UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

JOÃO VICTOR PORTAL

A Java Autopilot for Parrot A.R. Drone Designed with DiaSpec

Trabalho de Graduação.

Prof. Dr. Carlos Eduardo Pereira
Orientador

Porto Alegre, novembro de 2011.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitora de Graduação: Profa. Valquiria Link Bassani
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do ECP: Prof. Sérgio Luís Cechin
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

SUMMARY

LISTA DE ABREVIATURAS E SIGLAS....4
LISTA DE FIGURAS....5
RESUMO....6
ABSTRACT....7
1 CONTEXT....8
1.1 Motivation....8
1.2 Objectives....8
1.3 Drone Overview....9
1.3.1 Drone Operation....9
1.3.2 Sensors and Cameras....10
1.3.3 Wireless connection....11
1.3.4 Drone Input/Output Streams....11
1.3.5 Controlling The Drone....11
2 OUR APPROACH....13
3 DRONE COMMUNICATION LAYER....15
3.1 The Navigation Data Receiver Module....15
3.2 The Video Decoder Module....16
3.2.1 The Image Structure....16
3.2.2 The Video Decoder Algorithm....18
3.3 The Drone Controller Module....18
4 REAL-TIME POSITION LAYER....19
4.1 The Tag Detector Module....19
4.1.1 Research of Available Libraries....20
4.1.2 The Tag Design....20
4.1.3 The Procedure to Find Tags....22
4.2 The Position Calculator Module....24
4.2.1 Calculating The Position Using A Tag....24
4.2.2 Calculating The Position Without A Tag....26
4.3 The Drone Current Data Module....27
4.4 The Flight Plan Module....27
5 AUTOPILOT WITH DIASPEC....28
5.1 Autopilot Control System....28
5.1.1 The Autopilot State Machine and The Autopilot Monitor....28
5.1.2 The Autopilot Acceleration Calculator....30
5.2 DiaSpec Description....32
5.3 Autopilot Architecture....33
6 CONCLUSION....37
REFERENCES....38
APPENDIX A DRONE TECHNICAL SPECIFICATIONS....40
APPENDIX B THE SQUARE FINDER ALGORITHM....42
APPENDIX C THE POINTS ORDERING ALGORITHM....43
APPENDIX D DESCRIPTION OF THE WORK IN PORTUGUESE....45

LISTA DE ABREVIATURAS E SIGLAS

INRIA	Institut National de Recherche en Informatique et en Automatique
LaBRI	Laboratoire Bordelais de Recherche en Informatique
PFE	Projet de Fin d'Études
UAV	Unmanned Aerial Vehicle

LISTA DE FIGURAS

Figure 1.1..9
Figure 1.2..9
Figure 1.3..10
Figure 2.1..14
Figure 3.1..16
Figure 3.2..16
Figure 3.3..17
Figure 3.4..17
Figure 3.5..18
Figure 4.1..21
Figure 4.2..22
Figure 4.3..24
Figure 4.4..25
Figure 4.5..25
Figure 4.6..26
Figure 5.1..28
Figure 5.2..31
Figure 5.3..32
Figure 5.4..33
Figure 5.5..33
Figure 5.6..34
Figure B.1..42
Figure B.2..42
Figure C.1..43
Figure C.2..44
Figure D.1..45
Figure D.2..46

Um piloto automático em Java para o AR. Drone da Parrot criado usando DiaSpec

RESUMO

Este trabalho consiste na descrição de um piloto automático criado para o AR. Drone, um quadricóptero (que será chamado daqui em diante apenas de "drone") fabricado pela empresa francesa Parrot. O objetivo deste piloto automático é fazer o drone cumprir automaticamente um itinerário previamente definido. O itinerário consiste em uma sequência de pontos no espaço pelos quais o drone deve passar. O drone decola, passa por esses pontos e aterrissa no último ponto da rota.

O piloto automático é um software que é executado em um computador. Esse computador se comunica com o drone através de uma conexão Wi-fi. O drone apenas envia os dados de seus sensores e câmeras para o computador. O computador processa os dados recebidos do drone e após envia o comando que o drone deve executar. Ou seja, o drone é controlado remotamente pelo computador.

A linguagem de programação usada neste trabalho foi Java. Esta linguagem foi escolhida porque era desejável que o software pudesse ser executado em diferentes sistemas operacionais sem que fosse necessário alterar o código ou recompilar.

DiaSpec é uma ferramenta desenvolvida pelo grupo de pesquisa Phoenix dos laboratórios do INRIA. Esta ferramenta permite a geração automática de um framework de programação através de uma especificação da arquitetura do sistema que se quer construir. Um domínio de aplicação do DiaSpec é na criação de softwares aviônicos. O DiaSpec é usado neste trabalho para criar o módulo central do sistema.

A criação deste piloto automático envolveu conceitos de arquitetura de software, programação orientada a objetos, redes, sistemas de tempo real e processamento de imagem, além de muita trigonometria. Um vídeo do drone sendo controlado pelo piloto automático foi criado para demonstrar seu funcionamento. Seu endereço na web se encontra no capítulo de conclusão deste trabalho.

Uma descrição mais detalhada em português deste trabalho encontra-se no apêndice D (“Descrição do Trabalho em Português”).

Palavras-Chave: piloto automático, drone, linguagem Java.

ABSTRACT

This work consists in the description of an autopilot created for the AR. Drone, a quadricopter (that will be called simply “drone” from now on) manufactured by Parrot, a french enterprise. The objective of this autopilot is to make the drone accomplish automatically a previously defined itinerary. The itinerary consists in a sequence of points in the space by which the drone must pass. The drone takes off, passes through these points and lands at the last point of the route.

The autopilot is a software that runs in a computer. This computer communicates with the drone through a Wi-Fi connection. The drone only sends data from its sensors and cameras to the computer. The computer process the data received from the drone and after sends the command that the drone must execute. In other words, the drone is remotely controlled by the computer.

The programming language used in this work was Java. This language was chosen because it was desirable that the software run in several operating systems without it being necessary to change the source code or to recompile the code.

DiaSpec is a tool developed by the INRIA Phoenix research group. This tool allows the automatic generation of a programming framework through the specification of the target system architecture. An application domain of DiaSpec is in the creation of software for avionics. DiaSpec is used in this work to create the central module of the system.

The creation of this autopilot involved concepts of software architecture, object-oriented programming, networks, real-time systems and image processing and a lot of trigonometry. A video of drone being controlled by the autopilot was created to demonstrate its operation. His address in the web is in the chapter of conclusion.

Keywords: autopilot, drone, Java language.

1 CONTEXT

1.1 Motivation

The present work was developed during a PFE internship in the INRIA labs in Talence, France. In the INRIA labs, the Phoenix research group works in projects related to the services composition and orchestration. One of the Phoenix group projects is DiaSuite, that is a tool suite that guides the developer through the development of applications that interacts with the environment. DiaSpec is the a tool of DiaSuite used to design the architecture of the application and to generate code from this architecture. A more detailed description of DiaSpec can be found in the section 5.2 of this rapport.

One of the application domains of DiaSpec is in the development of software for avionics. The Phoenix group already has created applications using DiaSpec for avionics, but only as simulators. It would be interesting to this research group to have an application developed using DiaSpec that controls a real avionic device. The INRIA labs has some models of the Parrot A.R. Drone, that is a quadricopter controlled using a Wi-Fi connection. This quadricopter has a few automatic procedures: it can take off and land alone and this make the control by the user a lot easier. The detailed description of the drone can be found in the section 1.3 of this rapport.

1.2 Objectives

The objective of the present work is to create an autopilot for the Parrot A.R. Drone using the DiaSpec tool. The implementation will be done entirely using the Java programming language. The Java language has been chosen because of its compatibility and flexibility: the programs described in this language can be easily executed in any operation system with the Java Virtual Machine.

