A Thesis submitted for the degree of Master in Artificial Intelligence

Efficient Recognition Approaches for the Interaction Between Humans and Aerial Robots

Alejandro José Hernández Ruiz

Defense date: April 20, 2016
Last revision: April 14, 2016

Directors: René Alquézar Mancho[1,2], Michael Villamizar Vergel[2]
from: 1) Department of Computer Science (UPC)
2) Institut de Robòtica i Informàtica Industrial (CSIC-UPC)

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
FACULTAT DE MATEMÀTIQUES (UB)
ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA (URV)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)
UNIVERSITAT DE BARCELONA (UB)
UNIVERSITAT ROVIRA I VIRGILI (URV)

Contents

List of Figures iii

List of Tables iv

Glossary v

Acknowledgements vi

Abstract vii

1 Introduction 1

2 Background 5
2.1 Geometry Based Methods . . . 5
2.1.1 Thresholding . . . 5
2.1.2 Feature Detection . . . 6
2.1.3 Feature Description and Matching . . . 8
2.2 Machine Learning Based Methods . . . 10
2.2.1 Least Square Optimization Methods . . . 10
2.2.2 Boosting . . . 11
2.2.3 Decision Trees . . . 11
2.2.4 Random Forests . . . 11
2.2.5 Artificial Neural Networks . . . 12
2.2.6 Support Vector Machines . . . 12
2.3 Detection and Recognition . . . 13

3 Technical Framework 14
3.1 Flying Robot Platform . . . 14
3.2 Robot Operating System . . . 14
3.2.1 ROS Architecture . . . 15
3.2.2 ROS Messages . . . 16
3.2.3 LabRobotica Philosophy . . . 17
3.2.4 Platform deployment . . . 18
3.2.5 ROS on Android Devices . . . 19
3.3 OpenCV . . . 20
3.4 Experimental Guidelines . . . 20

4 Visual Marker Recognition **22**
4.1 Method . . . 22
4.1.1 ARToolkit and ar_pose . . . 23
4.1.2 ArUco . . . 24
4.1.3 bar_pose . . . 25
4.1.4 Marker Tracking . . . 25
4.1.5 Color Thresholding . . . 26
4.2 Results . . . 28
4.2.1 Experimental Setup . . . 29
4.2.2 ar_pose . . . 29
4.2.3 bar_pose . . . 30
4.2.4 Color Thresholding . . . 32

5 Face Detection **33**
5.1 Method . . . 33
5.1.1 Viola-Jones . . . 34
5.1.2 Face Tracking . . . 35
5.1.3 Face Distance Estimation . . . 36
5.1.4 Face Pose Estimation . . . 39
5.2 Results . . . 40
5.2.1 Detector Performance . . . 40
5.2.2 Detector with Tracking Performance . . . 41
5.2.3 Distance Estimation Performance . . . 41
5.2.4 Pose Estimation Performance . . . 43

6 Object Recognition **45**
6.1 Method . . . 45
6.1.1 Random Ferns . . . 47
6.2 Results . . . 47

7 Interaction Scenario **50**
7.1 Scenario Design . . . 50
7.1.1 Behavior Design . . . 52
7.1.2 Evaluation Design . . . 53
7.1.3 Procedure and Settings . . . 53
7.2 Results . . . 54

8 Conclusions **56**

Bibliography **58**

List of Figures

3.1 LabRobotica Philosophy Diagram 18
3.2 ROS architecture for Ona platform and final deployment 19
3.3 ROS architecture for test platform and test deployment 20

4.1 Marker Recognition Output 30
4.2 Marker Tracking 31
4.3 Marker Shadowed and Color Threshold 32

5.1 Diagram for the face distance estimation formula. FD is short for FaceDistance. 37
5.2 Face Detection Output 40
5.3 Distance Estimation using interpupillary distance 42
5.4 Distance Estimation using head breadth 42
5.5 Face Pose Estimation 43

6.1 Object Recognition Output 48

7.1 Robot controller android application GUI 51
7.2 Robot behavior FSM 53

List of Tables

3.1 Ona Platform Components 15

4.1 ar_pose Visual Marker Detector Confusion Table 30
4.2 bar_pose Visual Marker Detector Confusion Table 31

5.1 Viola-Jones Face Detector Confusion Table 41
5.2 Face Detector with Tracking Confusion Table 41
5.3 Average Measurements in Distance Estimation Methods 43
5.4 Face Detector with Pose Estimation Confusion Table 44

6.1 Object Recognition Confusion Table 48

7.1 Results of the scenario tests 54
7.2 Results of the pre and post-test questionaires 55

Glossary

AdaBoost: Adaptive Boosting method, see Chapter 2.
CNN: Convolutional Neural Network, see Chapter 2.
DMP: Dynamic Movement Primitive, see Chapter 2.
DoG: Difference of Gaussians, see Chapter 2.
DoH: Determinant of Hessian, see Chapter 2.
DPM: Deformable Part Model, see Chapter 6.
FPS: Frames Per Second.
FSM: Finite State Machine.
GUI: Graphical User Interface.
HOG: Histogram of Oriented Gradients, see Chapter 2.
HRI: Human Robotics Interaction.
IRI: Institut de Robòtica i Informàtica Industrial, see 1.
IPD: Interpupillary Distance, see Chapter 5.
LoG: Laplacian of Gaussian, see Chapter 2.
NN: (Artificial) Neural Network, see Chapter 2.
OS: Operating System.
PID: Proportional Integral Derivative controller, see [25].
PCA: Principal Component Analisys.
RANSAC: RANdom SAmple Consensus, see Chapter 2.
RNN: Recurrent Neural Network, see Chapter 2.
ROS: Robot Operating System, see Chapter 3.
SGD: Stochastic Gradient Descent, see Chapter 2.
SVM: Support Vector Machine, see Chapter 2.

Acknowledgements

I would like to thank my supervisor, Professor Alquezar, my co-supervisor Dr. Villamizar, and Professor Sanfeliu the director of the project, for giving me the opportunity to be part of the project. It has being a wonderful experience to work with you, and I learned many things from it.
Thank you.

Abstract

The flying robots are one of the most popular robots among both researchers and public, and this is because they have an incredible potential to perform many different tasks. Its main advantage is its ability to move very swiftly, and not having to deal with most of the obstacles that the ground mobile robots need to overcome. The main disadvantage is that they are limited in size and weight, which in turn limits the hardware it can carry. It is a technological challenge to build a good flying platform, specially if we also want very good sensors and computing power. The hardware limitations in turn will also impact on the software, making a challenge to build reliable and efficient software for these platforms.

The *Institut de Robòtica i Informàtica Industrial* or IRI, is a Joint Research Center that is part of UPC and CSIC. Within the IRI there are four lines of research: 1) Automatic Control, 2) Kinematics and Robots Design, 3) Mobile Robotics, 4) Perception and Manipulation. Each line of research has their own staff, laboratory and projects. This work is part of the Robot-Int-Coop project, one of the projects of the Mobile Robotics line, and it is currently being developed in the mobile robotics laboratory (LabRobotica).

The Robot-Int-Coop project is centered in the interaction between humans and robots, with special focus on the tasks that require the cooperation of both. The main goal of the project is: "to advance in the design of mobile and flying robots that can interact, learn and cooperate with people for urban tasks, adapting themselves to the environment and changing conditions.". It is within this goal that we define the objectives and requirements for our work.

The general objective of this work is to have a baseline implementation of Computer Vision algorithms that can be used to develop different use cases on the flying platform. The algorithms must aid the robot in the navigation in the environment and the interaction with people and objects. Also the algorithms should ideally run within the platform, making it autonomous, this is specially useful in the exploration scenarios, but also it is ideal for the control of the platform in any situation, since any delay in communication harms the stability of the platform.

The specific objectives of this work are: 1) To implement a visual marker recognition method, 2) To implement a face detection method, 3) To implement a general object recognition method. And these objectives are subject to the following requirements: The methods should be implemented according to the LabRobotica standards. The methods should be efficient, allowing them to be run at reasonable frame rates within the flying platform.

Abstract

The visual marker recognition is today considered a classic approach to the object recognition and pose estimation problem. In this approach, instead of detecting any object in the real world, a marker is specifically designed and placed in the scenario. Then the marker would be detected by an algorithm that usually follows these steps: 1) Thresholding, 2) Edge Detection, 3) Marker Matching, 4) Pose Estimation. We experimented with three visual marker detection algorithms: 1) ARToolkit, 2) bar_pose, 3) ArUCO. We also implemented a marker tracking algorithm to improve the frame rate, and a color thresholding algorithm to help the detection of the marker in outdoor scenarios with strong shadows.

The face detection is a classic problem in computer vision, and it is an interesting problem because it enables the visual interaction with humans. To perform the face detection in this work, we have decided to use the Viola-Jones method, or more specifically its implementation in OpenCV. The Viola-Jones method is a popular method for object detection, mostly used for faces today, and it consists in four steps : 1) Haar-Like Feature Selection, 2) Creating an Integral Image, 3) Adaboost Training, 4) Cascading Classifiers. We also implemented a face tracking algorithm, a face distance estimation algorithm and adapted a face pose estimation algorithm.

The object recognition is the more general and the more difficult task of the project, and we would like an algorithm that is efficient, robust and that can generalize to many different scenarios. Due to the project efficiency requirements, and also taking into account its flexibility for many tasks, the Random Ferns were chosen as the candidate method to implement object recognition in this work. The random ferns are a new type of classifier, comparable to the random forests, but they work in a different way. Applied to images, the random ferns have the following steps: 1) Feature Detection, 2) Data augmentation, 3) Feature Extraction, 4) Statistics computation (ferns training), 5) Classification.

Using the algorithms implemented for this work, we developed the command listener scenario. This scenario is a simple, yet general experiment that follows this story line: commands are given to a flying robot either by buttons, voice or gestures, and the task of the robot is to find its way back to a landing base. This demonstration can actually be extrapolated to many specific use cases, for example: following a person or searching for a particular object.

Finally we can conclude that the methods implemented are a baseline for each of the tasks, accomplishing the objectives of the project. Also important to mention is that the interaction scenarios serve as a demonstration of how the implemented methods can be used to perform complex HRI tasks. The software is ready to be deployed in the final platform, since it was all developed following the standards, and no further adaptation should be necessary for it to run on any ROS platform.

1 Introduction

The beginning of the Computer Vision field was full of naive approaches that tried to solve the human vision with simple geometry and calculus techniques, and a classical example of these approaches is the summer vision project by Papert [46] in 1966; in this project, Prof. Papert proposed to solve the human vision through the pattern recognition methods that were state of the art by that time. The summer vision project failed to achieve its ambitious goal, but it started the flame for the computer vision research, a field that has been growing steadily from that time, and that is nowadays one of the hottest topics in research for computer science, and its advances are causing big impacts in many other fields of science and technology.

For the robotics field, the Computer Vision field is one of its main partners, because it allows to use the input from cameras and other visual sensors and use it to take decisions and guide the robot on its task. In industrial robotics, many tasks such as the automatic assembly lines and automatic quality assurance are nowadays best practices, and they are the foundation for the manufacture processes of many industries such as the electronics and automotive industries. In these industrial tasks, computer vision methods are not only helpful, but sometimes are the only known solution that can achieve the goal. However, in industrial deployments, usually the conditions are very well known and studied, and therefore they can be controlled to assure the optimal outcome of the task.

On the other hand we have the growing field of Mobile Robotics, which is as its name implies the use of robotics with platforms that can move. Its ability to move makes them very useful to perform many tasks in different fields, however this also poses new technological challenges that are not present in general in the industrial robotics. The mobile robots usually have a constraint on its size, which means that to fit all the necessary hardware we must often sacrifice its quality or capability. This translates on sensors of less resolution, and less computational resources to process the signal.
Nonetheless, the advancement in the electronics, specially in the technology of batteries and efficient circuits and motors, it is easier and cheaper than ever to build aerial robots, and these robots may have many different use cases in urban scenarios, for example: to assist firemen and rescue teams, to transport small packages in urban areas or to assist the police in surveillance tasks. But all of such tasks rely on the assumption: the robot must be capable to identify its environment, the objects and the people nearby, and also must be capable to accurately estimate its location in the world. This assumption implies that we need to have foundational methods upon which we could build more complex projects. Because of this necessity, we devoted this work to test and improve computer vision methods for mobile robots.

The *Institut de Robòtica i Informàtica Industrial* (Institute of Robotics and Industrial Informatics) or IRI for its acronym, [1] is a Joint Research Center of the Technical University of Catalonia (UPC) and the Spanish Council for Scientific Research (CSIC). Within the IRI there are four lines of research: 1) Automatic Control, 2) Kinematics and Robots Design, 3) Mobile Robotics, 4) Perception and Manipulation. Each line of research has their own staff, laboratory and projects. This work is part of the Robot-Int-Coop project, one of the projects of the Mobile Robotics line, and it is currently being developed in the mobile robotics laboratory (LabRobotica).
The Robot-Int-Coop project is centered in the interaction between humans and robots, with special focus on the tasks that require the cooperation of both. The main goal of the project is: "to advance in the design of mobile and flying robots that can interact, learn and cooperate with people for urban tasks, adapting themselves to the environment and changing conditions." [2]. And it is within this goal that we define the objectives and requirements for our work.

The flying robots main advantage is their ability to move free in space, and this has being exploited in many commercial applications: photography[3], landscape exploration and mapping and express delivery[4], among others. All of these are very interesting tasks, but they rely on two assumptions:

1. The robot is outside, which means it can move freely and the interaction with people is minimal
2. The robot uses special sensors, e.g. GPS or proximity sensors, to orient with respect to the world or to the user.

These two assumptions are in conflict with the Robot-Int-Coop proposition, because in our project we precise the following conditions:

- The robot should be able to move in indoor environments and safe around people.
- Ideally, the robot should only rely on common sensors to perform its task.

The reason for these conditions, is that in many interesting use cases in urban areas, the robot will naturally have these conditions as hard constraints in the environment.

A practical example that illustrates the necessity for the conditions of the project: imagine that we want to use the robot to serve as a tour guide, it would need to position itself close enough to communicate with people, but stay within a safe distance. It would also need to position itself with respect to the environment, and follow the route of the tour. And of course it also would need to recognize the important places or objects during

[1] `http://www.iri.upc.edu/`
[2] `http://www.iri.upc.edu/project/show/144`
[3] `https://www.lily.camera/`
[4] `http://www.amazon.com/b?node=8037720011`

the tour, and communicate promptly the information to the tourists. Of course many parts of this example could be achieved easier with the use of special sensors and some tricks, but in the general sense, the example is valid and it is still an unsolved problem.

Inspired in use cases similar to the one described in the example, we can define the general objective of this work:
To have a baseline implementation of Computer Vision algorithms that can be used to develop different use cases on the flying platform. The algorithms must aid the robot in the navigation in the environment and the interaction with people and objects. Also the algorithms should ideally run within the platform, making it autonomous, this is specially useful in the exploration scenarios, but also it is ideal for the control of the platform in any situation, since any delay in communication harms the stability of the platform.

The specific objectives of this work are:

- To implement a visual marker recognition method: marker recognition is the baseline approach in this project, because they are a very simple, but also reliable and efficient method. We can use the markers to tag areas of the environment and objects, and use these tags as cues for the navigation for the robot, or identifiers for the objects, or even visual commands that can be read at a long range.
- To implement a face detection method: essential to position itself at a proper distance to users, which is required to establish a proper communication with a human, and also useful to avoid accidents by colliding with people. These two tasks are the core of any use case involving cooperation between humans and robots, therefore it deserve a special attention, even when more general approaches, like the object recognition, cover them.
- To implement a general object recognition method: The object recognition would allow us to identify the interesting objects in the environment, and with this information we can implement many different behaviors, e.g. to say relevant information about the object in a tour, or to search for an object and retrieve it, or even to spot a complex landmark like a statue, and position itself with respect to it. Due to its very general definition this is the most difficult task in the project, from the point of view of the Computer Vision field, and usually the algorithms are also more computationally expensive. For this reason we would use them for tasks that cannot be accomplished by simpler approaches.

