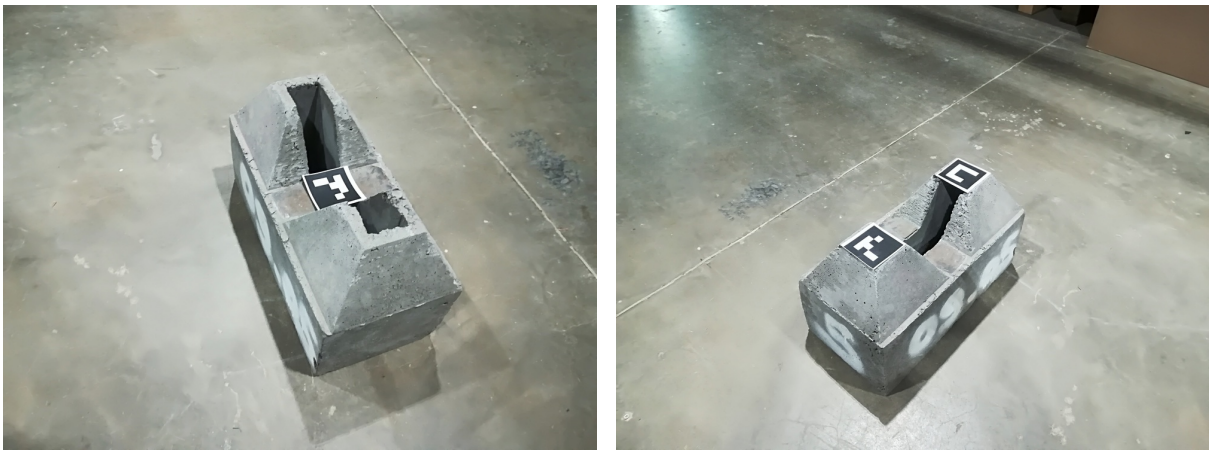


Figure 8.2: Configuration of six color spots (four red and two blue) in a rectangular shape

This configuration offered the precision we needed for the problem, but other configurations could be the subject of further research. For example, to remove the coplanar nature of the points, color points could be added on the top of the pyramid of the bricks. The increased precision gained by using more than 6 points compared to the added computational cost can also be studied. Moreover, with the dricks, if the camera is too much tilted, the points will not appear due to the slope of the pyramid on top of it. However, as you can observe on the Figure 8.2, if the camera is parallel to the ground, it causes no problem.

8.1.2 ArUco codes



(a) Single ArUco code

(b) Two ArUco codes

Figure 8.3: Different configurations for the ArUco codes on the brick

We first considered a single ArUco code, placed at the top of the brick between the two pyramids, as shown in Figure 8.3a. The biggest issue with this solution is that the drone will not be able to see an ArUco code that is half-covered by another brick.

A way to solve this problem is to use two ArUco codes, each placed on top of the pyramids of the block, as shown in Figure 8.3b. This would allow the drone to still see at least one code with a brick is on top of another brick. Furthermore, using two codes increases the precision of the

positioning algorithm, as more points are available to determine the spatial position of the brick. Additionally, the surface at the top of the pyramid (7cm-by-7cm) is bigger than the available area in the middle of the brick (6.5cm-by-6.5cm).

A potential downside is the increased computational costs of detecting twice as many codes, though Section 6 shows that the impact of the number of markers doesn't impact that much the achieved framerate of the algorithm.

8.2 Position of camera

The position of the camera on the drone has a great impact on the capability of the drone to detect the brick on which it needs to land on. First of all, the resolution of the camera affects the maximum altitude at which the drone can detect a brick on the ground, as well as the precision of the positioning parameters that the algorithm will output. Nevertheless, the markers can be noticed at a distance of more than 2m, sufficient in our case.

The field-of-view of the camera directly limits the area that the drone can scan when it lands on the brick. The webcam we used offers a field-of-view of 60°. Wide field-of-view lenses could be used for our application, though a greater care must be taken with the calibration process since these lenses suffer more from the distortion effect.