1.3 Drone Overview

It is very important to study the drone features, because this will determine the complexity of the autopilot and the auxiliary modules that are necessary. In this section we will present the drone features. The description of how the autopilot will be constructed will be discussed in the chapter 2.

The AR. Drone is a quad-rotor UAV (unmanned aerial vehicle) that receives commands via wireless connection. It can do automated actions like take-off, land and stay in the same position while flying (action known as “hover”). It may be used outside and inside buildings, with or without the protection structure (that will be named here as “indoor hull”). It uses a specific rechargeable lithium battery provided by Parrot. The drone sends real-time information about the its current state, current position and orientation, and also sends the video stream of a selected camera. It has an embedded Linux Operation System that can be accessed via Telnet protocol.

1.3.1 Drone Operation

For indoor use of the drone, the drone must be protected with the indoor hull. For outdoor use, there is another hull (the “outdoor hull”). Below are two pictures of the drone with there hulls.

Figure 1.1: The indoor hull (on the left side) and the outdoor hull.

To understand how the drone fly, consider the image below.

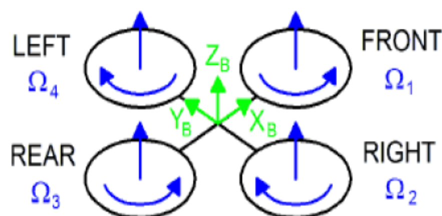


Figure 1.2: The rotation of the motors

Each pair of opposite rotor turns in the same way: one pair turns clockwise, while the other turns anti-clockwise. Consider Ω1, Ω2, Ω3 and Ω4 as the angular speeds of

each motor. To stay flying in the same position, the drone must keep all the angular speeds equal. To go in the direction of axis XB, YB and ZB, the drone change the angular speed of the angular speeds. To understand what the drone do to go in one direction, first consider the image below.

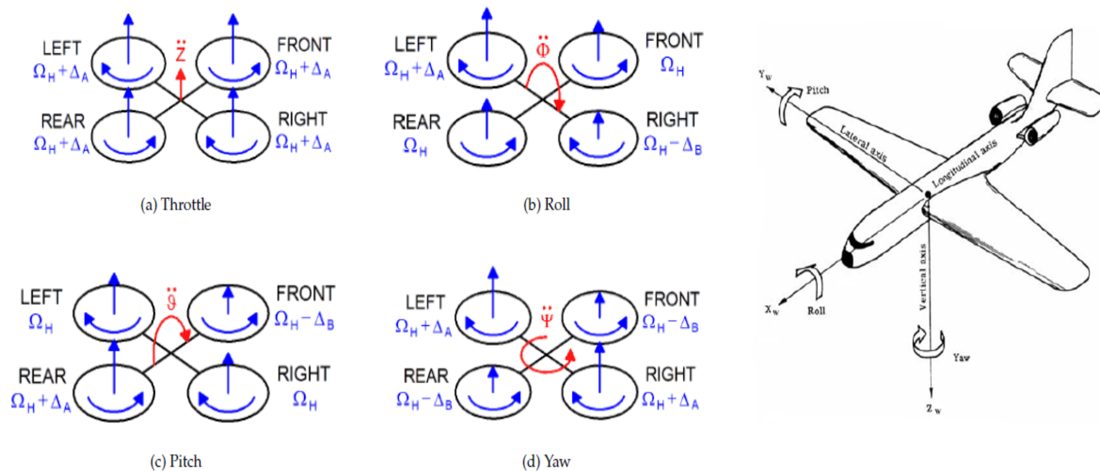


Figure 1.3: Throttle, roll, pitch and yaw (on the left side) and the equivalent in a plane.

Consider ΔA and ΔB as positive angular speed values.

To go in front and rear directions, the drone must change the pitch angle. To change the pitch angle, is necessary to change the angular speeds of the front and rear motors (case “c” of the above picture).

To go in left and right directions, the drone must change the roll angle. To change the roll angle, is necessary to change the angular speeds of the left and right motors (case “b” of the above picture).

To turn to left and to right, the drone must change the yaw angle. To change the yaw angle, is necessary to change the angular speeds of all motors (case “d” of the above picture), changing the speeds of each pair of opposite motors by the same amount.

Finally, to go upward and downward, the drone must change all angular speeds of the drone by the same amount (case “a” of the above picture).

To better understand what these angles mean, see the roll, pitch and yaw angles in a plane (figure 1.3, right side).

The drone uses a lithium battery designed by Parrot. A full charged battery gives power for about 12 minutes of continuous fly.

1.3.2 Sensors and Cameras

The A.R. Drone has many motions sensors. These motion sensors provides the software with pitch, roll and yaw measurements. These measurements are used for automatic pitch, roll and yaw stabilization and for assisted tilting control.

An ultrasound telemeter provides altitude measures that are used for automatic altitude stabilization and assisted vertical speed control.

The drone has two cameras: one front camera in horizontal direction, and a vertical camera towards the ground. The bottom (vertical) camera is used by the drone to

measure the ground speed, that is used for automatic hovering and trimming. The drone can send a encoded stream of these cameras: see section 3.2 for further details.

1.3.3 Wireless connection

The wireless connection is used by a client to control the drone. Any client device supporting Wifi in ad-hoc mode can connect to the drone. When we turn on the drone, it automatically creates a network with an ID called *adrone_xxx*, where *xxx* are numbers. If the drone detects an already existing network with the ID it intended to use, it joins the already-existing Wifi channel.

1.3.4 Drone Input/Output Streams

The drone sends two UDP output streams: the Navigation Data stream and the Video stream. The Navigation Data stream contains information about the current status of the drone and is described in the section 3.1. The Video stream contains the encoded data of one of the cameras and is described in the section 3.2.

The drone receives one UDP input stream: the stream that contains the commands. This stream is described in the section 3.3 and the way that the drone is controlled is shown in the subsection 1.3.5.

There is a communication channel that use the protocol TCP and is used to change the default configuration of the drone. This channel will not be used in the present work.

1.3.5 Controlling The Drone

To control the drone, a client must send commands in UDP packets using the Wifi connection. In this subsection, it's presented the automated procedures that the drone has and the PCMD command. This is what is necessary to understand how the drone is controlled. For a full description of all commands and their syntax, see the chapter 6 in Piskorski (2011).

The commands are formed by a string of 8-bit ASCII characters that always starts with "AT*", followed by the command name, the equal sign, a sequence number, and a optional list of parameters. An AT command must be separated of another by newlines if they are in the same UDP packet. Example of command: *"AT*PCMD=21625,1,0,0,0,0"*.

1.3.5.1 Automated Procedures

The drone implement some automated procedures. They are: "take off", "land" and "hover". To activate these procedures, we just need to send the right command and the drone will do the procedure. In the "take off" procedure, the drone will automatically start to fly and will stay in a distance of about 1 meter of the ground. In the "land" procedure, the drone will automatically start to land and this procedure ends when the drone is completely stopped in the ground. In the "hover", the drone will automatically try to stay in the same position in the space, canceling the inertial speed of the drone.

1.3.5.2 The PCMD commands

The PCMD command is the command used to make the drone translate and rotate. PCMD means “progressive commands”. The syntax of this command is: *“AT*PCMD=<sequence_number>,<flag>,<roll_p>,<pitch_p>,<gaz_p>,<rot_p>”*.

The argument *sequence_number*, as the name says, is the number of sequence of the command and depends of how many commands have been sent before.