And these objectives are subject to the following requirements:

- The methods should be implemented according to the LabRobotica standards, which includes using C++, ROS, OpenCV and the LabRobotica philosophy.
- The methods should be efficient, allowing them to be run at reasonable frame rates within the flying platform. This implies that the hardware constraints of the platform are one of the main drivers of the design.

The chapters of this report are organized in the following manner:

1. Introduction: introducing the topic, explaining the context for the work. Current chapter.
2. Background: briefly describes the theory and methods which are the foundation for this work.
3. Technical framework: describes the technical context and tools that were used for the development of this work.
4. Marker recognition, face detection, object recognition: are the main content of this report, in these chapters the research and the implementation for each method is explained in detail.
5. Interaction scenario: in this chapter a demonstration scenario is implemented using the methods that were developed for this work.
6. Conclusions: as it name implies this chapter concludes the work, it contains the usual reflections and recommendations for future work.

2 Background

In this section we will briefly explain the methods and concepts which are the theoretical foundation of the work, and that are necessary to have in mind in order to understand the implementation of the solutions for each of the problems. We can classify such methods in classical methods: those based on geometry and calculus ideas, and the more recent methods: the methods based on the machine learning approach.

2.1 Geometry Based Methods

The classical methods for computer vision are based on mathematical properties computed over the images, and these properties are able to transform the space of the pixels and detect interesting features.

2.1.1 Thresholding

Also called Image Binarization, thresholding is an image processing task or filter that we can use to obtain a Binary Image. We refer as binary image to an image with all its pixel values in binary form, each pixel is either black or white, often represented as 0 and 1 or as 0 and 255 depending on the underlying representation of the pixel value. It is important to note that thresholding is often used to perform basic image segmentation, but in our case we are not interested in segmentation but in edge detection, and that is a different but related process.

There are distinct ways of obtaining such binary image from a regular gray scale or intensity image. If our image is a color image, it must be first converted to a gray scale image. The two most prominent types of methods are Static Thresholding methods and the Adaptive Thresholding methods.

Static Thresholding

Static thresholding, sometimes called global thresholding or simply thresholding, is the most basic filter that can be applied to an image, it consists in picking a value v in the range of the pixel representation (e.g. from 0 to 255 if pixels are represented by byte), and all the pixels that are above the value will be replaced by the maximum value (e.g. 255), and all the pixels with value below the threshold are changed to zero. The only

problem in this simple approach is in how to choose v, two common options are: 1) setting a value that works for our experiments by trial and error, 2) performing a histogram and choosing a value that fits some criteria over the pixels. A classic implementation of this approach is the one taken by Otsu [43] in which the v is chosen to minimize the within-class variability and maximize the between-class variability. This approach can is effectively a Fischer Discriminant Analysis over the histogram of the image. There have been several published works, improving over Otsu's original algorithm, some researchers have tried to perform different statistical approaches, an example of these is the work by Kurita et al. [35] which threshold according to population mixture models. Other researchers have also tried to perform different types of histograms and operations, these works fall in the category of adaptive thresholding that we will explain next.

Adaptive Thresholding

The adaptive thresholding is also known as local thresholding, because this filter instead of fixing a threshold for the whole image, it analyses a small window or a pixel neighborhood, and sets an optimal threshold for such region or pixel, according to a certain procedure. The standard adaptive local thresholding sets a threshold T for each pixel (x,y), we may call this function T(x,y); the function will either compute an average or a weighted average of the NxN pixels in the neighborhood of the pixel (x,y). There are many other options however, instead of computing a T(x,y), we may compute a subset of such vector, and instead of obtaining a threshold value for each particular pixel, we may obtain values for windows of pixels. In the work of Jian et al. [31], the algorithm performs a 2D Thresholding that searches for an optimal value according to the entropy of the supposed foreground and background, and the output is the vector T(s,t), where (s,t) is a subset of (x,y). The value of T for the pixels that are not in the subset is equal to the value of the pixel that defines the subregion, such pixels are the pixels where (x,y) = (s,t).

2.1.2 Feature Detection

Feature detection is a term that encompasses many distinct methods for different tasks, however they all hold the same general idea: to detect a particular feature, a salient patch of the image that can be associated to a known geometrical figure or condition over a function space. Common examples is to detect lines, circles and corners; also it is very common to detect spots of maximum change in the image, according to a certain criterion such as the place which maximizes the ratio of the derivatives in both vertical and horizontal direction. We can see now that these algorithms perform very different computations, but they are however performing the same general task: finding the points of interest in the image.

Feature detection is usually within the first steps of the image processing pipeline for any

computer vision algorithm; the type of feature to be detected, as well as the method to detect such feature, will be chosen to give a useful input to the following steps in the pipeline.

Edge Detection

One of the most common types of features of interest are the edges of the objects in an image, the reason for this is simple: once we detect the edges we can match the found edges with the ones in the shapes we know and we can easily discard most of the candidates.

An example of the edge detection methods is the Hough transform[30], a classic method to find lines, circles and other geometrical features in images. This method was patented in 1962 by Paul Hough, but most modern implementations are based on the later refined version of Duda and Hart [16]. The Hough transform works by computing a new transformed feature space, and using this space to predict the existence of a geometrical figure like a line or circle.

It is also worthwhile of mention the Canny edge detector Canny [8], which is a method that improved the edge detection at its time and it is still regarded as a good option for many use cases. The Canny detector works by computing the intensity gradients of an image and then applying two binary thresholds over them, one high threshold and one low threshold. The high threshold removes most of the information, leaving only the most salient edges, and the low threshold will capture most of the edges in the image, but usually has a lot of spurious detections. The clever trick of the Canny detector, is that it combines the information of both thresholds, using a process called Hysteresis. In this process, the image obtained by the high binary threshold is used as a base, and it is refined by tracing the edges and filling the gaps they contain. The gaps are actually filled using the information from the low binary threshold, which usually retains much more edges and has less gaps. The image resulting of the Canny thresholding is a very good result, and it is an efficient algorithm despite its complexity, mostly thanks to its clever approach in most of the steps.

Similar to the edge detection, the contour extraction is a problem that helps us to identify the objects in an image by looking at its borders. However, the contour extraction usually works in binary images, and the problem is reduced to : is the pixel part of the image or part of the background?. The method by Suzuki et al. [53] is a classical method for contour extraction, and this method consists on tracing the contours by following the direction of its differences, assuming that this differences are the borders of an object within the image.

Blob Detection

One of the most general approaches to the feature detection, is the Blob Detection. In this approach the objective is to find any salience in the image, with the assumption that such salience must be part of an interesting object. Because its definition of the problem is very general, the solution proposed is also a very general, rather optimistic approach. Using gradient filters we are able to identify which pixels present the highest differences with respects to its neighborhood, then this pixels are analyzed using a heuristic approach to find if they could be useful to identify an object or not. This approach usually performs well in detecting objects with well defined contours and textures. Objects with fuzzy borders, or objects with very smooth surfaces are difficult to handle with gradient filters.

One of the most iconic methods in Computer Vision is the SIFT [40] method. The SIFT is actually a feature description method, that is applied on top of a feature detection. In their work, Lowe [40] propose to perform the feature detection by applying a Laplacian of Gaussian (LoG) filter; this filter will compute the most salient pixels in the neighborhood of the pixel, while simultaneously smoothing the image with Gaussian blur. However this operation can be costly, and because we would like to detect features in neighborhoods of different sizes, the cost of applying the filter multiple times can become too much for the method to be convenient. For this later reason, the Difference of Gaussians (DoG) is introduced as an approximation to the LoG. The DoG will detect features by taking the difference of the same image blurred with Gaussian filters of different sizes. The result of the DoG is a very good approximation of the LoG at a fraction of its original cost, this will come very handy when performing analysis of many images or videos.

SURF [5] is an alternative proposal to the SIFT method. The SURF is also a feature descriptor, and uses the gradient filters as the first step in the method. Bay et al. [5] proposed the Determinant of Hessians (DoH) instead of the LoG for the filters, mainly because it is able to retain similar information but is more robust to spurious detections. Like the LoG, the DoH itself is expensive to compute, to alleviate this problem instead of computing the DoH filter, the computation is approximated by using binary filters, similar to the Haar-Like features. And because of this approximation is done, the filter itself is very inexpensive to compute at different scales; this is an advantage over the SIFT method, that scales the image instead of the filter.

2.1.3 Feature Description and Matching

After detecting the interesting features of an image we would like to store them, in compact yet meaningful representations that would allow us to compare in an efficient manner two different images. These compact representations are called Feature Descriptors, and they are the backbone of many computer vision methods. Some of the most relevant feature descriptors are: SIFT[40], SURF[5] and HOG[12]. Although they are quite different, they all share the same core idea: the features of the image are obtained

by gradients, and this gradients are then put into bins that will represent a neighborhood of pixels. The main difference between these techniques is that in the case of SIFT and SURF, only descriptors of salient pixels are computed, while in HoG the descriptors are bins computed over the whole image.

With the feature descriptors at hand, we can now compare new images and find instances of our objects. This task is called feature matching, and the idea is that if our object is present in both images, then some of the feature descriptors must be quite similar between both. However we need to realize that is not sufficient one feature descriptor to match an object, we need many feature descriptors to coincide in order to recognize an object. The RANdom Sample Consensus or RANSAC, was introduced by Fischler and Bolles [19] as a general technique to identify patterns and perform regression. Using this technique we can also perform feature matching, it is a very simple approach that allows us to match objects between the two images, and it also tells us how these objects moved or changed between them. After applying a feature detector and descriptor algorithm over our images, we are left with a "cloud" of descriptors attached to certain pixels, we will call them keypoints, then we follow these steps iteratively:

1. Pick a random subset of keypoints from the first image, and we search in the second image the keypoints with the closest descriptors.

2. Fit a model of how these keypoints moved from one image to the next, i.e. a homography

3. Compare again the homography with the set of points and only retain the points that match within this model below a certain threshold, we call these points the inliers.

The process stops iterating when we have enough inliers, and the final step is to re-estimate the model by using a least squares optimization. The RANSAC matching works under the assumption that the images we are comparing are close enough, so picking out random points will often luckly result in a good match. However, finding a good solution is not guaranteed, after all is a random process.

RANSAC it is essentially a statistical method, which also hints us to another option: if instead of simply perform matching of the features we use the descriptors to train more sophisticated algorithms, machine learning algorithms that are good for regression or classification tasks, e.g. SVMs or NNs; this would give us more flexibility than the matching. This is the basis of many methods to perform the object classification task, this is an interesting task, but we are only interested in classification methods if they helps us to perform object detection and classification. One example of the interesting methods is the one proposed by Dalal and Triggs [12], use these classifiers to perform detection, in their case pedestrian detection. But we need to clarify that, to develop a multi-class object classifier is outside of the scope of this work.

2.2 Machine Learning Based Methods

Most references in this section will be addressed to the book "The elements of statistical learning" by Friedman et al. [21], which has a very good and in depth description of each of the machine learning methods.

2.2.1 Least Square Optimization Methods

The least squares methods are a family of optimization methods, which objective is to minimize the sum of the squares of the errors obtained by a set of equations. These methods can be used to obtain an approximate solution for overdetermined systems; these types of systems are almost always inconsistent, and because they have no solution, an approximation is often the best we can expect to obtain.

There are many least square methods, but among them the most relevant to our work are the following:

- Gradient descent: it is an iterative optimization method, and it uses the gradients of the function at each point to move over the error surface in the direction that minimizes the error. Because it is ignoring most of the surface error of function, except the current point, it often only finds a local minimum, and not the global.

- Gauss-Newton: the method was proposed by Björck [6] in 1966, and it is like the gradient descent an iterative method that finds a local minimum for the error of a function. It has the advantage to have a more clever way to figure out the direction that minimizes the error taking into account not only the gradient but also the jacobian matrix, and this allows the method to converge much faster. The main disadvantage of the method is that for extreme starting points it is often unable to find a solution.

- Levenberg-Mardquart: this method was developed independently by Levenberg [38] and Marquardt [41], and it can be viewed as an interpolation between the gradient descent and the gauss-newton methods. Its main advantage is that it is more robust to bad initializations than the gauss-newton, and it converges much faster to a solution than the gradient descent.

The least square methods are extensively used in machine learning, and they have two main uses:

- Train other machine learning models, because in many models computing the parameters that minimize the errors from the data is either impossible or too costly, we can perform an iteration to progressively adjust the parameters minimizing the errors. For example, the NN are often trained using one of this methods to update its weights in each step of the back-propagation algorithm.

- Refine solutions, often we obtain a rough solution for a problem, e.g. when performing feature matching, we obtain a model with values that are roughly close

to the desired solution. In this cases we can optimize the solution using a least square method, minimizing the fitting error of the matches.

2.2.2 Boosting

Boosting is an ensemble meta-algorithm, which means that is a method that takes another machine learning algorithms and use them in an ensemble fashion to compute its results. The idea behind boosting is that if we can combine the solutions of many weak learners, i.e. algorithms that can perform an acceptable job but are not the best, they combined answer can be as good as a strong algorithm, and it often has the advantage that computing with such weak algorithms can be much faster than the strong algorithm. A more in depth explanation about boosting can be found in chapter 10 of the book[21].

One of the most successful boosting algorithms is the AdaBoost, proposed by Freund et al. [20], and they won the Gödel Prize in 2003 for their work. The name AdaBoost means Adaptive Boosting, and it dues its name to the fact that it can adapt by adding new weak learners to the ensemble, further reducing the error of the overall classification.

2.2.3 Decision Trees

A decision tree is essentially a binary tree that represent a set of partitions in the space of a dataset. The partitions will allow us to compute statistics that describe the dataset in terms of the features of the data, and this will allow us to answer questions, essentially classifying any arbitrary point in that space. The name decision tree comes from the fact that it helps us to take decisions, because we can "ask" the model about new observations and it will allow us to easily classify it, depending on the partition it lands in. We can find an through explanation of the decision trees in the book[21].

2.2.4 Random Forests

The decision trees are good learners, but they can easily overfit the data if they are allowed to grow to much, and because it is not trivial to tell how much to let grow a tree, if we limit the tree too soon it can be very biased to, at least in some parts of the space. To overcome this problem Breiman [7] proposed the random forests. The random forests are essentially an ensemble of trees, which combine their outputs to produce a good answer. The random forests use the idea from Ho [27] to pick random features of the dataset, and with these features they train an ensemble of decision trees, but each decision tree is not trained with the whole dataset either, they are trained with a random subset of samples, and this is called Bagging. The random forests have been proven to be very robust models, that provide very good classification and regression results, provided that we have a dataset with good features and well constructed. More information about random forests can be found in the chapter 15 of [21].

2.2.5 Artificial Neural Networks

The artificial neural networks are models inspired by the biological neural networks, in the central nervous systems of animals. These models can be seen as a directed graph in which the nodes are the neurons and they exchange messages through the connecting edges; this connections have weights which influence the strength of the signal sent by the neurons, and when a neuron receives the signal it computes its activation function which determines the signal to be sent to the next neuron. The neurons are organized in layers, that is arrays of neurons that interconnect with each other in different patterns, and depending on these patterns we define the network architecture. Some basic network patterns are:

- Fully connected layer: each neuron of the layer connects with each neuron of the next layer.
- Convolutional layer: each neuron of the layer connects with a an adjacent, and often small group of neurons in the next layer.
- Recurrent layer: the neurons can connect to one or more neurons of the next layer, but also the neurons have looping connections to themselves, which creates a feedback of the output of its activation.