8.3 Global scenario

The global scenario can be divided into 4 big steps the drone has to perform repeatedly:

- First of all, the drone must travel to the location of the brick to pick.
- Secondly, once the drone is approximately close to the brick, it must land on the brick to pick it up and get back to an altitude above 1m.
- Third, once the brick is picked up, the drone must travel to the location of the wall
- Finally, once the drone is approximately close to the section of the wall on which the brick must be placed, the drone will land on the wall, drop the brick and get back to an altitude above 1m.

The scenario can be represented as a Finite State Machine, as shown in Figure 8.4, where the states represent the different steps that the drone must follow, and the transitions the different readings of the sensors during the scenario.

The schema starts at the upper-left corner, the drone will first take off and achieve a "high" altitude (here 1m). Once done, it will enter its taking phase, go to the approximate location of a brick. If something unexpected happens, a retry mechanism is triggered to redo the process form the beginning. If the drone retries and fails 3 times, it will try to reach the safe zone, land and shut down.

If the drone successful picks up a brick, it will travel to the wall and begin its dropping phase. In the same vein than the taking phase, the drone will try to drop the brick (possibly retrying if something fails). Again, if the drone fails 3 times to drop the block, it will fly to the safe zone. If the drone successfully drops the block, it will go back to its taking phase to redo the process.

Notice that the scenario can be completely encoded before the flight in order to know if the behavior of the drone corresponds to the expectations.

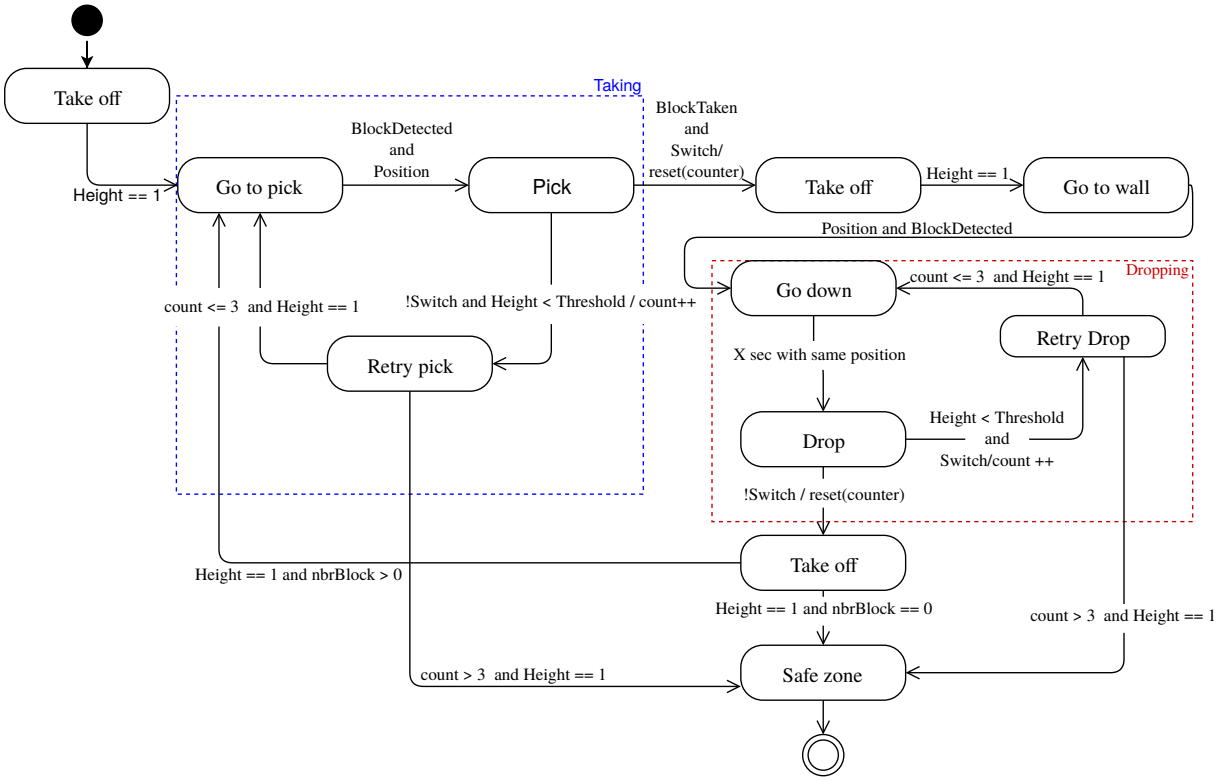


Figure 8.4: Finite state machine of the drone

8.4 Scenario applied to the dricks

Computer vision is specifically used during two phases of the scenario: the two landing phases the drone has to perform to pick up the brick and drop it on top of two other bricks.