The argument *flag* defines if the drone will consider the arguments that follow it. If *flag* is 1, the drone will consider the arguments *phi*, *theta*, *gaz*, *rotation*. If *flag* is 0, the drone will execute the “hover” procedure (it will try to stay in the same position in the space, canceling the inertial speed of the drone).

The argument *roll_p* is a value in range [-1..1] that represents a percentage of the max value of the angle roll that the drone can achieve. This max value is set as default in the drone as 12 degrees (this value will not be changed in this work). For example, if *roll_p* is 0.5, the drone will bend 6 degrees in the roll angle, consequently moving to the right. If *roll_p* is -0.5, the drone will bend -6 degrees in roll angle, consequently moving to the left. If *roll_p* is 0, the drone will stay at zero degrees in roll angle.

The argument *pitch_p* is a value in range [-1..1] that represents a percentage of the max value of the angle pitch that the drone can achieve. This max value is set as default in the drone as 12 degrees (this value will not be changed in this work). For example, if *pitch_p* is 0.5, the drone will bend 6 degrees in the pitch angle, consequently moving backward. If *pitch_p* is -0.5, the drone will bend -6 degrees in pitch angle, consequently moving forward. If *pitch_p* is 0, the drone will stay at zero degrees in pitch angle.

The argument *gaz_p* is a value in range [-1..1] that represents a percentage of the max value of the vertical speed that the drone can achieve. This max value is set as default in the drone as 0.7 m/s (this value will not be changed in this work). For example, if *gaz_p* is 0.5, the drone will go in the Z axis with a speed of 0.35 m/s, consequently moving upward. If *gaz_p* is -0.5, the drone will go in the Z axis with a speed of -0.35 m/s, consequently moving downward. If *gaz_p* is 0, the drone will stay in the Z axis with a speed of 0 m/s, consequently not moving in this axis.

The argument *rot_p* is a value in range [-1..1] that represents a percentage of the max value of the angular speed that the drone can achieve. This max value is set as default in the drone as 100 degrees/s (this value will not be changed in this work). For example, if *rot_p* is 0.5, the drone will turn right at 50 degrees/s. If *rot_p* is -0.5, the drone will turn left at 50 degrees/s. If *rot_p* is 0, the drone will not rotate.

This interface of the PCMD command makes the drone easy to control: if the client wants the drone to stay in the air doing nothing, he just need to send the command with all the 4 last arguments equal to zero, and the flag equal to 1. The drone will stay in his position, but will slide a little bit because of the inertia. If the client wants the drone to stay in his position canceling the inertia, he just need to send the command with the flag equal to 1. The drone will use internal algorithms and the image of the bottom camera to try to be in the top of the same point (this is the “hover” procedure). The user can make complex movements sending values for the 4 last arguments at the same time with the flag argument equal to 1. For example, the user can make the drone go forward and rightward while rotating to right and going upward, sending the right values of the 4 last arguments at the same time.

2 OUR APPROACH

To construct the autopilot, we need as input a system that can provide real-time information about the position and orientation of the drone in the space and its velocity. As output, we need a way to send commands to the drone.

The drone sends via Wi-fi connection its information of relative position, orientation and velocity. It sends information about the position X, Y and Z of the drone in the space, the angles of orientation Yaw, Pitch and Roll (see section 1.3.1 to understand these angles) and the current velocity of the drone in the X, Y and Z axis. When we turn on the drone (or when we send the "flat trims" command), it stays stopped and take as reference the horizontal plane where it is (Pitch and Roll are adjusted, and Yaw is set to zero). When we make the drone take off, it starts to send valid values of X, Y and Z, considering as the origin of the position system (X,Y,Z = 0, 0, 0) the point where it has took off. Also, after it takes off, it starts to send valid information of the velocity of the drone in X, Y and Z axis.

However, there are many problems in using the position and the orientation sent by the drone:

– First, the position information (values of X, Y and Z) sent by the drone has a precision that is not so good. For example, if the drone has gone 1,5 meters in one direction, the position system may only register 1,3 meters of displacement;

– Second, as the time passes, the drone slowly lost the reference of Yaw angle. The drone slowly increases or decreases the value of Yaw without any physical change of orientation of the drone. Consequently, the drone loses the reference of its orientation.

– Finally, in this work, the origin of the position system of the software not always correspond to the place where the drone took off. We want to make the drone take off in different places and we want that the drone always know his right position.

To resolve these problems, we will need a way to reset, or calibrate, the position and orientation system periodically on the fly. The solution found to do this calibration is to use the image of the bottom camera of the drone to recognize special markers in the floor, that we will refer in present work simply as "tags". When we find a tag in video of the bottom camera, we extract the information of position and orientation coded in this tag. Also, to have a precise value of the current position, we will use the information about the current drone angle, the information about the current altitude of the drone (measured by the drone using the ultrasound telemeter sensor) and the information of the position of the tag in the video image to calculate the physical distance from the drone to the tag in the X, Y and Z axis.

Our approach to create the drone autopilot is a layered scheme. We will create four software layers: the Drone Communication layer, the Real-Time Position layer, the

Autopilot layer and the User Interface layer. The Drone Communication layer will be responsible to decode the drone incoming streams and to send commands to the drone as an output stream. The Real-Time Position layer will calculate in real-time the right position and orientation of the drone using the information provided by the Drone Communication layer. The Autopilot will choose the right command to send to the drone using the information provided by the Real-Time Position layer. The User Interface layer will provide to the user global control and information about the system and will interact with all the others layers. This scheme is in the figure 2.1. In this figure, the arrows represent the data flow.

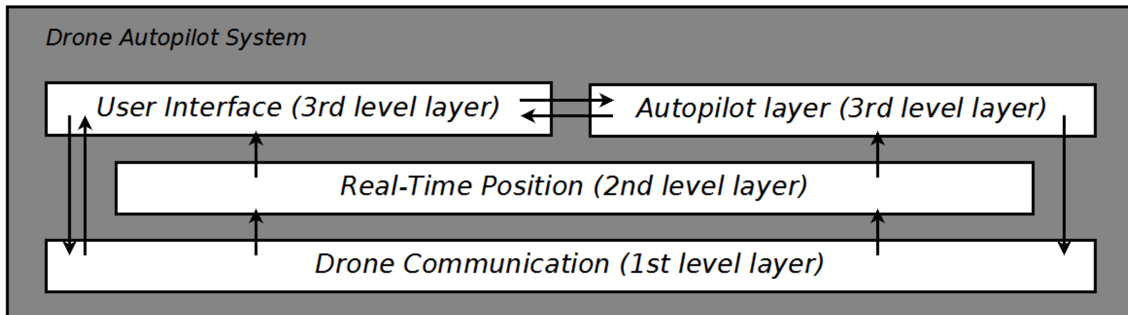


Figure 2.1

3 DRONE COMMUNICATION LAYER

The Drone Communication layer is responsible for all the communication between the drone and the Drone Autopilot System. Basically, this layer receives and decodes the drone output streams and sends commands to the drone. The drone output streams are two: the navigation data stream and the video stream. The navigation data stream contains the information of position, orientation, battery level, current drone mode (like “landed”, “flying”, “taking off”), velocity, drone altitude (measured by the ultrasound telemeter sensor) and others less important information. The video stream contains the encoded images captured by one of the two cameras (and can also send a mixture from the two cameras).

As the task of this layer is to receive two input streams and send one output stream, this layer is divided in 3 modules: the Navigation Data Receiver, the Video Decoder and the Drone Controller (that sends commands). Each module is described in the next sections.

3.1 The Navigation Data Receiver Module

The drone sends the stream called “Navigation Data” via wireless using the UDP protocol in the port 5554. This is the stream that contains information like the drone position, orientation, velocity and battery. The task of this module is simple: it receives the Navigation Data stream and unpack the necessary variables.