Each of these architectures are good at performing different tasks, and although there is no formal definition, we can give a rough guideline for its range of applications:

- Fully connected networks are good for classification and regression given a compact set of features. They are a general method comparable to other classifiers.
- Convolutional Neural Networks are good for extracting features and finding spatial correlations in the dataset. For this reason they are often applied to analyze images.
- Recurrent Networks are good for applications that have a temporal dimension, they are good and often used to find correlations and patterns in time series.

In chapter 11 of the book [21] we can find an extensive explanation on how NN work and how they are built.

2.2.6 Support Vector Machines

A support vector machine is essentially a linear binary classifier, and what it does is to fit an hyperplane that splits the classes in the dataset into two partitions of the space. The hyperplane that splits the space has dimension of the number of features of the dataset minus 1; this means that if the dataset is 2d, the split is done by a simple line, if the dataset is 3d it is a plane, and so on. To fit this hyperplane, we only need to find some examples that are "on the borderline" between the classes, these are called the support vectors and are what gives the name to the model. Because the model is so simple it is

extremely efficient computationally which makes it very useful for many tasks.

With a clever modification known as the kernel trick the SVM can be extended to perform non-linear classification, which is essential to most of the interesting problems in real tasks. The kernel trick essentially warps the original space of the dataset, extending its dimensions, and this allows to perform again a linear classification but in a warped space. This allows the SVM to overcome its only linear limitation, and be able to compete with NN and random forests as a general classifier.

There are many modifications and extensions of the SVM method, they can be modified to perform regression, or to give a confidence measurement along with the output class, etc. These extensions are widely used in many applications, including computer vision applications, e.g. Dalal and Triggs [12] use SVMs in conjunction with their HoG descriptor to perform object detection and classification.

2.3 Detection and Recognition

Some common scenarios on the field of Computer Vision are : Detecting pedestrians walking in front of the car, detecting the faces of the people in the picture, identifying the people in the picture, spotting defects on an assembly line, detect dogs, detect cars, etc. In each of these scenarios the specific task being performed is quite different, however they can be expressed in terms of two common problems:

- To detect whether there is a certain type of object in the picture, and where is it. We refer to this as object detection.
- To recognize or identify the object within a region of interest (ROI) or window of the image. We refer to this as object recognition.

Actually the terms detection and recognition have definitions that overlap and depending on the author it may change its meaning. Within this work we will refer to the terms as defined in this section, to maintain a coherence in the discourse through the different chapters. These definitions may differ however, when compared to those in the literature or in the cited references.

3 Technical Framework

3.1 Flying Robot Platform

The main platform proposed for this work is the platform from the Robot-Int-Coop project; that platform is still under development and it is called Ona.

Ona[1] is a quadrotor platform, designed for Human Robot Interaction (HRI) tasks, and specially for urban and indoor environments interaction; because of this, the platform body is protected to minimize possible harm to the users, and also the robot counts on several sensors such as cameras and microphone to perform the interaction tasks which usually involve detection and recognition over images and audio.

These are the most relevant hardware features of the platform, the ones we take into account when developing the software for it:

- **Size:** Max diameter: 768mm, Diameter between motor axes: 490mm, Height: 271mm.
- **Load:** Platform weight: 1600g, EPP Protection weight 200g, Useful load: 550g, Total weight: 2350g.
- **Processing:** Pixhawk 3DR controller, Onboard computer running Linux Ubuntu 14.04 Server and ROS Indigo, External laptop running Linux Ubuntu 14.04 Desktop, ROS Indigo and OpenCV software.
- **Communications:** RC Futaba 2.4GHz, Telemetry Xbee 2.4Ghz, Wireless 2.4GHz, GPS.

Also the platform will have a Google Nexus 5 [2] smartphone that can be optionally attached to it, letting the user interact through the touchscreen of the phone, the touchscreen serving as both input and output device, and additionally the phone has two extra cameras, microphone and other sensors that can be used.

3.2 Robot Operating System

The Robot Operating System (ROS)[2], is a framework to build software for robots, which means that is essentially a set of tools that help to develop, debug and interconnect

[1] `http://wiki.iri.upc.edu/index.php/Ona`
[2] `http://www.gsmarena.com/lg_nexus_5-5705.php`

Onboard Computer	Kontron pITX-E38
CPU	Intel® Atom™ E38xx SoC 4 cores
RAM	4GB
Controller	3DR Pixhawk
Battery	2x LiPo 3S1P 5300mAh, 30C
Transmitter	Tx Futaba T8J, 2.4GHz
Motors	4x T-Motor MN3110 780KV
ESCs	4x T-Motor ESC 20A
Propellers	4x Graupner 11" x5
IMU	MPU 6000
Optical Flow	PX4Flow
Gyroscope	ST Micro L3GD20, 3axis 16bits
Compass	ST Micro LSM303D, 3axis 14bits
Barometer	MEAS MS561
Telemetry	Radio Xbee Pro S1, 2.4GHz
GPS	uBlox LEA-6 3DR with compass

Table 3.1: Ona Platform Components

the software modules that conform the system of the robotic platform. Usually a robot has several modules, specialized to perform different tasks, and have they interact in different ways with the hardware. We can find that usually a simple robot has at least the following modules:

- A control module to interact with the motor or motors.
- A module that reads the input of each of the sensors.
- A planning module to plan the next actions of the robot.
- A navigation module, that allows the robot to map and navigate its environment.

And usually, they also have many other modules that are also needed to process the inputs and transform the signals into useful data for the planning. As we can see from this small example, even the simplest of the robots is a complex software system that requires a lot of interacting modules to accomplish its task.

3.2.1 ROS Architecture

ROS has a network oriented architecture, in which we can develop distributed applications that deploy spread between several devices.

The most basic part of the ROS architecture is the node, a node is and independent program designed to accomplish a specific task, and according to such task we can categorize the nodes into two broad groups:

1. **Algorithm Node:** as it name implies, an algorithm node is used to execute an algorithm, or put in other words to perform an interesting computation or carry out a process. The algorithm node does not interact with the hardware, it takes only messages and other sources as input, and publishes messages again as result.

2. **Driver Node:** a driver node has as main goal to interact with the hardware, such hardware may be a sensor or an actuator, e.g. a camera or a servo. Usually these nodes do not perform complex computations, and do not perform bidirectional messaging, instead these nodes read data from sensors and only formats it and publish them as messages, or read command messages and translate them to low level commands to interact with the actuators API.

A node or a set of these nodes can be seen as the conceptual module that we described earlier in this section, e.g. can define the control module, as a set that contains the different driver nodes that control the motors; or the planning module with the planning node as its sole member.

In the center of the architecture we have the ROS core, which coordinates all the communication between the different nodes, this communication between nodes can be seen as edges that connect the nodes, therefore we obtain a graph that can be visualized, a feature that is very useful to understand the communication of the distributed system.

The communication between the nodes is carried out in a messaging fashion, following the publish-subscribe pattern. Essentially, Nodes subscribe to "topics" that are either required or interesting to perform its task, and after such task is accomplished the result is published into another topic so other nodes can use the result of its computation.

3.2.2 ROS Messages

For all the methods in this work the output is sent as a transform message, which is the standard in ROS. Message 3.1 is the definition[3] of the transform message, the code was styled as JSON to aid the visualization of the message structure. As we can infer from its structure, the Transform message has two main sub structures, translation and rotation. These structures represent the location of an object in the world, with respect of a frame of reference.

It is important to mention that if we are detecting multiple objects, then the message sent will be an array of transforms instead of a single transform message. Also the

[3] Original definition can be found at: `http://docs.ros.org/jade/api/geometry_msgs/html/msg/TransformStamped.html`

LabRobotica version of the ar_pose method, that will be described in chapter 4, uses a message that includes the transform message but it also includes some additional information, such as the id of the marker detected and the confidence of such detection.

```
// This expresses a transform from
// coordinate frame header.frame\_id to
// the coordinate frame child\_frame\_id

// Message header
Header header
// The frame id of the child frame
string child_frame_id
// This represents the transform between
// two coordinate frames in free space.
Transform transform : {  // This represents a vector in free space.
                         // It is only meant to represent a direction.
                         Vector3 translation: { float64 x
                                                float64 y
                                                float64 z
                                              }
                         // This represents an orientation
                         // in free space in quaternion form.
                         Quaternion rotation: { float64 x
                                                float64 y
                                                float64 z
                                                float64 w
                                              }
                      }
```

Message 3.1: Transform

3.2.3 LabRobotica Philosophy

This work was developed in the Mobile Robotics Laboratory (LabRobotica) of IRI, and because of that all the software was developed by following the standards of the Laboratory. Such standards are embodied by the LabRobotica philosophy, by philosophy we refer to the tools and methodology used in the laboratory to develop the software used by all the robots in the lab. This philosophy promotes efficiency, reusability, maintenanbility and modularity of the software developed, and it also helps to spread the good programming practices and design patterns within the C++ and ROS environments.

LabRobotica philosophy has two basic toolboxes to help the researchers and developers:

- Scripts: the framework has a series of scripts that help in the creation of the necessary source code and configuration files, for the robot software projects. It also helps the developers to compile the projects and integrate them with other projects developed in the laboratory.
- LabRobotica library: along with the scripts there are available several source code

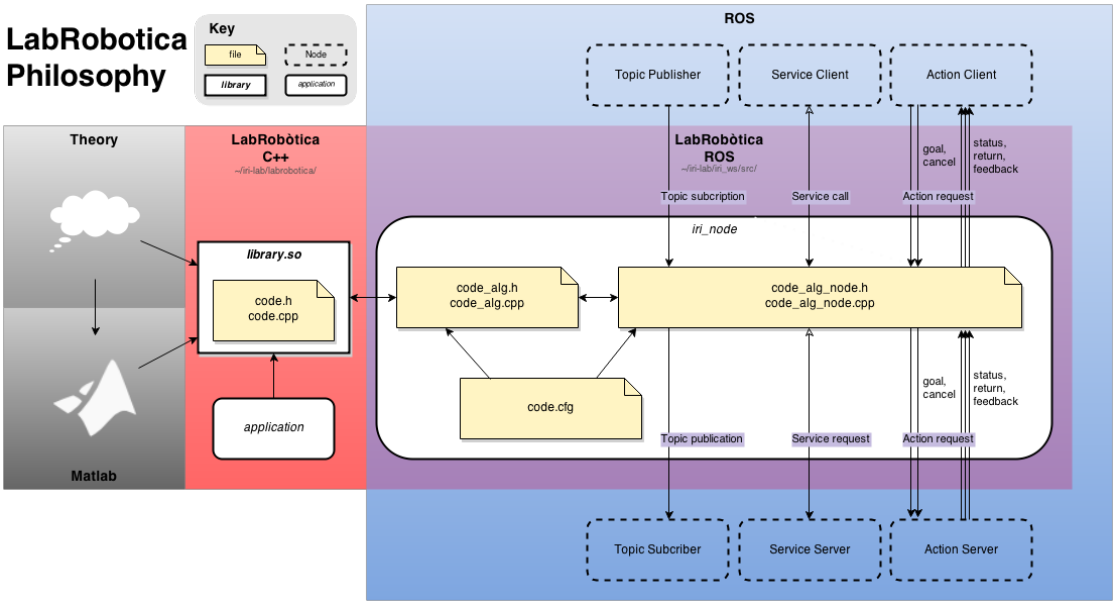

Figure 3.1: LabRobotica Philosophy Diagram

files in the form of libraries, for both C++ and ROS. These libraries facilitate the development of software for the robots, because they are efficient implementations of many common tasks following software design patterns that allow for a structured development. The code libraries are maintained in a common repository for the ease of use, and ease of maintenance.

As we can observe in the figure 3.1, the LabRobotica philosophy has the following workflow:

1. The theory and ideas are is first coded and tested as algorithms in matlab. This gives flexibility and swiftness to try new prototypes.

2. The working prototypes are translated to efficient C++ code. And this code is deployed as a software library in the OS.

3. An algorithm node is developed to use the software library. This node should be configurable through the standard ".cfg" file. And the node has two main classes: the "code_alg_node" class, which has all the threading and communication logic; and the "code_alg" class that interacts with the library and performs any other required task for the core algorithm to work.

3.2.4 Platform deployment

For our work there are two ways of deploying the modules[4] developed:

[4]Module means a set of nodes in this context.

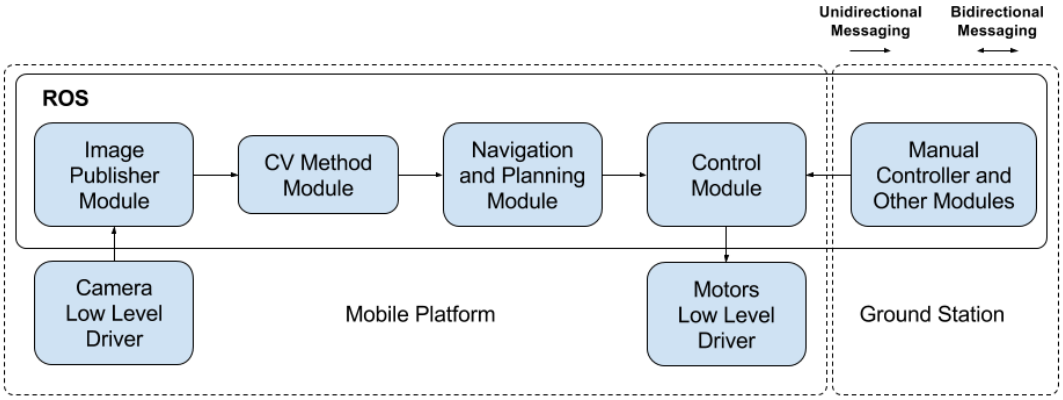

Figure 3.2: ROS architecture for Ona platform and final deployment

- Final deployment: is the deployment we have designed our platform to work on. Is the ideal scenario, in which the flying platform will have all the computer vision nodes running inside and interacting directly with the other nodes of the platform. This scenario is the one in figure 3.2.
- Test deployment: is the one we can see in Figure 3.3. It is different from the final deployment mainly because the test platform is incapable of deploying ROS nodes on its own.

During this work we will only work with the test platform, because the Ona platform is still in development and it is not ready to start the deployment of ROS nodes. The test platform is an AR.Drone 2 platform from the Parrot company[5]

3.2.5 ROS on Android Devices

For this work we also used an android device, the device can provide the images from its camera, and it also can serve as a medium for the interaction with the robot.

To develop the Android software it was necessary to use the ROS android library[6], which is based in the ROS Java library[7]. Essentially the ROS Android software are Android applications that run nodes of ROS as native software on the Android device. This allows to maintain the communication with the robot using the messaging capabilities of ROS. And also it allows to use any of the computer vision nodes developed for the robot, with the smartphone camera.