8.4.1 Taking

For the taking phase of the scenario, a perfect situation can be set up: a single brick fairly isolated from other bricks must be picked up by the drone. In this situation, illustrated at the Figure 8.5, the drone would first hover over the brick and try to guide itself to be just above it, at a height between 1m and 2m. The drone can use the measures of positions given both by the total station and the computer vision to confirm its location (and also to make sure one of the two systems are giving approximately the same position). Once the drone is above the drone, it can rotate itself to be aligned with the brick, using the measurement from the computer vision algorithm.

Once the drone is properly aligned, it can go down. The different measures can be used to maintain the stability and correct the trajectory of the drone during the descent. At some point, approximately 20cm above the brick depending on the position of the camera, the drone will no longer be able to see the markers on the brick. When this altitude is reached, the drone should try to lower itself as straight as possible, using the measurement from the other systems.

8.4.2 Dropping

This scenario differs slightly from the previous scenario. The drone, in this situation, must drop the brick on top of two bricks that are already placed.

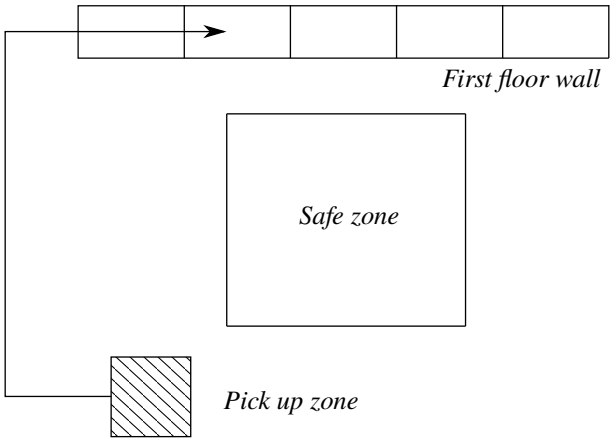


Figure 8.5: Overview of the scenario applied to the dricks.

The first step is to select the two right dricks to drop the one boarded by the UAV. Indeed, the drone can't drop the dricks anywhere because dropping a drick between two already placed dricks is really difficult even impossible. With the color points, no information iss present to distinguish two dricks. In order to respect the scenario, the drone can fly over the wall always with the same manner as indicated with the arrow on the Figure 8.5. Thanks to that, the drone will drop the first drick between the first two dricks of the floor wall by knowing how many dricks is overflight and so forth.

This problem is not present with the ArUco code because the dricks can be distinguished by the id of the ArUco codes laid on it. In this case, the UAV can directly go to the right place.

A second issue that arises with color point is the way a marker can be hidden by a brick that was already placed on top of a brick on which the drone wants to land on. Figure 8.6 shows how some of the markers of the right brick are hidden by the brick placed above it. Even if the color markers of the lower brick are hidden by the higher brick, the algorithm could still use the markers of the top brick, and take into account the difference of height between the lower markers and the higher markers when computing the spatial position of the bricks. However, for a future work, it can be imagined to modify the algorithm so that it would be able to find *parts* of bricks, that is the arrangement of markers where one or more markers are missing.

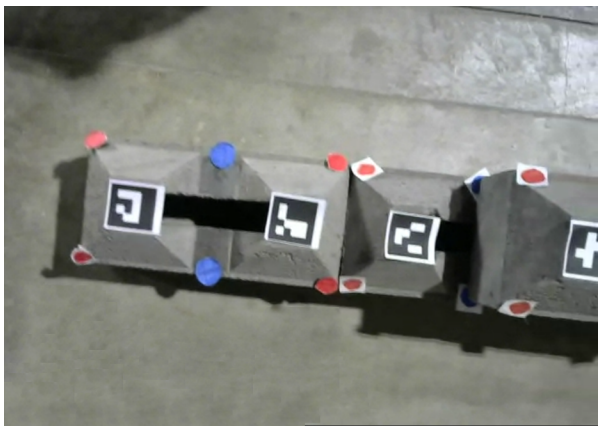


Figure 8.6: View from the drone of a possible configuration for the dropping phase

With the ArUco codes, we can solve by putting two codes by brick, one on top of each pyramid of the brick, ensuring at least one code is always visible.