Not all the values of the Navigation Data stream are read by this module. See below a list of the values of the stream that are read:

- The current battery level;
- The angles Theta, Phi and Psi, that inform the current drone orientation (see subsection 1.3.1);
- The value of the current altitude measured by the ultrasound telemeter sensor of the drone;
- The values of the current position of the drone in the space, as values in the X, Y and Z axis;
- The values of the current velocity of the drone in the X, Y and Z axis;

As already said in the section 2, the position information (values of X, Y and Z) sent by the drone has a precision that is not so good and the Yaw angle reference is lost as the time passes. This motivates the creation of the Real-Time Position layer. See section 2 (“Our Approach”) for full details.

3.2 The Video Decoder Module

The drone sends the video stream via wireless using the UDP protocol in the port 5555. The video stream is encoded in a simplified H.263 UVLC (Universal Variable Length Code). This means that the drone video stream is just a set of complete images sent one by one by the drone. To create a decoder to the video stream, it's not very complicated: it's necessary to create a decoder for one image and call this decoder each time that a new image comes. The video stream sends 15 images by second, and the resolution is QCIF (176x144 pixels) for the bottom camera and QVGA (320x240 pixels) for the front camera. A brief description of the structure of these images and the algorithm to decode them are presented in the next subsections.

3.2.1 The Image Structure

The images sent by the drone are split in Group of Blocks (GOBs). A GOB is just a horizontal slice of the image. The image below show an image divided in GOBs.

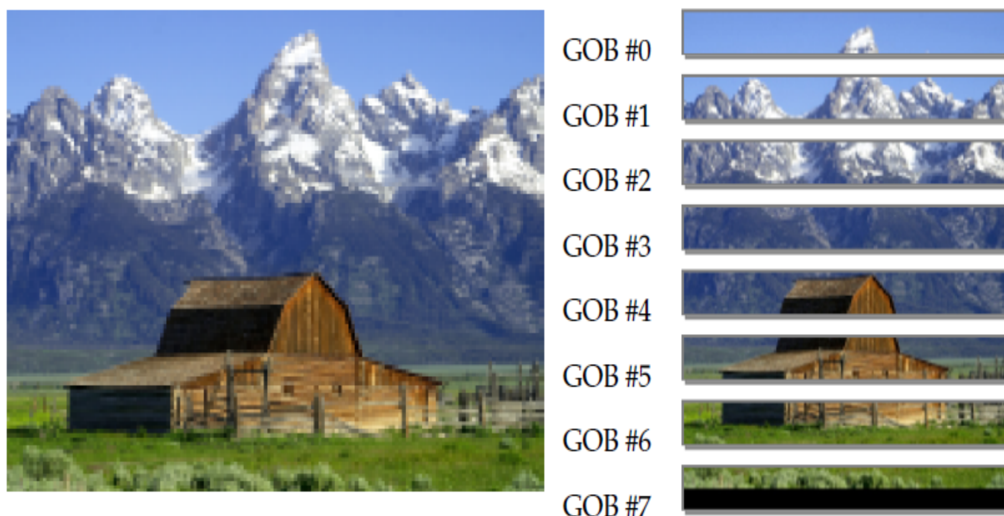


Figure 3.1: An image split in GOBs

Each GOB is split in Macroblocks, which represents a 16x16 image. The image below show a GOB split in Macroblocks.

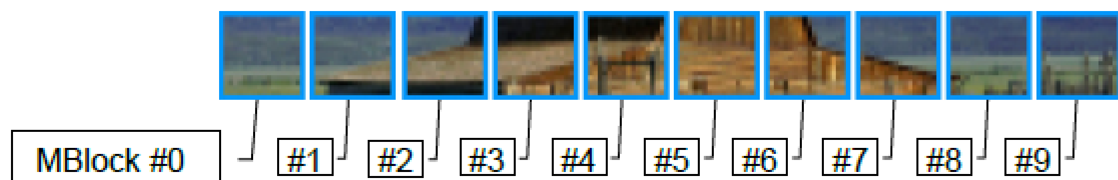


Figure 3.2: A GOB split in Macroblocks

Each Macroblock contains information of a 16x16 image, in YCBCR format, type 4:2:0. To understand this format, consider the image below.

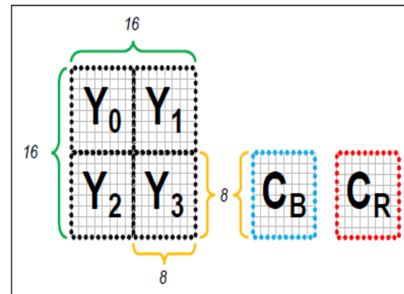


Figure 3.3: The Macroblock

As shown in the figure 3.3, the 16x16 pixels Macroblock is composed of six 8x8 blocks: four luminance blocks (Y_0, Y_1, Y_2, Y_3) and two down-sampled color blocks (CB and CR). This format, known as YCBCR 4:2:0, maintains the full information of the luminance of the 16x16 pixels image, while loses information about the color. The 8x8 color blocks cover an area of 16x16 pixels. In other words, each pixel of the color blocks will be used to generate four pixels of the final 16x16 image.

Each of the six 8x8 blocks of the Macroblock is encoded using the first steps of the JPEG encoding: DCT (discrete cosine transform), Quantization and Zig-Zag Ordering. After these steps, the zig-zag list of values are coded using an specific Entropy Encoding. The whole process is shown in the figure 3.4.

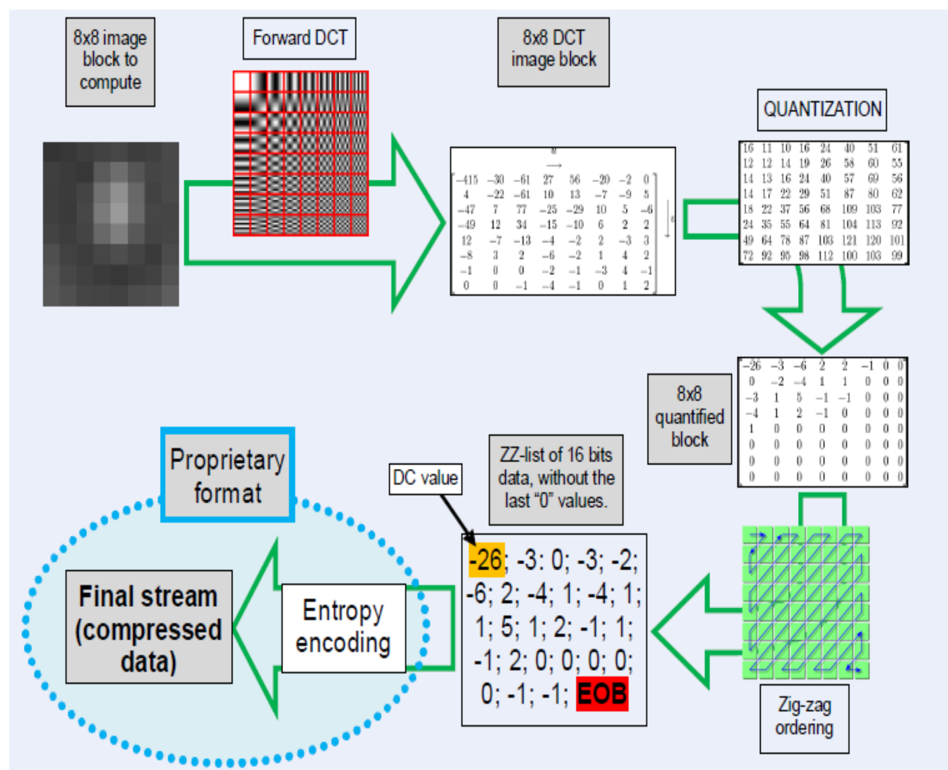


Figure 3.4: The process of codification of the 8x8 blocks of a Macroblock

The Entropy Encoding process is based on a mixture of Run-Length Encoding and Huffman Coding. This process will not be described in detail in this document. More information about the Entropy Encoding can be found in the chapter 7 in Piskorski (2011).