[5] `http://ardrone2.parrot.com/`
[6] `http://wiki.ros.org/android?distro=indigo`
[7] `https://github.com/rosjava/rosjava_core`

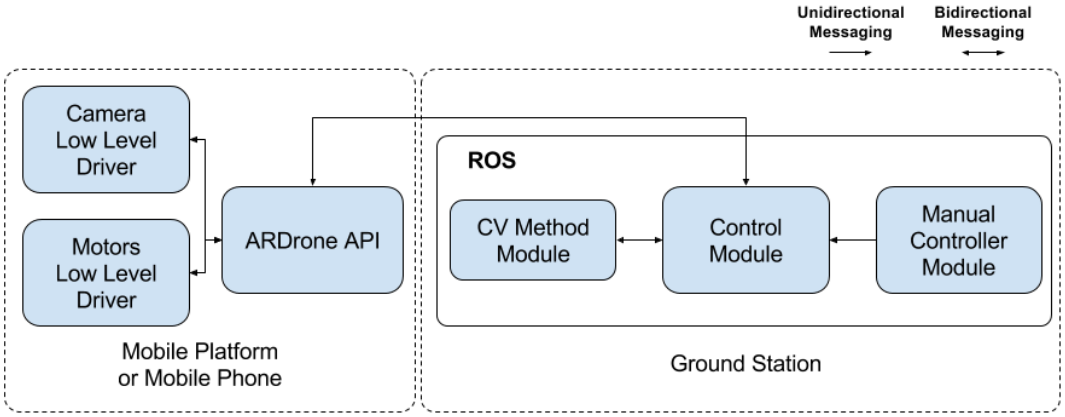

Figure 3.3: ROS architecture for test platform and test deployment

Also is important to mention that to develop the ROS android nodes is necessary to use the Android SDK and the Android Studio IDE[8].

3.3 OpenCV

The OpenCV [1] library was used extensively in this work, it was used to process the images and implement the different methods.

OpenCV is an open source library that has implemented over 2500 algorithms for computer vision, image processing and machine learning; it is for many tasks standard tool of the industry, and it is widely used in conjunction with ROS to solve computer vision problems in robotics.

3.4 Experimental Guidelines

The experiments were designed to help us determine if the chosen algorithms meet some desired properties for the proposed scenarios in the project, such properties are:

- Stability of detector accuracy under reasonable noisy images. We know that the images we will use will be noisy because the following reasons: small camera in the flying platform does not produce perfect images; the movement and specially the shaking of the flying platform will produce motion blur in the images, even if the camera is optically stabilized. Nevertheless, we would like algorithms that are robust enough to cope with these sources of noise.

[8] `http://developer.android.com/sdk/index.html`

- Predictability of detector resource consumption. Some algorithms consume different amount of resources depending on the input image, and because we know our platform will be moving and capturing images that can be very different from each other, we need to be able to estimate what is the resource consumption of the algorithm, even in the worst case scenario.

- Enough operating range. The flying robot main advantage is its mobility, but it also poses a huge challenge to the computer vision algorithms, because most of these algorithms work under the assumption than the point of view of the camera is fixed. We need that the algorithms developed work within enough distance, i.e. covering the longitude of the test bed; and also the algorithms should work with enough variation in the angle of the point of view of the camera, to prevent losing the detection of the objects it is currently interacting with.

To determine such properties the general guidelines for the experiments are:

- Simulate different flight conditions of the platform: scenarios such as slow and fast flight, steady and turbulent flight, are essential to determine if the algorithm will withstand during the real tasks. The goal of these simulations is to determine under which conditions the detection algorithms fail, and produce false positives.

- Simulate extreme ranges of operation: we can separate from the object, and determine the maximum working distance of the algorithm. Also rotate around the object, changing the angle of view. The goal ot these simulations is to determine which is the maximum range we can expect the algorithm to work properly.

- Simulate noisy inputs: by putting the camera in front of a cluttered scenario, and trying to produce motion blur, etc, we can asses the stability of the algorithms, in terms of resource consumption, false detections, lose in accuracy, etc. A noisy scenario and a noisy image can usually make many methods produce. The goal of these simulations is to confuse the algorithm and determine the false positives in a worst case scenario.

4 Visual Marker Recognition

The visual marker recognition is today considered a basic or classic approach to the object recognition and pose estimation problem; it is so, because it was one of the first successful approaches in computer vision, and allowed researchers to perform augmented reality tasks in the late 1990s, with very limited hardware capabilities, slow CPUs and noisy digital cameras, and also with less software available to build their solutions. One of the most cited authors is Rekimoto [47], who was one of the first to successfully complete the augmented reality task of putting 3D objects within a real scene that is being captured by a camera. In his work Rekimoto used fiducial markers that allowed him to locate the spot in the scene where the 3D object should be rendered, and also the marker gives the crucial information about the scale and the pose such 3D object should have.

In this approach, instead of detecting any object in the real world, a marker is specifically designed and placed in the scenario, and because it is a simple object (e.g. a square) with known features, it is very easy to identify the target or marker in the image, and also it is inexpensive to compute an estimation of its translation and rotation coordinates with respect to the camera. To summarize, these two are the main reasons why it is used as a baseline approach: it is reliable, at least under laboratory conditions, and it is efficient in computing resources.

Marker recognition algorithms are so efficient that most augmented reality applications for mobile devices are built using them, and they are used to process the video from the camera in almost real time.

It is important to note that marker recognition methods perform both marker detection and marker recognition, because they are able to tell where is the marker in the image and which is the specific marker. Even more, the marker recognition algorithms usually even perform pose estimation, telling where and how is located the marker with respect to a model of the world. This is usually more information than most object detectors or recognizers can tell.

4.1 Method

There are many variants to the method, but it is always roughly the same idea being implemented by using different algorithms. In this section we will describe a general method that describe the idea of performing marker recognition. This method can be

then instantiated to a particular implementation.

The following steps illustrate the general sequence that marker algorithms follow:

- **Thresholding:** the Thresholding is the first step in most marker recognition algorithms, it is so because later we will perform edge detection, and most edge detection algorithms require a binary image.
- **Edge Detection:** given the binary image some kind of edge detection or contour extraction will be performed. This step will output a set of candidate edges or contours, that may be markers in the image.
- **Marker Matching:** In this step, the algorithm will match the candidate either by comparing each candidate iteratively with each known model, or extracting information from the image itself to match the marker directly. Also, in this step usually a lot of candidates are discarded by using heuristic tests such as closure of the shape, size and aspect ratio of the contour; these tests speedup the matching process.
- **Pose Estimation:** finally we estimate the position and orientation of the marker in the space, this can be done via either regression algorithms or RANSAC.

4.1.1 ARToolkit and ar_pose

The Augmented Reality Toolkit (ARToolkit) is an open source library for Augmented Reality tasks created by Kato and Billinghurst [33], and it is specially popular to perform marker recognition.

ARToolkit implements the general method in the following way:

- **Thresholding:** standard static thresholding.
- **Edge Detection:** line segment detection, standard. Line intersections are searched for as possible corner candidates. If a closed contour is found joining four line segments, it is saved as a candidate.
- **Marker Matching:** the known square markers templates are projected to the image rotation and translation, and the regions are then normalized and sampled to match with the projections.
- **Pose Estimation:** when the markers are matched, an implicit estimation of the pose is already known, that is the same as the coordinates of the corners in the contour extracted. However, the pose is refined by optimizing with the Levenberg-Marquardt algorithm the rotation components of the pose.

The ARToolkit method for markers is similar to the previous work by Stricker et al. [52], but in this case the image is not thresholded, instead two gradients "White-to-Dark"

and "Dark-to-White" are searched for to identify the rectangle candidates; this difference makes the ARToolkit more streamlined and potentially more efficient computationally, since many possible region candidates can be discarded by the thresholding.

ar_pose arp [3] is a ROS package that allows us to perform marker recognition and pose estimation using the ARToolkit library. This package has modules for single and multiple marker recognition; such modules take a standard ROS camera topic as input and publish a topic with the information of the detected markers translation and rotation with respect to the camera, this topic is in an standard spatial transform matrix. Also, it can perform the reverse task: estimate the camera position with respect to fixed markers with known coordinates; this later task is specially useful to estimate a mobile robot position in the world. ar_pose uses exclusively binary matrix markers, although this is not a restriction for the ARToolkit, it helps to simplify the process and avoid errors.

4.1.2 ArUco

Another take on the marker problem is the library ArUco, a work by Garrido-Jurado et al. [22] from the *Universidad de Cordoba*. It is again heavily inspired by the ARToolkit, but has some important differences that update the method, making it more robust without losing performance.

The ArUco method can be summed up in the following steps:

- **Thresholding:** it uses adaptive thresholding by default, and also can use Canny edge detector as thresholding algorithm.
- **Edge Detection:** contours are directly extracted from the binary images using Suzuki et al. [53] method.
- **Marker Matching:** markers are matched by sampling the regions inside the matrix marker, each pixel inside the region is counted as a vote, and the majority decides whether the cell is black or white.
- **Pose Estimation:** This final step takes exactly the same approach as ARToolkit, corners are detected by the line intersections and the pose is refined by taking the corners and performing a Levenberg-Marquardt optimization.

Although ArUco seemed to be a very good candidate and performed very well during the initial tests, we are forced to discard it because it is incompatible with the ARToolkit Markers, which are the LabRobotica standard and are being used in all the other projects that require markers. It is important to notice that this work should also be compatible with the other projects, although its integration is out of the scope.

4.1.3 bar_pose

bar_pose is a ROS module developed in the IRI, and it is the implementation part of the work by Amor-Martinez et al. [4]. This module is inspired in the ar_pose module and it is designed to perform recognition of solid 3D bars with the aid of markers.

It performs adaptive thresholding, followed by a standard contour extraction algorithm [53]. However in the matching step it is completely different from the other methods previously mentioned, it does not perform the match by sampling the pixel values, instead it relies on the shape of the marker, and it performs a Gauss-Newton optimization to match the contour candidates with the templates. This gives the bar_pose another level of complexity beyond classic visual markers, because it is not working with a marker, essentially it can match any contour of any shape. In this work however we do not exploit this characteristic.

Finally, the method performs a pnp regression using the algorithm by Lepetit et al. [37], which refines the estimate of the pose and gives a very accurate output. Usually, the pnp algorithms can be very costly because its optimization requires to iterate until convergence, however, because in bar_pose the algorithm is initialized very close to the solution very few iterations are required for the algorithm to converge to an excellent solution. This step also gives the method another flexibility, because it was designed to work with 3D objects, specifically bars, and the method can perform the pose regression to any 3D model, and this is different from the other marker algorithms that can only estimate the pose of a square maker.

The performance of the module is expected to be good, of course that of the ARToolKit should be faster because it is a simpler approach. However the accuracy of the detection is expected to be higher also because is a more complex model.

4.1.4 Marker Tracking

One of the main shortcomings of the bar_pose method is that its matching step can take a long time if a large number of contours is extracted from the image. This makes it slow to compute when the scene is cluttered.
To improve the performance of the algorithm, a tracking algorithm was implemented. The algorithm tracks the movement of the marker in the image, by remembering the previous positions of the marker and using a linear velocity function to estimate its next position. The function is formally defined as follows:

$$Pos_i(x, y) = Pos_{i-1}(x, y) + (Pos_{i-1}(x, y) - Pos_{i-2}(x, y)) \tag{4.1}$$

The formula 4.1 can be interpreted as, the position of the marker on the frame i, is estimated to be the position of the marker in the frame i-1 plus the velocity it had at frame i-1, which is the difference of the position of the marker on frame i-1 and i-2.

After estimating the next position of the marker, the contour detection algorithm is only given the area surrounding the prediction, this means that the contour detection will be much faster and that the detected contours will also be useful contours to match, speeding up the whole algorithm.

There is only one caveat in this method, and it is: what happens if the marker we want to detect was not detected in the previous frame, then this area would never be searched for contours and the marker would never be detected. The solution to this caveat is very simple: every n number of frames we allow the detector to scan the whole image, then we perform tracking on the next frames. This is a practical solution that works under the assumption that in the worst case scenario, we are discarding a marker for a small amount of time. For example if our algorithm works at 20 fps, and we perform the whole image scan every 5 frames, then in the worst case we are losing a possible detection for 1/4 of a second, and the average case is that we lose the detection of the marker for 1/8 of a second. These are very reasonable numbers that we can deal with, and therefor this solution seems to work out on the theory.

4.1.5 Color Thresholding

When testing the marker algorithms inside the laboratory light conditions are always controlled and ideal, therefore the results of the computer vision algorithms are stable. But when testing outside the lab light conditions change, and specially under the sunlight this can be a huge problem, because the sun is a strong light source and it can produce strong contrasts and shadows in the images.

Shadows are difficult to handle in marker recognition algorithms, because these algorithms often use only the intensity of the pixel even if the input image is in color; this problem is specifically found in the thresholding algorithm, and that is the first step for the marker detector. In the thresholding step shadows are often interpreted as black regions that get mixed with the real black regions, distorting the contours of the markers in the picture, and recovering the shape of the markers by the use of morphological operators is not possible in most of the cases.

There is another way to perform the thresholding however, and use more information that is already available in the input. That other way is to perform the thresholding by using the information from the color channels, this is not a straight forward task from the usual color image formats RGB or RBG, because the intensity information is mixed with the color information in all three channels, and to segment a specific color with different intensities, a dynamic thresholding would be needed over the different channels. However, other color spaces exist that make the intensity analysis more clear:

- **YUV:** The YUV space was designed for TV, and is still used within many industry standards such as PAL and SECAM. In this color space, the luminance or intensity

component is the "Y" Channel, while the "U" and "V" channels represent the color information, more concretely these channels encode the color differences. YUV is very efficient and robust for image transport, and makes it easier to perform a thresholding over the color channels, it would only require to set two static thresholds, one over the "U" and other over the "V". However, it can be difficult to define a color in human terms within this framework, because the "U" and "V" are not colors but differences, so moving a threshold manually within this space is not intuitive.

- **HSI and HSV:** The HSI and HSV color spaces are very similar, and both were defined for graphic design applications. In these color spaces the color is uniquely defined by the "H" component and the H is a circular space; this fact makes it very easy and intuitive to set a threshold of color for a human, a specific range of colors is defined by selecting the "H". The "S" is the saturation channel, which encodes how much of the color has the pixel, finally the "I" encodes the intensity or luminance[1] and goes from black to white. The "V" of the HSV space is a concept close to the "I", but its range goes from black to "gray" or the pure color represented by the "H" channel.
- **Chromaticity:** The chromaticity is an space of color that discards all the intensity information, leaving only the hue and the saturation components. The saturation is also often called chroma, from which this space derives its name. Also there is the variant of rg-chromaticity, which is essentially the same idea but it derives from the RGB space.

There are several works and methods to perform thresholding using the color information, Horprasert et al. [29] use the color information, combined with some statistical concepts to perform robust segmentation of the foreground and background, discarding the shadowed pixels that can be confused with the foreground object or person. Cucchiara et al. [11] detected moving objects in their work, but to clean the spurious detections they implemented a color thresholding that allows them to tell which are the real moving objects and which pixels are only shadows.

Color thresholding has also being used to detect and segment shadows in aerial images. With such goal Tsai [56] proposed to perform a simple computation of this ratio: $(H(x,y)+1)/(I(x,y)+1)$ where $H(x,y)$ is the Hue at the pixel (x,y) and $I(x,y)$ is the Intensity or Luminance of that same pixel; over that resulting image, an Otsu threshold is performed to obtain the candidate shadowed pixels. The reason to perform such ratio and thresholding, is because according to Tsai [56] the aerial images have the following properties:

1. "lower luminance (intensity) because the EMR from the Sun is obstructed;"
2. "higher saturation with short blue-violet wavelength due to atmospheric Rayleigh scattering effect [2], [21];"

[1] for this reason sometimes HSI is called HSL

3. "increased hue values because the change of intensity of an area when shadowed and not shadowed is positive proportional to the wavelength [12]."

Highly based on Tsai [56] work, Chung et al. [10] proposed a new improved ratio to perform the thresholding, and after that two more steps:

1. **Local thresholding:** after the global thresholding, the pixels believed to be part of a shadow are submitted to a connected component analysis, this will give us connected regions of the image and within such regions a local thresholding is performed.