Finally, in the case of the dricks, the markers seem to be the best solution because it can easily

distinguish the dricks visible by the UAV.

8.4.3 Security

Multiple safety rules must be put in place to avoid security hazards when dealing with drones this size. Moreover, when the ALX Operating System takes the control of the drone, it is currently impossible to switch to a remote control.

In general, we adopt a more defensive and pessimist attitudes when interpreting the readings from the sensor. If the given position by the total station and the one given by the computer vision algorithm (relative to the brick) differs too much, the drone will stop what it's doing and try to go the safe zone.

If the drone is not sure that the brick has been correctly taken via the picker sensor, but that the position sensors (both from the total station) give the correct location, then the drone will assume something is going wrong and will retry the whole picking process.

This way of designing the scenario assures us that we minimize the possible faults during the scenario that could result in damage to the drone.

CHAPTER 9

Conclusion

We presented in this thesis the role and the constraints of the computer vision component used in the larger project of realizing a UAV capable of building a structure without any human assistance.

We presented a first method of detecting the bricks via color points placed on the brick. The second method used the ArUco algorithm, which uses squared black-and-white markers.

We also explored a positioning algorithm, solvePnP, which allows the drone to compute the spatial position of a brick, and the different ways to properly configure it, the calibration, for example.

After reviewing the precision those two methods can offer, we conclude that both can be used to help the drone guide itself, as they both respect the constraints imposed by the situation. The first method yields, in general, better rotation measures while the second offers better translation measures.

We were unfortunately not able to make a real-world test of our solutions with the drone, as we couldn't find enough time with ALX Systems to properly set up the framework on the drone and install all the components developed this year. However, the module is already integrated into the computer vision framework, ALX Vision. Nonetheless, we presented different strategies that can be implemented on the drone to help it accomplish its tasks, as well as some design decisions that must be taken into account when implementing the solution on the drone.

As described in Section 6, our solution is configurable to other kinds of elements. The solutions can be adapted to new building elements. Moreover, the project can also be used for different kind of drone-related applications: to help a drone land on a platform, for example.

9.1 Difficulties encountered

On a smaller note, dealing with the different ways of expressing a rotation has been slightly troublesome for us, as the different resources on the subject sometimes implicitly use different formalisms. For example, OpenCV outputs axis-angle rotation vectors. Many rotation formalisms in three dimensions exist, and while the Euler angles are the most intuitive ones (it corresponds to a combination of three rotations along the three axes), it is not used in general in mathematical computations.

The collaboration with an extern company caused us some troubles during our work. As explained in the report, we were not able to make real-world tests due to a lack of time with ALX Systems

to set up the framework totally on a drone. The company brought us their knowledge and the quality of their work. However, we were totally dependent on them and their availability. Moreover, the distance between Louvain-La-Neuve and Liège didn't help. In the future, why not imagine to be the owner of the drone control system by developing it on an open-source framework on top of ROS (Robot Operating System), though it would ask more knowledge about electromechanics and drone controls.

9.2 Limitations and upcoming works

There are multiple limitations to the current implementation of the different algorithms that can be addressed in future works.

The current placement of color markers doesn't allow the drone to see all the markers when a brick hides some of the markers of the brick below. A possible way to solve this would be to allow the algorithm to do shape matching with only a fraction of the original markers or possibly adding more markers in visible areas.

In the same vein, the current configuration of ArUco (that is, one at the center of the brick) makes the code unreadable when another brick is placed on top of it. As stated before, a simple solution would be to place two ArUco codes, one of the top of each pyramid of the brick.

The camera used in the project might not cover enough ground area once on the drone, due to its low field-of-view. Lenses attachable to special cameras exist and allow wider field-of-views, at the cost of a bigger fish-eye effect and a more expensive price.