3.2.2 The Video Decoder Algorithm

The algorithm used to decode the stream of the video is shown below. It is described here in pseudo-code and in a high-level way.

```
For each new image that arrives {
	For each GOB {
		For each Macroblock {
			For each of the six 8x8 blocks {
				Undo the Entropy Encoding;
				Undo the Zig-Zag Ordering;
				Revert the Quantization;
				Apply the Inverse DCT;
			}
			Convert from YCbCr to RGB
		}
		Combine the Macroblocks in one GOB
	}
	Combine the GOBs in one image
}
```

Figure 3.5: The algorithm of the video decoder

3.3 The Drone Controller Module

The drone receives commands via wireless using the UDP protocol in the port 5556. The AR Drone Developer Guide recommends to send commands every 30ms (or less) for a satisfying control of the drone. This is what this module does: it implements a queue of commands and sends a new command every 20ms. If there isn't any new commands, it sends the last command sent. See section 1.3.5 to understand the commands and how the drone is controlled.

The other responsibility of this module is to initialize the communication with the drone. This means to make the drone initiate the transmission of Navigation Data and Video streams.

4 REAL-TIME POSITION LAYER

In this chapter is presented the system that is used to detect the position of the drone in the space. This system is implemented in the second layer of the software. To know the position and the right orientation of the drone, we will need to extract information from the environment. This is necessary because, as already discussed in the chapter 2, the drone loses the reference of its orientation and has a internal position system that is relative and imprecise. To extract information of the environment where the drone is, the only reliable way is to use the the image of the cameras. As the drone will always have a nearly constant distant of the ground, the most suitable camera is the bottom camera, that captures images of the ground.

To create this system, we will use “tags”, that are special markers. These markers are recognizable in the image of the bottom camera. When we find a tag in the image of the camera, we will use the position and orientation of the tag to calculate the position and orientation of the drone in the space. To find the tags, we will use some image processing algorithms.

The position system of the autopilot software in the environment will be fixed and will have two axis: X and Y. When we find a tag, we will extract the absolute position of the tag in the environment. That means that if one tag is placed in the position X=100 and Y=200 in the environment, we need to extract from the tag the pair X=100, Y=200. Also, the unit of the system will be millimeters, because all measures of length sent by the drone are in millimeters.

The tasks of this layer are: locate tags and extract the information of them in the image streamed from the drone, calculate the position and the orientation of the drone using the extracted information and merge the calculated data with the other navigation data sent by the drone, providing a single source of flying information about the drone. Moreover, as a separated task, this layer will stock the flight plan that will be used after by the autopilot.

The task of localize and extract information from tags in an image will be discussed in the section 4.1, “The Tag Detector Module”. The task of calculate the drone position and orientation using the extracted information will be discussed in the section 4.2, “The Position Calculator Module”. The task of merge the calculated data with other navigation data sent by the drone will be discussed in the section 4.3, “The Drone Current Data Module”. The task of stock the flight plan will be discussed in the section 4.4, “The Flight Plan Module”.

4.1 The Tag Detector Module

This module will detect tags (special markers) in the image of the bottom camera

sent by the drone. In the next subsections, will be presented the research of available libraries found to do this task, the design of the tags and the procedure used to find the tags in an image.

4.1.1 Research of Available Libraries

Many libraries that could do the task of find a marker in an image were found in the Internet. In this section, we will talk about these libraries.

The library Zxing (2011) is an open-source project of image processing of 1D/2D barcodes. The idea was to use the barcodes as tags and process the images of the bottom camera using this library. Many tests have been done using this library, and the conclusion was that it could not be used, because of the low quality of the bottom camera (the resolution of the video is 176x144 pixels). To recognize any type of barcode at 1 meter of distance from the bottom camera, the barcode would need to be very big.

The libraries GOCR (2010), Tesseract-OCR (2010) and Asprise (2007) can do recognition of texts in images. The tags could be some letters in the ground that we could extract some information. The main problems are: the libraries cannot automatically detect a rotated text and, one more time, the low quality of the bottom camera, because these software need a good resolution in the input image. Moreover, Asprise is not free of cost and GOCR and Tesseract-OCR are not written in Java (see section 1.2, “Objectives”: this work will be written entirely in Java).

The libraries Camellia (2008), OpenCV (2010) and OpenSurf (2010) are well-known open-source projects of computer vision. But all of them are written in C/C++ and have their own formats of image. It was verified that do the interface between Java and C/C++ and do the conversion between the image formats would take a lot of time. Moreover, there is always the risk that the library not work with a such a low resolution camera.

The library ImageJ of Rasband (2011) and the library JavaVis (2005) are open-source projects of image processing and are written in Java. They can find artifacts of images and apply various filters. In particular, a very interesting feature of ImageJ is that it can find circles in a thresholded image, extract information like the position of the center of these circles in the image and their size. Also, it does this very fast. This feature has motivated the creation of a custom design of tags that uses some circles in its design. For more information about ImageJ, see Abramoff (2004). For more information about JavaVis, see Perez (2007).

The decided approach was to create a custom design of tag and a custom algorithm to detect this tag, using in this algorithm the data of circles found by ImageJ after it has processed the image captured in the bottom camera. Doing this, we can have an design of tag as simply as necessary to be detected in a low resolution image.

4.1.2 The Tag Design

The constraints for the design of tags are: the resolution of the bottom camera (only 176x144), the usual distance from the tag to the camera (environ 1 meter) and the size of the tag itself (that must not be too big and must be printable in an ordinary printer). After some tests, the size of the tag was defined as a square of environ 20x20 cm.

A positive feature of the bottom camera is that it has very good detection of brightness and reasonable detection of colors. We can use use this feature to encode

information in the tags.

Thinking in the available resources and in the constraints, the model of the tag created was this: a set of 4 circles, arranged as geometrically as a square, using 5 colors (black, red, green, blue and gray). The figure 4.1 shows a sample of a tag.

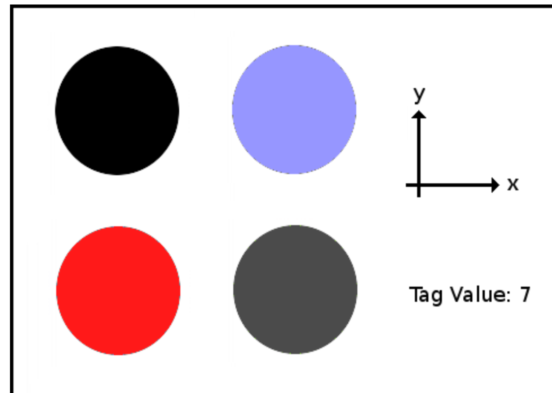


Figure 4.1: The tag model

The tag is the set of four circles. The top-left circle is always black. It has two purposes: determine the tag orientation and be the reference of darkest color. The other 3 circles may have any of these four colors: red, green, blue and gray. They have the purpose of encode the tag value. In the center of the set of four circles, there is a white area that has the size of a circle. The purpose of this area is to be the reference of the lightest color.