2. **Fine shadow determination process:** within the regions, the pixels are analyzed to hold the following properties:
   - "Property 1: True shadow pixels usually have lower intensity values than that of the neighboring nonshadow pixels, but both of their chromaticity values are similar."
   - "Property 2: For the color aerial images, most true shadow pixels are connected."

Our problem is different to the ones solved in previous works because our images are not still (contrary to [29]), nor have a static point of view (contrary to [11]); also, they will be taken from an aerial platform, but they are relatively much closer to the target and its surroundings than the aerial images taken from aircraft or satellites, thus, they do not have many properties described in [56]. However we still can use some ideas from these previous works as inspiration for our solution. Two essential facts are still valid for our problem:

1. The Intensity values change largely between shadowed and non shadowed areas, but their Hue values remain almost constant, especially when the image is taken by a camera at a relatively short distance to the subject, e.g. less than 10 meters distance.

2. The neighboring pixels with similar Hue values are very likely to be part of the same object.

Also it is important to remark that the problem that we are trying to address is for the outdoors experiments scenario, and in this scenario we have strong light sources, very hard contrast and hard shadows, and the colors are easily recognizable. For other scenarios where the light sources are weak and the contrast is poor, this method would not help to segment out the shadows.

4.2 Results

In this section we analyze the results of the experiments for the ar_pose and bar_pose nodes described in the previous section. For this task we designed a set of experiments that assess the performance and accuracy of the marker recognition algorithms.

4.2.1 Experimental Setup

Following the guidelines in section 3.4 we propose to perform the following experiments with the test platform:

- Record five different markers in sessions of approximately one minute. The recorded sessions must include slow and fast movements including pan and rotation movements.
- Record the same five markers in sessions of simulated high turbulence, obtaining a high camera shake.
- Record again the five markers, but this time in sessions that slowly separate from the target, to estimate the maximum detection distance. And the analogous experiment but with the angle towards the marker.
- Record a high cluttered scene, but without any visible markers on it, performing slow and fast movements, and also taking part of the session with simulated high turbulence.
- Record three color markers in exterior, under strong sunlight conditions with accentuated shadows, and along them regular black and white markers.

The videos are recorded, and then are published as an image topic to the nodes developed in ROS, which will work exactly in the same way as if they were deployed in the actual platform.

The total length of the videos in minutes is no less than 3 minutes, considering that we are working at 30 fps, this translate to no less than 5400 frames per marker, or no less than 27000 frames in total. This seems to be significant figures to compute the statistics over the performance of the algorithm.

4.2.2 ar_pose

The ar_pose node has demonstrated to perform with an astonishingly low consumption of resources. We can see the output of the algorithm on the figure 4.1. To compute its detections, the node requires only about the 32.89% of the CPU power. More over, it has very little variance, within the 1% range even under strong noise conditions, such as heavy shake; this is surprising because some parts of the algorithm are not deterministic, specially the last step which includes an optimization method to re-estimate the pose.

It is important to notice that not all frames are classified, because of race conditions, and the delay of the algorithm in processing the incoming images, there is a set of images which will not be processed, will simply be discarded. For these node it sums a total of 387 frames, that where never processed.

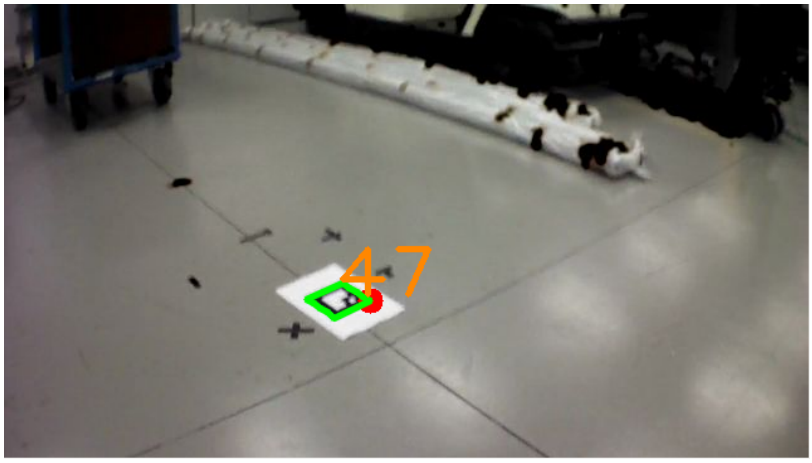

Figure 4.1: Marker Recognition Output

		Predicted	
		Positive	Negative
True	Positive	6688	345
	Negative	123	1109

Table 4.1: ar_pose Visual Marker Detector Confusion Table

The accuracy of the detector was also pretty good, as we can observe in is confusion table 4.1. It is important to notice that during the flight without shaking, no false negatives or positives where obtained, but it was possible to make the detector fail under strong noise conditions.

The max operating distance of the algorithm is estimated to be approximately 8 meters, this distance actually is related to the fact that the algorithm requires at least one clear pixel, with high contrast, in each of the sections of the marker, and visual markers of 15 x 15 cm start to blur out in a 640x360 image at about that distance. If we were working with a camera of higher resolution or print the markers even larger, then we could have even more range. However, the range that concluded this experiments to have are more than enough to develop the interaction scenarios, since a range of 8 meters actually allow the robot to identify the marker without problem from one side of the test bed to the other.

4.2.3 bar_pose

The CPU consumption of the method was not surprisingly higher than the ar_pose method, this fact was already expected because it is a more complex method. However, it requires about 127.68% of the CPU to operate under normal conditions, and that means that it is even using more than one core to compute its results, and even so it is

Figure 4.2: Marker Tracking

		Predicted	
		Positive	Negative
True	Positive	4803	2085
	Negative	0	1053

Table 4.2: bar_pose Visual Marker Detector Confusion Table

not performing at real time, but at mere 7FPS. This situation goes even worse in the cluttered scenario. However, when using the marker tracking, we can improve the use of the CPU to only 58.53%, and the algorithm works almost at real time frame rate. On figure 4.2 we can visualize the output of the algorithm with the marker tracking clearing out the rest of the scene.

The slower the algorithm, more frames are discarded without processing, in this node we have lost 532 input images without processing.

The accuracy of the algorithm is lower than expected, this was a surprise, because it is a more complex method, and more accuracy is usually expected as the main benefit of complexity. On table 4.2 we can see the accuracy of the detector. To be fair, most of the false negatives happened under noisy conditions, and it does not detect false positives. This algorithm seems to be less robust to noisy inputs, one of the reasons for this may be: because the Gauss-Newton optimization is used to fit the contours, and this method does not converge to well defined solutions if the starting point is too far away from the target.

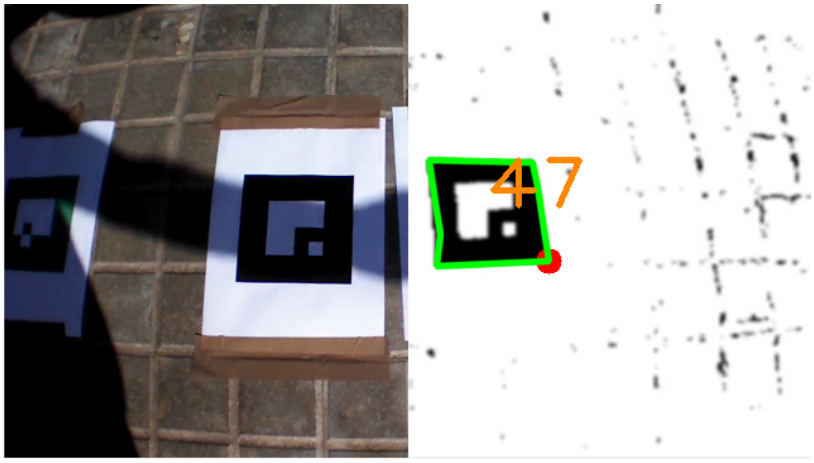

Figure 4.3: Marker Shadowed and Color Threshold

4.2.4 Color Thresholding

If we compare the left and right side of the figure 4.3 the result of the color thresholding becomes very apparent. With proper calibration, the color thresholding can successfully recover suficient information for the visual marker detector to operate properly. The thresholding operates in realtime, and takes a constant 25% of CPU to operate. Because the algorithm does not perform the detection itself, we cannot talk about its accuracy; but if we calibrate the thresholding properly, the marker detection algorithm will benefit of it, maintaining its accuracy.

It is important to note that low resolution cameras also often have poor colors. For our experiments, we recorded videos with both frontal and bottom cameras of the AR.Drone 2 platform. The videos recorded with the frontal camera could be thresholded perfectly and with little effort to tune the calibration. But the videos from the bottom camera had such poor saturation of color, that when we try to apply the color threshold it is often very difficult if not impossible to calibrate. The results of a poor calibrated color threshold is either a blank image or a very noisy image, both useless to the marker detector. So we can conclude that the color threshold is an effective method, provided that we have good cameras in our platform.

5 Face Detection

One of the classic problems in computer vision is to detect human faces, that may be because of its broad range of interesting use cases, such as security clearings, augmented reality applications, process automation, etc. However, within these scenarios, we must define precisely the implicit goal of the case: face detection or face recognition.

A classic method for face recognition is the Eigenfaces method by Turk and Pentland [57]. In this method and many others later developed, the idea is to encode a face as a combination of low dimensional features. For the Eigenfaces method such features means to compute a Principal Component Analisys (PCA) and then encode the face as the combination of the eigenvectors.

We are however more interested in the face detection problem, because our input is an image of the real world and we are not sure if that image contains or not a face, or where. The output of the face detection algorithm will then be the input for other algorithms such as face recognition and navigation.

In the proposed scenarios for our aerial platform one of the most basic parts is to detect and identify the user. This enables for complex interaction patterns between the robot and the person, for example:

- We can adjust the position of the quadrotor by tracking the user, sustaining a "face to face" interaction to improve the user experience.
- The quadrotor could approach the user if it is too far away, or take some distance if it is too close. It is important that the platform stays within a distance range of the user. This allows to sustain a proper communication and avoid crashes or harming the user.

5.1 Method

The state of the art in face detection is being currently achieved by using CNNs. The work by Schroff et al. [51] show results that easily surpass any other previous model for face detection, recognition, and even clustering; and the work by Farfade et al. [17] shows a model that can be used to perform robust face detection, learning to detect faces without using annotations on the dataset, and it can accurately estimate a bounding box for a face in the image.

These state of the art models are however extremely computationally intensive, and while feasible for workstations and offline processing, the CNNs are far from being feasible for real time detection inside an aerial robot. This is the main reason for discarding the use of such models, even when we know that these are the models with the best performance.

Previous work that is worthwhile of mention is Mathias et al. [42]. In this work, a DPM is used in conjunction with an SVM to perform a robust face detection. This model was the state of the art until the latest generation of CNNs surpassed its performance. Also worth of noting is the work by Li and Zhang [39], that combine SURF descriptors with a cascade of classifiers, resulting in a fast yet robust frontal face detector. This work can be seen as an update of the Viola-Jones method using more complex features, of course it is also more computationally expensive than the classic Viola-Jones.

In this work, we have decided to use the Viola-Jones method, or more specifically its implementation in OpenCV; this choice was favored because it is a very efficient implementation and has an acceptable accuracy for our later experiments.

5.1.1 Viola-Jones

The current standard or baseline algorithm to perform face detection is the Viola-Jones algorithm, it was proposed in 2001 by Viola and Jones [61] as a general framework to perform object detection, and it was improved and refocused in their posterior work Jones and Viola [32] to detect faces. As a general framework for object detection it does not perform very good, but for the face detection task, it has reasonably good results and it is very efficient in computing resources; because of this, the algorithm has been implemented in many open-source and commercial computer vision libraries, such as OpenCV, and it has been embedded into many different devices, such as digital cameras and smartphones.

The Viola-Jones method can be summed up in the following steps:

- **Haar-Like Feature Selection:** Haar-Like features are binary differences over patches of an image, these differences are used to detect certain relationships between regions of the image, and its binary nature and rectangle shape reminds the Haar wavelets used for signal analysis. Haar-Like features where first proposed by Papageorgiou et al. [45] as a compact encoding for the information in an image, however the ones who popularized its use and coined the term were Viola and Jones. The Haar-like features transform an image into a small vector representation that only encodes the differences, and this can be interpreted as a feature space that we can use to classify the images later.
- **Creating an Integral Image:** the integral image is actually a matrix, and it is also called the summed area table; the matrix has the same size of the image and

the value of each cell is the summation of all the values of pixels above and to the left. Computing the integral image is $O(n * m)$, for an image with $n * m$ pixels, and after computed it takes constant time to calculate the intensity of any region in the image, only having to take the values of four cells of the matrix.

- **Adaboost Training:** leveraging on the AdaBoost method, the algorithm essentially tries iteratively to add weak linear learners to build a strong learner. More specifically, what the method really does is to compute a haar feature over all the images in the dataset, then if the result was better than the other features previously computed, it is added to the set, otherwise it is discarded and other feature is tried. When we have a set of features that are good enough, we combine them by simply making a weighted sum of its outputs.
- **Cascading Classifiers:** the idea of the cascades is that instead of applying a strong classifier to all the patches, we build an ensemble of classifiers, and we apply them iteratively. First it is applied a weak classifier of only two features, and this discards the vast majority of windows that are not useful, because do not contain faces. Then we apply a stronger classifier of ten features, that will discard the same ratio of the previously selected windows. If we continue in this fashion, we obtain a very good detection rate, with low false positive rate.

5.1.2 Face Tracking

To improve the frame rate we implement a face tracking algorithm, this tracking algorithm is very similar to the marker tracking algorithm introduced in section 4.1.4. It follows the same idea and uses the same linear velocity model to estimate the position of the face on the next frame. However, the face tracking differs in some aspects with the marker tracking:

- When a face was detected in a previous frame and is not detected on the current frame, the algorithm keeps estimating its position, and keeps searching in the area close to where it was, all these for a n number of frames. This modification was done because the face detection algorithm is more prone to false negatives, but it usually only happens in a small number of frames, so if we keep updating its position it is very likely to recover the detected face in two or three frames. When the tracking is prolonged without a detection, the size of the search window around the marker is enlarged, to account for the uncertainty that is accumulated in each successive frame. Finally the algorithm will fall back to perform detection on the whole image, just like the marker tracking, if the face is not recovered in a small window of frames.
- Face detector algorithms do not return an id for the face, only its bounding box. This is an issue when performing tracking, because we cannot tell which face is between successive frames, and therefor we cannot estimate its velocity. There are some options to overcome this issue: one is to perform a fast recognition

algorithm such as eigenfaces, to assign an id to the face in each frame, and then track only the faces whose ids coincide between the frames. An alternative to the recognition is to compute any feature descriptor algorithm, and then track following the same principle, only when the descriptors coincide. Another option is to take an heuristic approach, if the detected face is within a small region of the predicted position, it is very likely to be the same face, so we can handle it as the same. We took the heuristic approach in our implementation, mainly because it is more computationally efficient. The reason to perform face tracking is to improve the performance, so taking an additional costly method would be counterproductive.

5.1.3 Face Distance Estimation

Because one of our goals for the face detection is to maintain a safe distance to the user of the platform, we need to estimate the distance from it to the user face. There are several works that attempt to estimate the location of objects using images, one of the earlier works was done by Goncalves et al. [24], who using monocular cameras track the arm and estimate its location in the world with a 3D model. This work is interesting because the task they studied is very similar to the task we are trying to accomplish. It serves to show how this kind of problems was addressed in the past, but most of the ideas are not really useful for our work.