Another area that could be explored is the use of Simultaneous Localization And Mapping (SLAM) to better help the drone navigating its environment and avoid collisions. SLAM works by using the spatial reading of sensors (such as the depth map of a stereo-vision feed, for example) to simultaneously build a map of the surrounding environment and locate the drone in this environment. This could be used by the drone to detect the brick as well as the surrounding obstacles, to make sure that there is enough space around the brick without bumping into something else.

Moreover, the solutions proposed are adaptable for different kinds of building elements but it could be advantageous to develop a dedicated computer vision solution for a specific element, as it would make it possible to benefit from the characteristics of that and to be more robust.

APPENDIX A

Comparison Odroid XU4 and Raspberry Pi 3

	Odroid XU4	Raspberry Pi 3
Dimension	83 x 58 x 22 mm approx.	85x 58 x 14 mm approx.
CPU	Samsung Exynos5422 Cortex^{TM}-A15 2GHZ and Cortex^{TM}-A7 Octa core	1.2 GHz 64-bit quad-core ARMv8
GPU	Mali-T628 MP6	Broadcom video core 4
RAM	2GB LPDDR3 RAM PoP stacked	1 GB
Network Connectivity	Gigabit Ethernet	Wi-Fi 802.11n & Bluetooth 4.1 / BLE & 10/100 Ethernet
Power Source	5V/4A Power Jack	5.1V, 2.5A microUSB
USB	2xUSB 3.0 & 1xUSB 2.0	4xUSB 2.0
Storage	eMMC5.0 and rMicro SD	microSD

APPENDIX B

Delivered content

The `zip` file delivered with this report contains two folders, as well as a `README.md` file that contains some information to an interested reader.

B.1 Calibration

`calibration/` contains the necessary script to perform a camera calibration. The script is based on the script given by OpenCV, but with added features to more easily perform the whole process.

The `README.md` contains information about the command to calibrate the camera and some tips and examples.

B.2 Detector

`detector/` contains the main component project, that is all the code to detect bricks out of an image. The project uses `CMake` to handle the build and package the source code and `CMakeLists.txt` contains the different build target available.

The source code is located in `detector/src/`, and the files that may interested a curious reader are `detector_color.cpp` and `detector_ArUco.cpp`. The two build target that produce executables implementing these methods are `color` and `ArUco`.

Moreover, there is the implemented module used by the ALX Vision called `Drovisio`

The `README.md` file inside the folder contains information about the additional source files we have written during the course of the project, as well as the commands to build and run the project.

`detector/videos` contains the videos we have used to produce the different measures examined in Section 6 and that can be used to visualize the performance.

Bibliography

[1] ALX Systems website. http://www.alxsys.com. Accessed: 2018-05-14.

[2] Boston Dynamics. https://www.bostondynamics.com/atlas.

[3] Camera Calibration. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html. Accessed: 2018-05-25.

[4] Camera Calibration and 3D Reconstruction. https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. Accessed: 2018-05-25.

[5] Convert from HSV to RGB Color Space. http://radio.feld.cvut.cz/matlab/toolbox/images/color4.html. Accessed: 2018-04-30.

[6] Gimbal Lock. https://en.wikipedia.org/wiki/Gimbal_lock.

[7] Image Processing Toolbox - Using rgb2ind. https://www.mathworks.com/help/images/convert-from-hsv-to-rgb-color-space.html. Accessed: 2018-04-30.

[8] Logitech HD Webcam C270. https://www.logitech.com/en-us/product/hd-webcam-c270?crid=34. Accessed: 2018-05-23.

[9] Logitech HD Webcam C270 Specifications. http://support.logitech.com/en_us/article/17556. Accessed: 2018-05-23.

[10] OpenCV : Camera Calibration. https://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html. Accessed: 2018-05-21.

[11] OpenCV (Open Source Computer Vision Library). https://opencv.org/. Accessed: 2018-05-9.

[12] ZBar bar code reader. http://zbar.sourceforge.net/. Accessed: 2018-05-21.