The tag value is encoded using 3 circles: the top-right, the bottom-left and the bottom-right ones. Each of these 3 circles can have one of the 4 colors: red, green, blue and gray. Because of this, each of these 3 circles carry 2 bits of information ($\log_2 4 = 2$). Consequently, we have 6 bits of information using these 3 circles (3 times 2 bits). As result, the tag value can be 64 different values ($2^6 = 64$). To decode the tag value, we decode the bits of the top-right circle, the bits of the bottom-left circle and the bits of the bottom-right circle, in this order, and after we concatenate all bits. The value in binary of each color is: “00” for blue, “01” for red, “10” for green and “11” for gray. Considering the tag of the figure 4.1, we have: “00” (top-right circle), “01” (bottom-left circle) and “11” (bottom-right circle). Concatenating these values, the result is:

“00” + “01” + “11” = 000111 = 7,

as said in the image (“Tag Value = 7”).

The tag value is an integer value in range [0 .. 63]. It cannot inform directly the position of the tag. Because of this, the tag value will be used as index in a table of positions, that is stored in the software. For example, if the tag value is 7, we use this value as input in the table and we have: tag.x = table.x[7] and tag.y=table.y[7].

The colors of the tag haven't been chosen randomly. As said before in this section, the bottom camera has very good detection of brightness and will not have problems detecting black, gray and white. The detection of colors is not so good, and many colors has been tested. As result of these tests, we have that the blue and the red colors have been very well detected (probably because of the original YCBCR codification of the video). The green color also have been well detected (but not as well as the red and the blue). The other argument to use the red, blue and green colors is because of the output

format of the decoded image of the video, that is RGB. These three colors are easy to detect in this format.

The tag may be detected in any position by the drone (rotated 180 degrees, for example). But the tag will be placed in the ground always with the correct orientation: the X and Y axis printed in the tag will be always aligned with the X and Y axis of the position system of the autopilot (the X and Y axis of the environment). In other words, if the drone finds a tag rotated 60 degrees in the image of the camera, that means that the drone is rotated -60 degrees compared to the axis of the environment. This is the other information that we will extract from the tag: the angle of rotation of the drone compared to the position system implemented. This is easy to be done, because there is only one the black circle in the tag and it must always be considered as the top-left one. To find the black circle in a tag using a computer, we just need to compare the colors of all the four circles: the darkest one will be the black circle. In short: we will discover the current angle of rotation of the drone analyzing the angle of rotation of the tag in the image captured by the camera. Doing this, we can correct the loss of reference of the rotation angle (this is discussed in the chapter 2).

When we print a tag and see it in a decoded image of the bottom camera, we can notice that the black color isn't exactly black and the white color isn't exactly white, because the light in the environment vary. That is also a reason to always have present in a tag a black circle and a white area: the color of the black circle will be considered the reference of max black value and the color of the white area will be considered the reference of max white value.

As this model of tag allow the indirect storage of his position and the calculation of the current drone rotation angle, it satisfies the needs as spatial source of information.

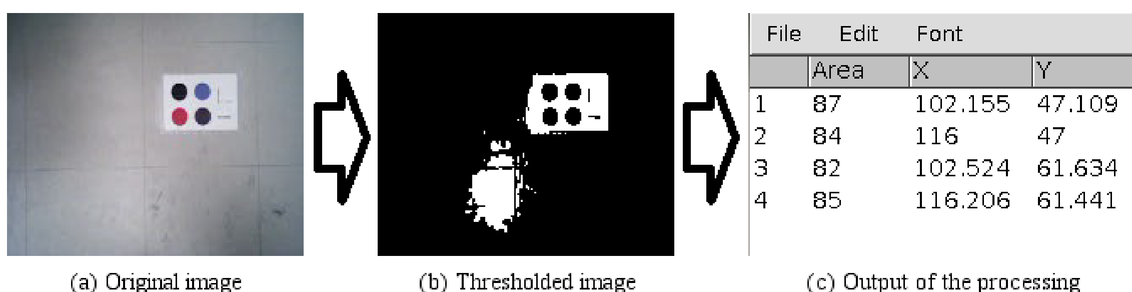
4.1.3 The Procedure to Find Tags

The task of find a tag in an image will be split in three steps. First, we will extract information of circles presents in the image using ImageJ. After, we will process this extracted data and determine if there is one or more tags in the image. Finally, if there is at least one tag present, we will extract the values of the tags.

4.1.3.1 Processing the Image with ImageJ

To find the circles in an image, the Particle Analyzer of ImageJ will be configured to find circles. As it can only find particles in a thresholded image, we first need to threshold the image. For further information of how the Particle Analyzer of ImageJ works, see the chapter 27 in Ferreira (2011).

To better understand the whole process, consider the figure 4.2 below.



	Area	X	Y
1	87	102.155	47.109
2	84	116	47
3	82	102.524	61.634
4	85	116.206	61.441

(a) Original image (b) Thresholded image (c) Output of the processing

Figure 4.2: ImageJ analyze process

The original image (4.2a) is thresholded (4.2b) using another function of ImageJ.

The output of the processing of the thresholded image is shown in the case 4.2c . It informs the area and the X and Y position of the center of each circle found.

4.1.3.2 Processing The Extracted Data From The Image

The available data about the circles in the image is a table that contains the area, the X position of the center and the Y position of the center of each circle. Now, it's necessary to determine if in this set of circles there is one or more tags.

A tag is four circles with the same size that are arranged as a square. To find a tag in a set of circles, we need first to verify if there is any combination of 4 circles that forms a square. To each combination found that forms a square, we need to verify if the four circles have approximately the same size.

To verify if any combination of 4 circles forms a square, it's only necessary to consider the position of the central point of each circle. In short, we will verify if any combination of 4 points in the Cartesian space forms a square. The optimized algorithm created to do this is presented in the Appendix B, named “The Square Finder Algorithm”.

If any combination of 4 circles passes through the two tests (they forms a square and the size of the 4 circles are approximately the same), it goes to the final step: the tag value extraction.

4.1.3.3 Extracting The Tag Value

A tag has been found in the picture and the position of the four circles is available. Now, the value of the tag will be extracted. This will be done in two steps: the first step is collect the colors of the circles and the second step is to determine which circle is the “top-left” (the black circle), which circle is the “top-right” (that carries the most significant bits of the tag value), which circle is the “bottom-left” and which circle is the “bottom-right” (that carries the less significant bits of the tag value). This second step will be called the ordering of the circles. Remember: the tag may be found in the image in any orientation (like upside-down, for example), and this is why it's necessary to do the ordering. With the information of the ordering and of the colors, it's easy to determine the tag value (see section 4.1.2).

To collect the color of a circle, it's necessary to take some pixels inside the circle as samples and calculate the average color value of these pixels. As we know the area of the circle, we can calculate the radius of the circle. With the value of the radius and the Xc,Yc position of the center of circle, we can determine the position of the pixels that are inside of the circle. For example, the pixel with the position X=Xc, Y=Yc+(radius/2) will surely be inside of the circle.

To put the circles in the right order to extract tag value, we will use the position of the black circle. In short, what must be done is to name the three other circles (as “top-right”, “bottom-left” and “bottom-right”) taking as reference the black circle (the “top-left” circle). To do this, the only information available is the positions of the central point of the circles. The algorithm that does this is presented in the Appendix C, “The Points Ordering Algorithm”. Doing the ordering, we know what information each circle carries and the value of the tag can be determined as was explained in the section 4.1.2.

4.2 The Position Calculator Module

This module will calculate the drone position in real-time. When one tag is found (by the Tag Detector Module), this module receives the value extracted and the center position of the tag in the image and calculates the exact position of the drone in the space. This procedure is discussed in the section 4.2.1. When there is no tag in the video, this module uses the real-time data sent by the drone and the data collected in the last time that one tag was found. This procedure is discussed in the section 4.2.2.

4.2.1 Calculating The Position Using A Tag

When one or more tags are found in the video, the Position Calculator module receives a list. This list contains data about all the tags found. What the Position Calculator does is: it chooses one of this tags (usually the tag closest to the center of the image, but any tag can be used), calculates the distance from the drone to the tag and adds this distance to the position coded in the tag (that is referenced by its value).