In the work by Saxena et al. [50], the task of depth estimation is taken from a machine learning approach. They have constructed a dataset by taking simultaneously monocular color and depth images, and later they trained a CNN with this dataset. Another work also from the machine learning perspective was done by Torralba and Oliva [54]. In this case the dataset has monocular images tagged with an estimated depth of the whole scene, and the depth model built is a probabilistic clustering model, which allows to perform a regression. In our case we cannot take any of these solutions, the first because did not show precision, and the second will only tell us about the overall depth of the scene but it is not able to tell the depth of each particular object in the picture.

Also, we would like a very fast model, able to run in real-time in an embedded processor. For this kind of task the approach is usually different because we need to compute features and estimations fast enough to feed a navigation or SLAM algorithm. Çelik and Somani [9] use simple line and corner detectors to obtain features in the image and feed their SLAM algorithm. However computing hardware has evolved enough since the year 2009, and we may be able to use more complex features for our project.

Another good idea is to exploit the intrinsic features of the objects we want to detect or that we know are in the scene. Said et al. [49] take this approach, and using a simple geometrical model and the previous knowledge of the object properties, they are able to perform a very good estimation of the depth of the object. We took this approach to our problem, because we know that faces have very well known and stable properties [15]. We can confidently estimate the distance to a face by using a geometrical model.

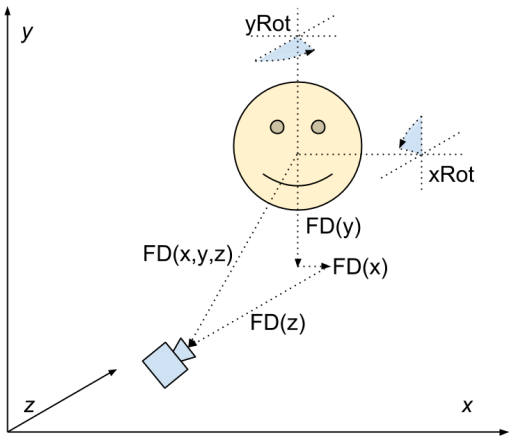

Figure 5.1: Diagram for the face distance estimation formula. FD is short for FaceDistance.

In our model we first estimate the distance in a straight line to the face:

$$|FaceDistance| = FocalLengthMm * \frac{InterEyeDistCm}{InterEyeDistPx} * \frac{ImageWidthPx}{SensorWidthMm} \quad (5.1)$$

Then we can compute the x, y and z components; x and y are derived from the 2d image properties, while the z dimension is reconstructed by the trigonometrical properties of the model:

$$FaceDistance(x) = (FaceMidXPx - \frac{ImageWidthPx}{2}) * \frac{InterEyeDistCm}{InterEyeDistPx} \quad (5.2)$$

$$FaceDistance(y) = (FaceMidYPx - \frac{ImageHeightPx}{2}) * \frac{InterEyeDistCm}{InterEyeDistPx} \quad (5.3)$$

$$FaceDistance(z) = \sqrt[2]{(|FaceDistance|)^2 - (FaceDistance(x))^2 - (FaceDistance(y))^2} \quad (5.4)$$

The formal expression for our distance vector will be:

$$FaceDistance(x, y, z) = (FaceDistance(x), FaceDistance(y), FaceDistance(z))$$

The error estimate of our model can be derived from the formula:

$$|FaceDistance| = FocalLengthMm * \frac{InterEyeDistCm + FaceDifference}{InterEyeDistPx} * \frac{ImageWidthPx + DetectionDifference}{SensorWidthMm} \tag{5.5}$$

$$\frac{|FaceDistance|}{FocalLengthMm * InterEyeDistPx * SensorWidthMm} = (InterEyeDistCm + FaceDifference) * (ImageWidthPx + DetectionDifference) \tag{5.6}$$

Since the focal length, average IPD and sensor width do not change during each experiment, we can replace all these by the constant $k1$.

$$\frac{|FaceDistance|}{k1} = (InterEyeDistCm * ImageWidthPx) + (InterEyeDistCm * DetectionDifference) + (FaceDifference * ImageWidthPx) + (FaceDifference * DetectionDifference) \tag{5.7}$$

Similarly we can introduce $k2$, for the factor of IPD and image width.

$$\frac{|FaceDistance|}{k1} - (k2) = (InterEyeDistCm * DetectionDifference) + (FaceDifference * ImageWidthPx) + (FaceDifference * DetectionDifference) \tag{5.8}$$

In the next formula $k3$ and $k4$ are just short names for the IPD and image width, since they are not changing and therefore not relevant in our formula.

$$\frac{|FaceDistance|}{k1} - (k2) = (k3 * DetectionDifference) + (FaceDifference * k4) + (FaceDifference * DetectionDifference) \tag{5.9}$$

As we can observe, if we group the constants, we can express the error as being proportional to a much compact expression:

$$FaceDistanceError \propto (e1 + (DetectionDifference + FaceDifference + (DetectionDifference * FaceDifference))) * e2 \tag{5.10}$$

Finally we can conclude the following approximation:

$$\begin{aligned} FaceDistanceError \propto & DetectionDifference + FaceDifference + \\ & (DetectionDifference * FaceDifference) \end{aligned} \tag{5.11}$$

The expression in 5.11 can be interpreted as the interaction of two main phenomena that contribute to the error: one is the difference of the person's real face width to the average face width, and the other is the difference of the detected face width to the person's real width. These two factors contribute to each other in an additive and multiplicative manner because of our model, which means that we need to have reasonably good estimations and maintain the factors small, otherwise the estimation can rapidly go wrong.

On the bright side, it is an advantage to have such concise model for the error, because it allows us to adjust the parameters, debug the algorithm, and concisely proof our method. For many methods, specially in machine learning based methods, there is no such model for the error and it needs to be estimated empirically.

There is one problem with our method however: it relies on the detection of the eyes, and this can be a difficult task depending on the resolution of the image. An alternative solution, is to use the head breadth, if we detect the face we are certain that we have the head breadth detected. We can find statistics for the head breadth in [36]. It is important to note that the head breadth have larger variance than the IPD, and this might pose a problem for the estimation. So we will use the head breadth only as a fall back measure when the eyes are not detected in the image.

5.1.4 Face Pose Estimation

Once the face is detected, we can detect some of its parts such as the eyes and nose. And with these parts we can estimate the pose of the head by fitting a geometric model, in a procedure similar to that followed for depth estimation. Horprasert et al. [28] followed this approach in their work, and obtained a reasonable success. We tried the same, but encountered two problems that difficult the task:

- The parts of the face are very small, and are very difficult to detect alone in the image. This makes the model to lose the detection in many frames, which makes it useless for real applications.
- Even when the parts are detected, the variance in the detection is large, making the error of the estimation often also very large.

For these two reasons we searched for another suitable solution, the solution found was the one proposed by Dantone et al. [13]. In their work they propose to use random forests to classify the face into one of five possible poses (profile left, left, front, right, profile right). The poses are actually clusters computed from the dataset, and the algorithm

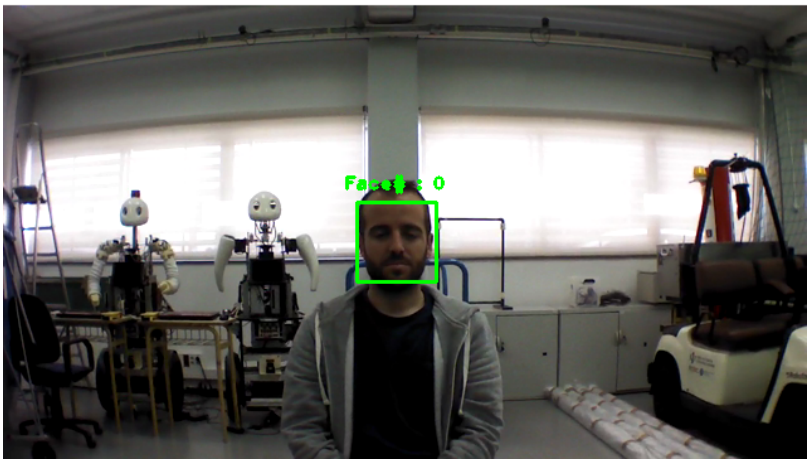

Figure 5.2: Face Detection Output

will perform a regression of a new image as a combination of these clusters.

This approach works much better because it is leveraging on the information of the whole bounding box, and it is not only looking for a part of the face, but is performing a regression of the whole face. So it is a more complex and powerful algorithm, which actually obtains much more information than our previous approach. Gladly, the algorithm is also efficient enough to use in our flying platform. So we took the code of the authors and adapted it to our needs, and also incorporated it to the face detection node.

5.2 Results

5.2.1 Detector Performance

The performance of the Viola Jones algorithm is directly affected by the size of the image to be scanned. For 640x360 images, we have a framerate of only 4 FPS, but if we scale the image by a factor of 2, i.e. working with an image of 320x180, the algorithm works at 30 FPS, the same speed at which images are being published. However, by scaling the image we will also lose accuracy of the detector, in fact we lose so much accuracy that in some cases make the face detection completely useless.

If we count the raw number of detected frames with such low FPS, the numbers are going to be pretty low, and this is because the detector is discarding many frames from the feed, without even processing them. To account for these loss, the detection measures given in this results section, will be re-scaled. The re-escale will be done using the following formula: $ScaledDetections = Detections/FPS * 30$, the idea behind this formula is simple and can be explained with an example: if we detect 3 frames, at 4 FPS, that means that we are detecting the 75% of the incoming frames, and then its detection rate would be

		Predicted	
		Positive	Negative
True	Positive	846	1259
	Negative	100	1193

Table 5.1: Viola-Jones Face Detector Confusion Table

		Predicted	
		Positive	Negative
True	Positive	1692	413
	Negative	100	1193

Table 5.2: Face Detector with Tracking Confusion Table

roughly equivalent to detect the marker in 22.5 frames if we could run at the ideal 30 FPS.

On the table 5.1, we can observe that the numbers for our Viola-Jones based face detector, are not very good. This fact can be explained by two reasons:

- The detector is not so bad actually, if we only take into account the slow experiment we can see that the scaled number of detections is about 352 of 378 total, that is a detection rate of about 93%.
- The low frame rate and the high noise combine its effects: because the low frame rate means that we will discard many images, and under noisy conditions the ones we are left with, are less probable to have a good starting point for the detection.

5.2.2 Detector with Tracking Performance

If we now take a look at table 5.2, we can see the performance of the face detection algorithm with the tracking algorithm on top. This time we can see that actually the face detection is performing much better. It has nearly the double of detection rate but also it improved its average frame rate to 17. As we argued in the previous section the frame rate plays a role in the detection rate, however there is another reason why the tracking improves the Viola-Jones algorithm:
To compute the detections first we need to extract a "pyramid" of windows, i.e. windows of different sizes are extracted from all the ROI to then apply the cascading classifiers. If we have to extract windows from all the image, the scales between the windows have a considerably large variation, some will be large patches of the image, and some very small. Also, we would perform longer strides in pixels between each successive window, this is done to ensure the whole image is covered uniformly.
In contrast, the windows extracted from a small ROI, they have lower variance and are more tightly packed together. This is an ideal situation, because it maximizes the probability of finding the face we are looking for, and find it well framed inside one of those windows.

5.2.3 Distance Estimation Performance

The distance estimation algorithm does not impact on the performance of the face detection, because the only extra work is to compute a small and constant set of formulae.

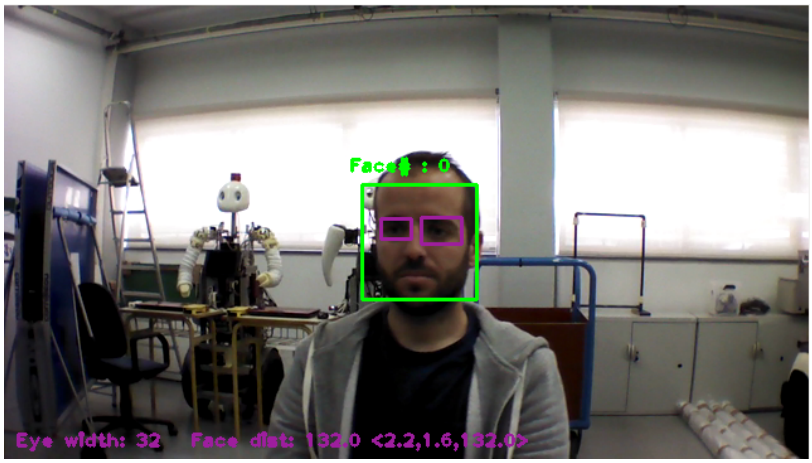

Figure 5.3: Distance Estimation using interpupillary distance

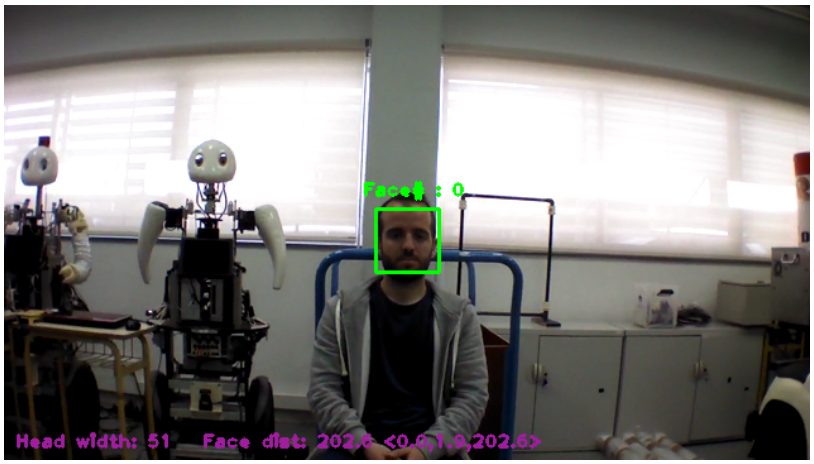

Figure 5.4: Distance Estimation using head breadth

Real Distance	IPD Estimate	Head Breadth Estimate
2m	2.04m	2.03m
4m	3.98m	3.96m

Table 5.3: Average Measurements in Distance Estimation Methods

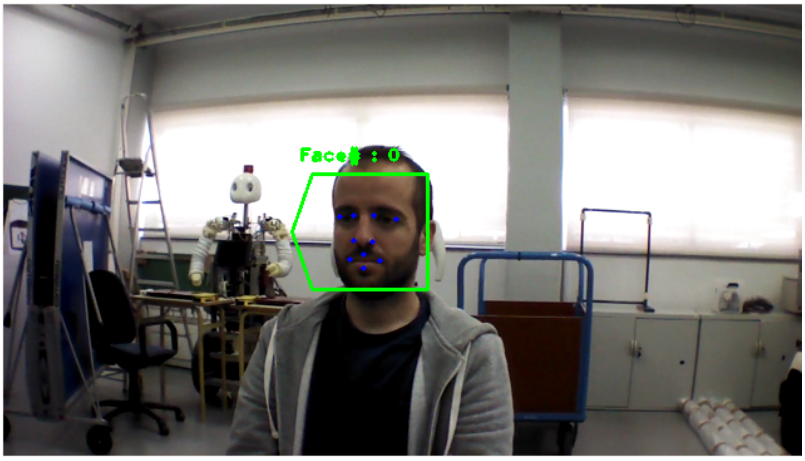

Figure 5.5: Face Pose Estimation

Therefor, the performance of the face estimation is the performance

To precisely measure the precision of the algorithm, we would need to setup external sensors, with precision enough to tell if our measurement is correct to a millimeter scale. Such setup is hard to accomplish, and in our case is not actually needed. Mostly because we know that the variance in the error of our algorithm works on a centimeter scale, so there is no need for such precise measurements. More over, even if we had a very precise method, controlling the movements of a flying platform is a very imprecise task, and the errors in the movements are usually within the decimeter range.