[13] F. Augugliaro, S. Lupashin, M. Hamer, C. Male, M. Hehn, M. W. Mueller, J. S. Willmann, F. Gramazio, M. Kohler, and R. D'Andrea. The Flight Assembled Architecture installation: Cooperative construction with flying machines. *IEEE Control Systems*, 34(4):46–64, Aug 2014.

[14] F. Augugliaro, A. Mirjan, F. Gramazio, M. Kohler, and R. D'Andrea. Building tensile structures with flying machines. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3487–3492, Nov 2013.

[15] M. R. Bloomberg, D. J. Burney, and D. Resnick. *BIM Guidelines*. New York City Department of design and construction, July 2012.

[16] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., 2nd edition, 2013.

[17] C. Brown and Duane. Close-Range Camera Calibration. 37, 12 2002.

[18] J. Canny. A computational approach to edge detection. In *Readings in Computer Vision*, pages 184–203. Elsevier, 1987.

[19] C. Coppieters de Gibson. Environmental and financial feasibility of a drone-built architectural structure. Master's thesis, Université Catholique de Louvain-la-Neuve, 2016.

[20] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*, pages 726–740. Elsevier, 1987.

[21] X-S. Gao, X-R. Hou, J. Tang, and H-F. Cheng. Complete solution classification for the perspective-three-point problem. *IEEE transactions on pattern analysis and machine intelligence*, 25(8):930–943, 2003.

[22] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014.

[23] Information technology – Automatic identification and data capture techniques – QR Code bar code symbology specification. Standard, International Organization for Standardization, 2015.

[24] Alex Krizhevsky, I. Sutskever, and G.E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[25] P. Latteur, S. Goessens, J-S. Breton, J. Leplat, Z. Ma, and C. Mueller. Drone-Based Additive Manufacturing of Architectural Structures. Master's thesis, 08 2015.

[26] P. Latteur, S. Goessens, N. Rogeau, G. De Beusscher, and C. Mueller. Parametric Design of Drone-Compatible Architectural Timber Structures, 07 2018.

[27] J-F. Leboutte and V. Parisel. Développement d'un système constructif "drone-compatible" pour la construction de maisons unifamiliales en maçonnerie et en béton. Master's thesis, Université Catholique de Louvain-la-Neuve, 2017.

[28] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate O(n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[29] J. Leplat and J-S. Breton. Feasability study for drone-based manufacturing of architectural. Master's thesis, Université Catholique de Louvain-la-Neuve, 2015.

[30] Q. Lindsey, D. Mellinger, and V. Kumar. Construction of Cubic Structures with Quadrotor Teams. Master's thesis, 06 2011.

[31] C. Manderliert and T. De Furstenberg. Etude de faisabilité de structures en bois drone-compatible. Master's thesis, Université Catholique de Louvain-la-Neuve, 2017.

[32] A. Maxim, O. Lerke, M. Prado, M. Dörstelmann, A. Menges, and V. Schwieger. UAV Guidance with Robotic Total Station for Architectural Fabrication Processes. Master's thesis, Universität Stuttgart, 2017.

[33] N. Moroney, M. D. Fairchild, Robert W. G. Hunt, C. Li, M. R. Luo, and T. Newman. The ciecam02 color appearance model. In *IS&T/SID 10 th Color Imaging Conference*, pages 23–27, 2002.

[34] A. Naveau and A. Moncourrier. Development of a Drone-Compatible Masonry Construction System. Master's thesis, Université Catholique de Louvain-la-Neuve, 2016.

[35] N. Nerhi and A. Paques. Étude et conception de l'architecture de contrôle d'un drone destiné à la construction d'une structure en intérieur. Master's thesis, ECAM/UCL, 2016.

[36] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. *arXiv*, 2018.

[37] M. Reniers. Development of elementary BIM tools for drone-compatible construction systems. Master's thesis, Université Catholique de Louvain-la-Neuve, 2016.

[38] D. Rose. Quaternions. `http://danceswithcode.net/engineeringnotes/quaternions/quaternions.html`. Accessed: 2018-05-29.

[39] K. Zuiderveld. Contrast Limited Adaptive Histogram Equalization. In Paul S. Heckbert, editor, *Graphics Gems IV*, pages 474–485. Morgan Kaufmann, 1994.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**