To calculate the distance from the drone to the tag we will use the camera angles, the drone angles and the altitude of the drone. The bottom camera of the drone has an diagonal aperture of 64°. Doing some trigonometric calculation, we discover that this camera has 51.62° of horizontal aperture and 43.18° of vertical aperture. The figure 4.3.a shows the capture area of the camera, the figure 4.3.b shows the image captured by the camera and the figures 4.3.c, 4.3.d and 4.3.e show the aperture angles of the camera.

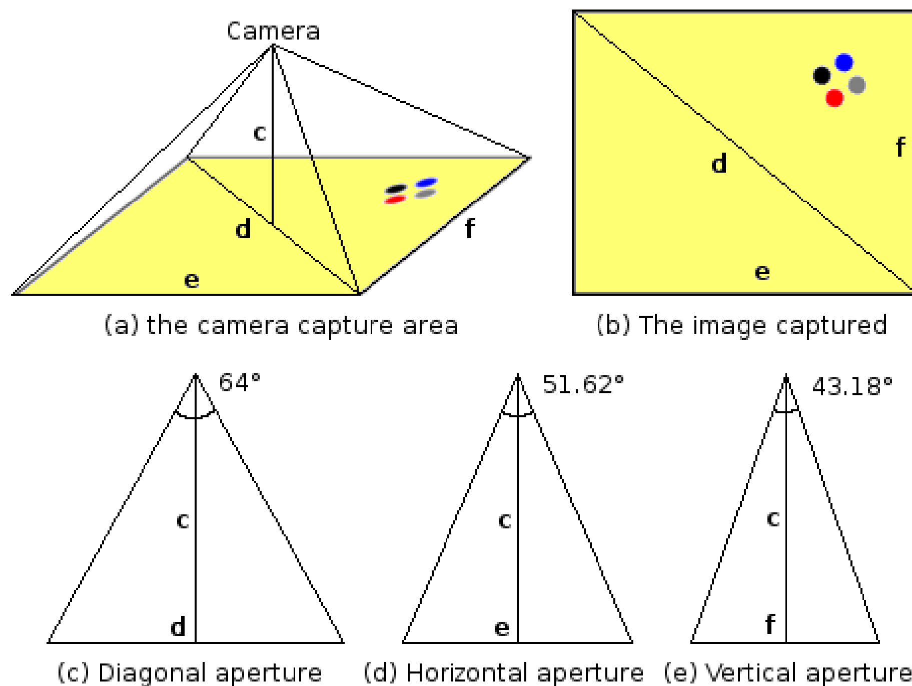


Figure 4.3

The image captured has a resolution of 176x144 pixels. In other words, it has 176 horizontal points and 144 vertical points. The central position of the tag is *px, py*. The value *px* is the horizontal position of the tag in the image and is measured in pixels. The value *py* is the vertical position of the tag in the image and is also measured in pixels. Each point in the image has two angles associated with it: alfa and beta, related to the horizontal and the vertical apertures of the camera. These points and angles are shown

in the figure 4.4.

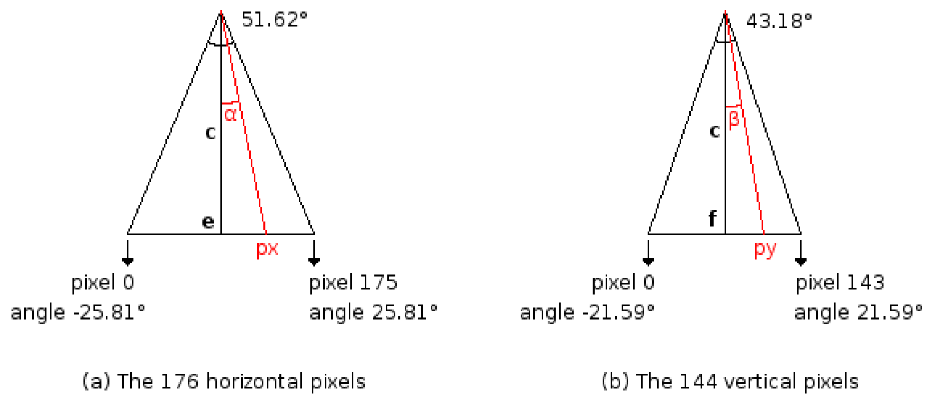


Figure 4.4

The figure 4.4 shows the case when the drone is in a plane parallel to the ground, but this is not always valid. The drone changes the pitch and the roll angles to move in the Y and X axis (see section 1.3.1). This will incline the camera also. The pitch rotate the vertical aperture of the camera and the roll rotate the horizontal aperture of the camera. This effect is shown in the figure 4.5.

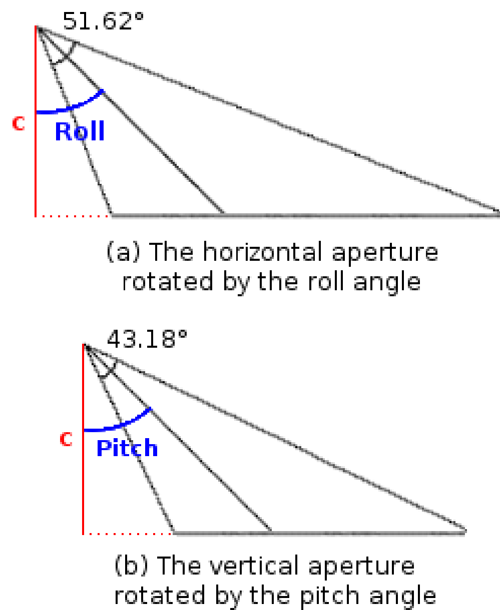


Figure 4.5

Now, to determine the distance from the drone to the tag, we use the equations below. In the equations, *Hd* represents the horizontal distance (in the drone horizontal axis), *Vd* represents the vertical distance (in the drone north-south axis) and *c* is the altitude of the drone (in the drone Z axis). They are all measured in millimeters.

$$Hd = c.\tan(\alpha + roll)$$

$$Vd = c.\tan(\beta + pitch)$$

The tag is always aligned to the position system of the autopilot. As said in the

section 4.1.2, the tag will be placed in the ground always with the correct orientation: the X and Y axis printed in the tag will be always aligned with the X and Y axis of the position system of the autopilot (the X and Y axis of the environment). This means that if the drone finds a tag rotated 60 degrees in the image of the camera, that means that the drone is rotated -60 degrees compared to the axis of the environment. The distances Hd and Vd were calculated using the horizontal and vertical axis of the drone as reference. If this reference is not aligned with the position system of the autopilot, we need to rotate Hd and Vd in the plane XY. The rotated value of Hd will be called DeltaX and the rotated value of Vd will be called DeltaY.

To do the rotation, it's necessary to find the rotation angle of the tag in the image. This is easy to be done, it's only necessary to know the position of the "top-left" circle (the black circle) and the "top-right" circle. With their positions, we calculate the angle between the drone horizontal axis and the line formed by the two points. This angle will be called μ and is shown in the figure 4.6.

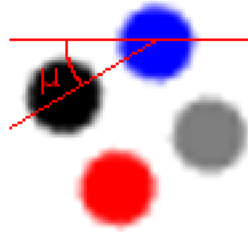


Figure 4.6: In this case, $\mu \approx$ -40 degrees

The drone is rotated $-\mu$ compared to the position system of the autopilot (that is implemented using the tags). The distance from the drone to the tag in the position system of the autopilot is defined by:

$$Angle = \arctan(Vd / Hd) + \mu$$

$$DeltaX = \sqrt{Hd^2 + Vd^2} . \cos(Angle)$$

$$DeltaY = \sqrt{Hd^2 + Vd^2} . \sin(Angle)$$

The position of the drone in the position system of the autopilot is:

$$X = TagX - DeltaX$$

$$Y = TagY - DeltaY$$

The variables TagX and TagY are the values X and Y decoded from the tag value.