Nontheless we have measured the error by using the classic metric tape, and we have obtained the results in table 5.3. The results are surprisingly good, surpassing the expectations for a simple method, and in fact we can observe that the error measured is within the 2% range. There is a caveat however: the method presents a large variance from frame to frame when the platform is moving. So if we are going to use these estimates in a control algorithm, we would need a filter to take care for the high variance and the outliers.

5.2.4 Pose Estimation Performance

In this section we evaluate the performance of the 3 algorithms working together: face detection, face tracking and face pose estimator. We can see in table 5.4 that the numbers are roughly similar to those of the face tracking algorithm. In fact, the only reason that

		Predicted	
		Positive	Negative
True	Positive	1436	669
	Negative	100	1193

Table 5.4: Face Detector with Pose Estimation Confusion Table

would make them drop, is because the average frame rate of the algorithm also drops, from the 17 FPS we had with the tracking alone, now we have 8.8 frames. We can observe again the correlation between the frame rate and the detection rate.

To measure the precision of the pose estimation, we would need to setup external sensors, just like the case of the distance estimation. However, we can tell from its behavior when is running, that it is not a very precise algorithm. And it also presents a large variance between the estimation of subsequent frames. However it is good enough to give us a rough approximation of the rotation of the face, and this approximation combined with a filter, can be strong enough to perform tasks such as: move the platform to follow the gaze of the user.

6 Object Recognition

Object recognition is one of the most interesting yet difficult tasks to perform in computer vision. It is difficult because it is a very general task that requires a great quantity of domain knowledge to be solved. To overcome the later, usually the task is reduced to detect specific objects, e.g. faces, dogs, cars, etc.

Most of the methods for object recognition will perform also object detection, since they need to tell where the object is before being able to identify it. Even more, most of the methods, e.g. Viola-Jones[61] method, are not good at recognizing particular objects. These methods are more suited for the task of general object detection. Some methods, e.g. the HoG[12], are useful to detect and classify objects, in other words they are good at telling the class of objects, e.g. cats, birds, etc; but they do not perform well on identifying individual objects. However, some methods, e.g. Random Ferns, are well suited for all of these tasks, depending on how we use them. For the sake of simplicity we have grouped all these different methods in the same chapter, and use the title object recognition. Another important thing to notices is that in many scenarios it is not sufficient to detect the object or its class, but we would like to tell if we are dealing with a particular object.

6.1 Method

The Viola and Jones [61] method is still one of the most used methods for object recognition used. However nowadays it is mostly used for face detection and not much else. Mainly because it has some shortcomings, and it has been surpassed by many methods now. the main problems of the Viola Jones are:

- It is good at detecting the objects, but it is not so good at recognition. The reason for this is that the method is not strongly discriminative unless we add many classifiers to the cascade, in which case it becomes computationally expensive.
- It needs to be trained for each class of object to be detected. And it cannot reuse the features computed for other classes. This means that for recognition problems or multi-class classification, we would need to train several ensembles of classifiers, and the computational cost of the detection would also increase linearly with the number of objects or classes to be detected.

These problems are not a concern when only performing face detection, in which case the algorithm is still a good option. The whole method is explained in more detail in

section 5.1.1 of chapter 5

Current state of the art detectors use Convolutional Neural Networks to detect and classify objects. Girshick et al. [23] presented a CNN architecture that can be used to extract features in a hierarchical fashion, allowing to produce accurate detection and good enough classification over the ImageNet [14] dataset. The problem with the CNN approach is the same in this case as the other areas, it is computationally intensive. In the work by Ren et al. [48], this problem is addressed by testing different architectures and it achieves results on par with the original, at a fraction of the computational cost. However, it is still too intensive to be run inside most mobile platforms, the most efficient model proposed in the work is the FZ model, and it runs at 17 fps in a NVIDIA K40 GPU [1], which is a server GPU. If we compare that with the fastest mobile GPU available nowadays (NVIDIA Jetson TX1[2]), the mobile GPU has less than one tenth of that processing power, and even less memory and bus speed. Even more, we are looking for models that can be computed using a CPU, since our platform does not count with a GPU yet.

There is also previous work that searched for more efficient solutions, Torralba et al. [55] tried the machine learning approach to object recognition, but using boosting methods. In their work they computed very simple features that describe the image in a very coarse but efficient form, they combine these features as decision stumps by using a boosting method. The method itself share many characteristics with the Viola and Jones [61] method. They improved however in a couple of key points: 1) the computed features although simple, are more descriptive than the HAAR features, 2) the boosting method is designed to share features between multiple classes, this makes the method more efficient and robust when dealing with multiple classes of objects.

The Deformable Parts Model (DPM) are also worthwhile of mention, they were proposed by Felzenszwalb et al. [18], and are a very efficient yet robust option for many problems. The DPM approach takes two levels of abstraction: searching for the parts, and searching for the whole objects. These searches are actually computed as HoG filters that are applied over the image. And after that, both levels are mixed, looking for places that maximize the activation of both filters. The motivation for this method is very intuitive, for example, if we are looking for a car in the picture, we are interested in wheels, doors, windows, trunk, engine compartment, etc, but we are interested also in the spatial relationship between them, a car only is a car if it is assembled. These models are efficient, but are not discriminative enough to detect particular instances of the objects.

Due to the project efficiency requirements, and also taking into account its flexibility for many tasks, the Random Ferns were chosen as the candidate method to implement object recognition in this work.

[1] `http://www.nvidia.com/object/tesla-workstations.html`
[2] `http://www.nvidia.com/object/jetson-tx1-module.html`

6.1.1 Random Ferns

Random ferns is a new type of method that have many points in common with the boosting and random forests. It was originally proposed by Özuysal et al. [44] with special focus on using it as a classifier for computer vision applications. We can summarize the method originally proposed in the following steps:

- **Preprocessing:** laplacian keypoints are detected, similar to SIFT. These keypoints are only used to identify the regions of interest (ROI) to be used within the classifier. Then random affine transformations are computed over this ROI, and also random noise is added, to create a new synthetic and extended dataset of patches.
- **Feature Extraction:** the feature extraction process consists in simply choosing random pixels within the ROI, and perform simple intensity differences over the pixels or small sub patches within the ROI.
- **Statistics computation:** with the features of each ROI computed, a series of buckets are filled to create a fern; it is important to note that in this case the classes within the ferns are the ROI.
- **Classification:** finally, we have all we need to classify the test examples, the differences over the pixels are computed and a lookup over the bins is performed. Then a bayesian product is used to combine the result of a set of ferns into a probability which tells us how likely is the test image (or patch) to be the object that we want to detect.

Villamizar et al. [59] took the random ferns and combined them with boosting, to create an additive model that is efficient to perform online learning, adding new objects to the detector on the fly and taking less than one second to train the model for that object. On top of that, the method is interactive, it asks for human intervention to classify the difficult cases, and it also learns from that intervention improving the detector confidence over the time.

In later work, Villamizar et al. [60] improved the method, to make it even more efficient and robust to detect multiple classes. We will use this version of the method for the object recognition in this work, which will enable us to perform tasks that require to recognize simultaneously different objects.

6.2 Results

The results of the random ferns based algorithm, were quite good. As we can observe on table 6.1, the accuracy is higher than 94%, and this is by far the best result on any of the detection or recognition algorithms tested in this work. But we should also be clear

		Predicted	
		Positive	Negative
True	Positive	4888	306
	Negative	0	1212

Table 6.1: Object Recognition Confusion Table

Figure 6.1: Object Recognition Output

that this algorithm is assisted, which means that when the algorithm is unsure it will ask the user if the detection is correct or not. From this interaction the classifier is retrained, making it more confident to detect the object, and also making it stronger on discarding the negative instances.

The average assistance rate during our tests was about 5%, which can be quite a lot, 5% of assistance means that for a minute of video, the user was asked 90 times, or more than once every second. However, these interventions by the user usually occur on the first frames processed by the algorithm, and the requirement for intervention drops at an exponential ratio. Because our videos were short, the assistance rate seems to be high, but if we were to test the algorithm for longer periods of time, the assistance rate would drop dramatically. This makes the algorithm suitable for scenarios with long interaction between the human and the robot.

The performance was acceptable, the algorithm delivered stable 10 FPS, and it is invariant to noise. This frame rate could be improved by tracking using a similar approach to that followed in the previous chapters. However, we would need to implement a tracking system that worked for multiple objects, which means applying the algorithm over several small ROI, and this would require some reworking on the current implementation.

The robustness of the algorithm to the noise is also quite impressive. Taking a look at the 6.1, we can see that the confidence of the classifier on the detected object is 0.79, which means it is 79% confident, and the image is clearly very blurry. This characteristic is ideal to have in a flying platform, where the fast movements and the turbulence constantly produce motion blur in the image.

The robustness to noise could also be exploited for some scenarios where the images that are being compared were taken by different cameras. Many detector algorithms would fail in this task, because the descriptors depend heavily on intrinsic properties of the image, such as the quality of the sensor and the lens. For example: we could use this algorithm to show the robot an image on its frontal camera, and it could search for the object with the bottom camera, which makes sense if the object is lying on the ground. Another example: we could send the robot a picture of the desired object to search, from a mobile application, and it could search for it.

The working range of the algorithm was exceptional, it could easily work with objects that were over 4 meters of distance. Also the angle of view did not affected significantly the confidence of the detection, it is able to detect the objects even when the camera is almost 90 degrees with respect to them. However, the algorithm was only tested with objects with a clear frontal face, which are the types of objects it was originally designed for. We suppose that, if we test the algorithm with complex 3d objects, with shapes that vary wildly with the angle of view, the algorithm should not perform so well.

To work with the complex 3d we could modify the algorithm to introduce multiple classifiers per object, this was also tried by Villamizar et al. [58] in previous work. The Idea is to classify an object not by one view but with several views of it, e.g. a car can be seen from the front, back, side, top and down, and each of that views is quite different from the other.

7 Interaction Scenario

The command listener scenario is a simple yet general experiment that follows this story line: commands are given to a flying robot either by buttons, voice or gestures, and the task of the robot is to find its way back to a landing base. This demonstration can actually be extrapolated to many specific use cases, for example: following a person or searching for a particular object.

The experiment combines two of the methods developed for this work: face detection and marker recognition. The face detection will be used to center the flying robot at the user gaze, in order to sustain proper communication. Also, the same face detection algorithm was tested to recognize the hands of the user, in order to recognize gestures, i.e. hand movements pointing to a direction.
The marker recognition will be used to identify the base of the platform. However we will not use the ar_pose markers in this case, instead we will use the markers that are proper of the AR.Drone platform. The reason for using these markers, is that they are recognized by the internal software of the robot, and are very efficiently computed, since it is an unique type of marker. This gives us the advantage to perform control of the platform even at 200Hz, which is far more than we can expect to produce with our visual marker node. Also, because they are recognized within the robot, this allows us to keep watching the frontal camera, while recognizing markers with the bottom camera. This is impossible otherwise, because the robot can only transmit one camera at a time. We need to chose if we want to use the front camera or the bottom camera, and switching between them has a small delay.

7.1 Scenario Design

According to Hall's[26] theory of interpersonal distances, it is crucial that the robot do not invade personal space of a user, as it would lead to the feeling of discomfort or nervousness. The social zone accepted for wide range of contact types vary from about 122cm to 366cm. So we designed the experiment setting the drone to stay within this range of distances, this way it would not violate the personal space of the user, but it would allow the proper interaction.

Another important issue to consider about the interaction is the one studied by Kendon[34] in its F-Formation System. We decided that the participant and the drone should form a via-a-vis formation, because we want to use the position of the face of the user as a cue for the position of the drone. Furthermore, this formation is the most natural for the experiment, because when a person approaches to another for social

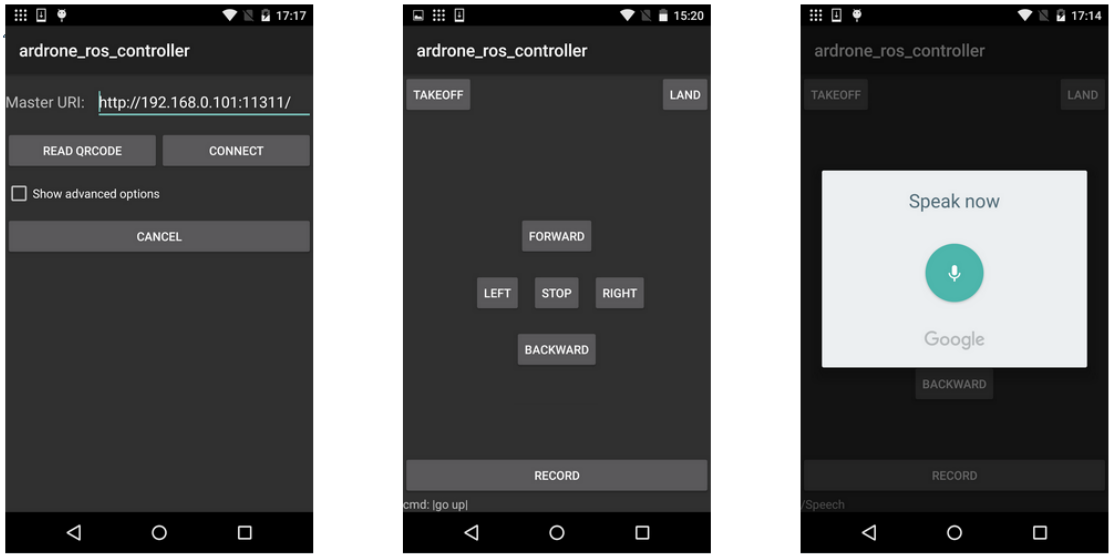

Figure 7.1: Robot controller android application GUI

interaction, he or she positions naturally in face-to-face towards the other person.

To demonstrate the capabilities of interaction with the platform, it will receive commands of the user from three different interaction channels:

- Video channel: we will capture images with the cameras of the robot, which in turn will be analyzed with the face detection algorithm.
- Voice channel: using the microphone of the android device, we can take advantage of the speech recognition capabilities of the Android OS. This speech recognizer would translate the voice commands to ROS messages that control the robot.
- Button command channel: a simple android application, with a graphical interface with buttons, will allow the user to control the robot directly by touching the device screen.

The user gets the feedback from the drone when it moves, or when it sends voice responses through the application. The voice responses are actually text messages that are passed to Android's Text-To-Speech (TTS) system, which reads them aloud. This allows the user to know better the current state of the drone, and this is crucial for proper the HRI.

The drone will leverage on the information obtained simultaneously with the two cameras, with the frontal camera we will watch for the user, and with the bottom camera for the markers. This way the drone can be constantly watching to the user, detecting the face and gesture commands, while searching for its base.

For the button command channel, and android application was developed. We can observe the interface of the application in 7.1. The application is pretty simple and intuitive to use; it only has three screens:

- Start screen: when we open the application we land on this screen. It serves for the connection with the ROS master.
- Button screen: is the main screen of the application. It contains buttons that allow us to send direct commands to the drone.
- Listening screen: if we touch the record button, the application will listen for our command, and will perform a speech recognition algorithm to translate our speech into a text command. The text command will then be parsed, and finally the command will be sent as as standard ROS message.

The application is also listening for incoming text messages that the drone controller can send. These messages contain hints about the state of the drone, e.g. "I'm Lost", "Target Found", "Landing", etc. The text messages are read aloud by TTS through the speaker of the smartphone.

7.1.1 Behavior Design

To control the robot we designed a node that can read the output of the other nodes, i.e. face detection node and flight control node, and uses the information to take a decision, make a movement or perform a task. We call the outcome of this process, and the process itself, the behavior of the robot.

The design for the behavior was based on the finite state machine (FSM) we can observe on figure 7.2. This is a sequential behavior, in which the robot will perform different tasks sequentially, updating its internal state in each step:

1. Base state: is the initial state of the drone, it is waiting for a take off command.
2. Face detection: once the platform is on the air, it will start performing face detection. Once the face of the person is detected it will approach the person and position itself at approximately 1.5 meters, centering the face of the person in the picture.
3. Gesture recognition: once the face is centered, we will start to detect the hands of the user to perform gesture recognition.
4. Target detection: the gesture will hint the robot on which direction it should fly, then it will start moving in that direction while simultaneously scanning for the target. We should remind that our target is just a special marker of the AR.Drone platform.
5. Target found: once the target is found, the robot perform a landing on it. The robot will try to perform a smooth and centered landing, and for this we will use a proportional integral derivative (PID)[25] controller algorithm.

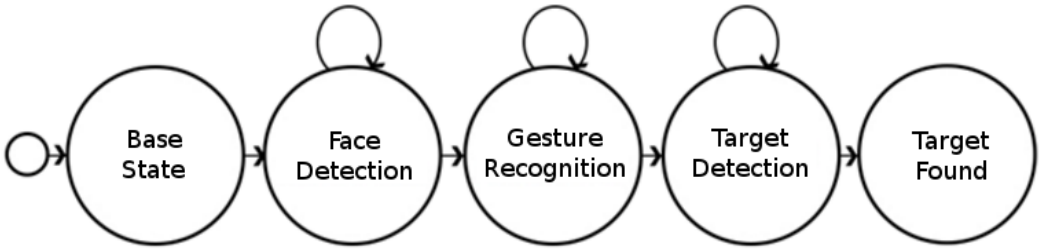

Figure 7.2: Robot behavior FSM

7.1.2 Evaluation Design

To measure the usability and performance of the platform in this scenario, we decided to use the following metrics:

- User ratings: how comfortable and safe users felt around the flying platform.
- Success: how successful was the robot on the search of the target, after a correct recognition of a command.
- Accuracy: how accurate was the platform on recognizing a command that was given by the user.

And our main validation goals were:

- Interaction with the user
- Finding the target

7.1.3 Procedure and Settings

For the safety reasons, we decided to perform the experiments in a stable lab environment. The first experiments were be performed by the development team, for simplicity and safety. Afterwards, the tests were extended to study participants, who were not involved in the development process.

The participants were asked to complete a Pre-Test Questionnaire, with questions concerning their familiarity with drones and technology in general. Then we performed the experiment, measuring the time and accuracy of the robot. And finally the participants were asked to complete a Post-Test questionnaire, that assessed the user rating metric.

The plot of the experiment follows these steps:

1. A participant approaches the robot and gives a voice command.
2. The robot takes off and recognizes a face of the user.

Name	Manual Ctrl	Voice Ctrl	Face Rec	Gesture Rec	Landing
User1	OK	OK	OK	OK	OK
User2	OK	OK	OK	OK	OK
User3	OK	OK	OK	FAIL	OK
User4	OK	OK	OK	FAIL	OK
User5	OK	OK	OK	FAIL	OK
User6	OK	OK	OK	FAIL	OK
User7	OK	OK	OK	OK	OK

Table 7.1: Results of the scenario tests

3. The robot centers on the face.
4. The participant conveys a command by hand gesture, speech or button on the application.
5. The robot moves towards the pointed direction, while searching for the target with the bottom camera.
6. The robot finds the target and informs the user.
7. The robot lands on the target.

7.2 Results

The results of the experiments were satisfactory, as we can observe in table 7.1. The robot easily recognized the manual control and voice commands, and also was able to perform the face recognition and centering successfully for every user. Furthermore, it was able to find the target and land correctly, with a margin of error within the 20 cm.

The only problem was found on the gesture recognition state, the robot performed this task with very random results. The problem was that the hand detection algorithm did not worked properly during testing, it showed a high rate of both false negatives and false positives. And this produced a negative impact in the performance of the robot.

Why the algorithm did not behaved like expected? A couple of hypotheses arise:

- The experiments were performed behind a safety net, which may confuse the haar cascades trained for hand detection.
- The flight of the robot adds movement to the image, which may also affect the detection of the hands, because the hands are a small object in the image, any blurring effect may make impossible its detection.

The human-machine communication turned out very well: we can observe on the table 7.2 that despite the different levels of technological skills of the participants, and their

Name	Pre-1	Pre-2	Pre-3	Post-1	Post-2	Post-3	Post-4
User1	3	1	Math	4	5	1	1
User2	5	3	Comp.Eng	5	5	1	1
User3	5	1	Comp.Eng	5	5	1	1
User4	4	1	Comp.Eng	5	5	1	2
User5	5	3	Comp.Eng	5	4	2	1
User6	5	3	PhD	4	5	1	1
User7	5	3	PhD	4	5	2	2

Table 7.2: Results of the pre and post-test questionaires

previous experience with drones, the interaction with the robot was smooth and effective. They enjoyed the communication with the robot and a feeling of safety was provided. The vital characteristic of our experiment was the simplicity, with which participants directed the "Assistant".

This scenario allowed us to learn an important lesson: the detection algorithms based on the Viola Jones method work well within certain constraints, they work to detect faces or planar object, up to certain distance, and without much movement or occlusion. It seems to be a wrong approach to use this detection algorithm for any type of object and environment.

8 Conclusions

For this work we were required to implement robust and efficient algorithms that were capable to perform visual marker recognition, face detection and the more general task of object recognition. And we have developed nodes for ROS that accomplish each of these tasks:

- Visual marker recognition: we use the ar_pose based on the ARToolkit library, and improved it with marker tracking and color thresholding. We also explored the use of the ArUCO, and bar_pose methods, that were later discarded for being either incompatible or too complex for the task at hand.
- Face detection: we developed the face detection node from scratch. Based on the Viola-Jones method, it was implemented using OpenCV, the LabRobotica philosophy, and the random forests face pose regression code from Dantone et al. [13].
- Object recognition: based on the method and code by Villamizar et al. [60], we implemented the node that also uses OpenCV and the LabRobotica philosophy.

We can say that these are baseline methods for each of the tasks, thus fulfilling the objectives of the project. Also we can say that the software is ready to be deployed in the final platform, since it was all developed following the standards, and no further adaptation should be necessary for it to run on any ROS platform.

The interaction scenarios serve as a demonstration of how the implemented methods can be used to perform complex HRI tasks. With some external additions such as the Android application for the speech recognition and button interaction, we had successful user experiences. In fact some experiences were so good that the users suggested to extend the experiment to make a publication of it.

During the interaction scenario the participants prized the system for its simplicity and intuitive user interface. The easiness of directing the robot was a surprise, and it is only made possible when we include computer vision algorithms. Nowadays most robots lack of this type of interaction with the user, and this presents a gap in the user experience. Because of this, the development of algorithms that help the human robot interaction is a crucial research task.

The accuracy and performance tests are an important part of the analysis for any algorithm, however these are often performed under ideal circumstances that do not

reflect how the algorithm will behave in a more realistic setting. When adding noise to the images, by the shake of the flying platform, we were able to test how robust the chosen methods really are. In many cases, the false positives and negatives grow exponentially to the amount of noise, in other cases also the computing resources required are also increased. Even the common tasks, such as face detection, are not perfectly solved with current methods. So there is still plenty of room to improve the methods developed for this work, but of course it is not a trivial task to develop efficient and robust computer vision methods.

For each of the tasks in the objectives, we have chosen and developed a different method, the one that could perform the best for that particular task. However, if we abstract from all the particular details, we can get to a common core to all the tasks, and it is the object recognition task. So theoretically we could combine all of our tasks, making a single recognition algorithm capable of handling well all of them. That could be a huge improvement, specially in terms of computational resources needed, because instead of scanning the image several times and applying different filters or feature detectors, scanning the input a single time would be sufficient. Some of the state of the art methods for object classification go in this direction, but they most of them are focused on detecting the objects and classify them according to some taxonomy, and are not well suited to perform the recognition or identification task.

Despite we have the right tools: ROS, LabRobotica Philosophy and OpenCV; a big part of the development time was consumed in the implementation of the logic to control the nodes, handling the messages and the threads to make the all the pieces of the system to work right. To develop software for robots, we first have to overcome a barrier of domain knowledge, and even after knowing which tools to use and how, the development of the simplest of the algorithms always implies managing lots of low level details. This may be a reason for the existing gap, between the theoretical complexity of the algorithms developed for robots, and those developed for servers or workstations.

Also there is a gap between the hardware available for mobile robots and for servers and workstations. This gap makes sense because mobile platforms require compact hardware, that is difficult to manufacture. The problem is that this gap in hardware feeds back on the software gap, making it another reason to keep the software of mobile robots simple. Put in other terms, many algorithms are simply not feasible to compute with current mobile hardware. However, future hardware such as the mobile GPUs, which are getting exponentially more powerful computationally speaking; will change the game for mobile platforms in the near future, allowing them to use in real time many state of the art methods. Specially those methods based in deep learning and graphical models could benefit of these new hardware products, thus we can expect that these methods will replace many of the current baseline methods for mobile robots, in the coming years.

Bibliography

[1] OpenCV open computer vision library. `http://opencv.org/about.html`. Accessed: 2016-02-26.

[2] ROS robot operating system. `http://www.ros.org/about-ros/`. Accessed: 2016-02-26.

[3] ar_pose augmented reality marker pose estimation using artoolkit. `http://wiki.ros.org/ar_pose`. Accessed: 2016-02-26.

[4] A. Amor-Martinez, A. Ruiz, F. Moreno-Noguer, and A. Sanfeliu. On-board real-time pose estimation for uavs using deformable visual contour registration. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2595–2601. IEEE, 2014.

[5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006.

[6] A. Björck. *Numerical methods for least squares problems*, pages 342–346. Siam, 1996.

[7] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[9] K. Çelik and A. K. Somani. Monocular vision slam for indoor aerial vehicles. In *Journal of electrical and computer engineering*, volume 2013, pages 1566–1573. IEEE Press, 2009.

[10] K.-L. Chung, Y.-R. Lin, and Y.-H. Huang. Efficient shadow detection of color aerial images based on successive thresholding scheme. *Geoscience and Remote Sensing, IEEE Transactions on*, 47(2):671–682, 2009.

[11] R. Cucchiara, C. Grana, M. Piccardi, A. Prati, and S. Sirotti. Improving shadow suppression in moving object detection with hsv color information. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 334–339. IEEE, 2001.

[12] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[13] M. Dantone, J. Gall, G. Fanelli, and L. Van Gool. Real-time facial feature detection using conditional regression forests. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2578–2585. IEEE, 2012.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[15] N. A. Dodgson. Variation and extrema of human interpupillary distance. In *Electronic imaging 2004*, pages 36–46. International Society for Optics and Photonics, 2004.

[16] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[17] S. S. Farfade, M. J. Saberian, and L.-J. Li. Multi-view face detection using deep convolutional neural networks. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 643–650. ACM, 2015.

[18] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

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

[20] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[21] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, second edition, 2008.

[22] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/j.patcog.2014.01.005. URL `http://www.sciencedirect.com/science/article/pii/S0031320314000235`.

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[24] L. Goncalves, E. Di Bernardo, E. Ursella, and P. Perona. Monocular tracking of the human arm in 3d. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 764–770. IEEE, 1995.

[25] T. Hagglund and K. Astrom. Pid controllers: theory, design, and tuning. *Published by ISA: The Instrumentation, Systems, and Automation Society, 2nd edition, ISBN*, 1556175167(1), 1995.

[26] E. T. Hall. The hidden dimension . 1966.

[27] T. K. Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[28] T. Horprasert, Y. Yacoob, and L. S. Davis. Computing 3d head orientation from a monocular image sequence. In *25th Annual AIPR Workshop on Emerging Applications of Computer Vision*, pages 244–252. International Society for Optics and Photonics, 1997.

[29] T. Horprasert, D. Harwood, and L. S. Davis. A statistical approach for real-time robust background subtraction and shadow detection. In *IEEE ICCV*, volume 99, pages 1–19, 1999.

[30] P. V. Hough. Method and means for recognizing complex patterns. Technical report, 1962.

[31] G. Jian, L. Liyuan, and C. Weinan. A fast recursive algorithm for two-dimensional thresholding. In *Signal Processing, 1996., 3rd International Conference on*, volume 2, pages 1155–1158. IEEE, 1996.

[32] M. Jones and P. Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3:14, 2003.

[33] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Augmented Reality, 1999.(IWAR'99) Proceedings. 2nd IEEE and ACM International Workshop on*, pages 85–94. IEEE, 1999.

[34] A. Kendon. Spatial organization in social encounters: The f-formation system. *Conducting interaction: Patterns of behavior in focused encounters*, pages 209–238, 1990.

[35] T. Kurita, N. Otsu, and N. Abdelmalek. Maximum likelihood thresholding based on population mixture models. *Pattern Recognition*, 25(10):1231–1240, 1992.

[36] J.-h. Lee, S. Hwang, C. L. Istook, et al. Analysis of human head shapes in the united states. *International Journal of Human Ecology*, 7(1):77–83, 2006.

[37] 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):155–166, 2009.

[38] K. Levenberg. A method for the solution of certain non–linear problems in least squares. 1944.

[39] J. Li and Y. Zhang. Learning surf cascade for fast and accurate object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3468–3475, 2013.

[40] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[41] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[42] M. Mathias, R. Benenson, M. Pedersoli, and L. Van Gool. Face detection without bells and whistles. In *Computer Vision–ECCV 2014*, pages 720–735. Springer, 2014.

[43] N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11 (285-296):23–27, 1975.

[44] M. Özuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(3):448–461, 2010.

[45] C. P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *Computer vision, 1998. sixth international conference on*, pages 555–562. IEEE, 1998.

[46] S. Papert. The summer vision project. memo aim-100, 1966.

[47] J. Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. In *Computer Human Interaction, 1998. Proceedings. 3rd Asia Pacific*, pages 63–68. IEEE, 1998.

[48] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[49] Z. Said, K. Sundaraj, and M. Wahab. Depth estimation for a mobile platform using monocular vision. *Procedia Engineering*, 41:945–950, 2012.

[50] A. Saxena, S. H. Chung, and A. Y. Ng. Learning depth from single monocular images. In *Advances in Neural Information Processing Systems*, pages 1161–1168, 2005.

[51] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[52] D. Stricker, G. Klinker, and D. Reiners. A fast and robust line-based optical tracker for augmented reality applications. In *Proceedings of the international workshop on Augmented reality: placing artificial objects in real scenes: placing artificial objects in real scenes*, pages 129–145. AK Peters, Ltd., 1999.

[53] S. Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[54] A. Torralba and A. Oliva. Depth estimation from image structure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(9):1226–1238, 2002.

[55] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–762. IEEE, 2004.

[56] V. J. Tsai. A comparative study on shadow compensation of color aerial images in invariant color models. *Geoscience and Remote Sensing, IEEE Transactions on*, 44 (6):1661–1671, 2006.

[57] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[58] M. Villamizar, H. Grabner, F. Moreno-Noguer, J. Andrade-Cetto, L. v. Gool, and A. Sanfeliu. Efficient 3d object detection using multiple pose-specific classifiers. 2011.

[59] M. Villamizar, A. Garrell, A. Sanfeliu, and F. Moreno-Noguer. Online human-assisted learning using random ferns. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2821–2824. IEEE, 2012.

[60] M. Villamizar, A. Garrell, A. Sanfeliu, and F. Moreno-Noguer. Modeling robot's world with minimal effort. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4890–4896. IEEE, 2015.

[61] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.