The variable $-\mu$ is the true drone Yaw angle in the position system of the autopilot. Its value will be used correct the loss of reference problem discussed in the chapter 2.

4.2.2 Calculating The Position Without A Tag

The values of position and orientation can be calculated when there is no tag in the video, using the navigation data sent from the drone. But to do this, it's necessary that the drone has seen one tag at least once before, because the values calculated using the tag will be used.

The value of Yaw received now from the drone is compared to the value of Yaw received from the drone during the last time that one tag was found. The difference of

them is added to the value of Yaw deducted from the tag in the last time it was found.

The value of the position X and Y received now from the drone is compared to the value of the position X and Y received during the last time that one tag was found. The difference of them will be called dX and dY. But the axis X and Y of the drone are not the same X and Y axis of the position system of the autopilot, because the variable Yaw received from the drone is not the same variable Yaw of the drone in position system of the autopilot (that was deducted from the tag), so the axis of the drone and axis of the position system have some difference. The values of dX and dY must be rotated in the plan XY by the difference in degrees between the two XY planes. This angle difference is the difference between the Yaw deducted from the last time that one tag was found (the value is -μ, as said in the section 4.2.1) and the value of Yaw received from the drone in the last time that one tag was found. This difference will be called here AxisDifference. Finally, the rotated values of dX and dY are called dXrot and dYrot and are defined by:

$$Angle = \arctan(dY / dX) - AxisDifference$$

$$dXrot = \sqrt{dX^2 + dY^2} . \cos(Angle)$$

$$dYrot = \sqrt{dX^2 + dY^2} . \sin(Angle)$$

The position of the drone in the position system of the autopilot is:

$$X = TagX + dXrot$$

$$Y = TagY + dYrot$$

The variables TagX and TagY are values X and Y decoded from the tag value in the last time that the tag was found.

4.3 The Drone Current Data Module

This module has a single purpose: be the only source of information about the drone to all the layers of the system that are above of the second layer. What this module does is: it receives the output of the Navigation Data Receiver module (see section 3.1) and overwrites the variables that were recalculated by the Position Calculator module (see section 4.2). These variable are: the X position of the drone in the position system of the autopilot, the Y position of the drone in the position system of the autopilot and Yaw angle of the drone in the position system of the autopilot.

4.4 The Flight Plan Module

This module contains the flight plan of the autopilot and provides this plan to the other modules. The flight plan is simply an queue of point in the XY plane. The first point in the queue is the first destination, the second point is the second destination, and so on. This module encapsulates this queue and control the addition and the removal of destinations in the flight plan. Also, it can inform the other modules when the flight plan changes.

5 AUTOPILOT WITH DIASPEC

In this chapter, the autopilot is presented. The control system is presented in the section 5.1. The DiaSpec, that is the tool used to design the autopilot system, is presented in the section 5.2. The architecture of the autopilot system (created using DiaSpec) and its implementation is presented in the section 5.3.

5.1 Autopilot Control System

In this section, it's described the control system used to create the autopilot. In the subsection 5.1.1, it's described the state machine and the monitor that control the drone. In the subsection 5.1.2, it's presented how the drone acceleration and inertial velocity are controlled.

The drone itself can move in any direction and never need to rotate to move to another location. But the created autopilot works like an autopilot of an helicopter: it rotates towards the right direction before of move. The reason to this is that making the autopilot this way it's more stable to operate in small places, like rooms in buildings.

5.1.1 The Autopilot State Machine and The Autopilot Monitor

The drone, while it is flying, is controlled using a state machine. It has only 5 states. These states and the events that make the changing from one state to another is illustrated in the figure 5.1.

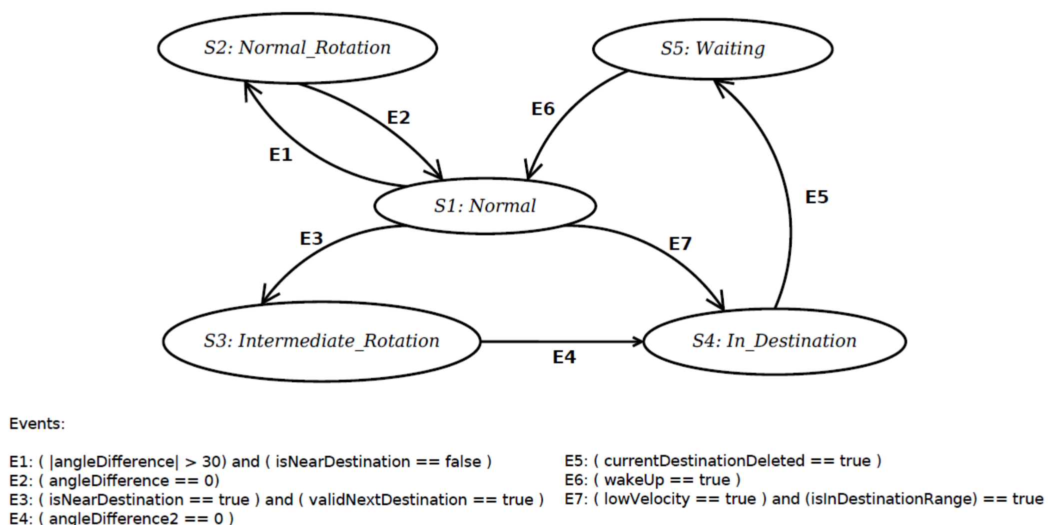


Figure 5.1

The autopilot control system has a autopilot monitor that will be always monitoring the flight plan and the drone state. This autopilot monitor controls globally the state machine. If the flight plan is empty and the drone state is "flying", it automatically sends the command "land" and turn off the state machine from the figure 5.1. If the flight plan is not empty and the drone state is "landed", it automatically sends the command "take off" and turn on the state machine from the figure 5.1. In other words, as already said, this state machine only is turned on while the drone is flying. The commands "take off" and "land" are sent by the autopilot monitor, that can turn on and off the state machine of the figure 5.1. This separation makes easy the manipulation of the autopilot by the user interface: the user interface sends a message to the monitor if it wants the autopilot to be turned on or be turned off, and the monitor will take care of all the necessary procedures. Another task of the autopilot monitor is to send periodically to the user interface some information about the autopilot, like what it's doing.

The only task of the state machine illustrated in the figure 5.1 is to achieve the current destination in the flight plan and after delete this destination. It will do this and enter in the "Waiting" state. The autopilot monitor will be reported by the flight plan that it has changed (the current destination was deleted). If there is any remaining destinations in the flight plan, the autopilot monitor will send a "wake-up" message to the state machine, causing the state machine to change from the state "Waiting" to the state "Normal" (this causes the state machine to restart). If the flight plan is empty, the autopilot monitor sends a "land" command to the drone and will turn off the state machine, that was in the "Waiting" state.

To understand how the autopilot state machine works, consider the following scenario: the flight plan has two destinations, the first is A and the second is B. The user interface turns on the autopilot. The autopilot monitor verifies that the flight plan is not empty and does two actions: send the "take-off" command to the drone and turns on the autopilot state machine. The state machine starts in the state *Normal*. This state only does something after the end of the "take-off" procedure.

The procedure "take off" ended and the drone is flying. In the *Normal* state, the drone will verify if its north direction is aligned with the direction of the current destination (the current destination is A). The difference between the north direction and the direction of the current destination is bigger than 30 degrees (|angleDifference| > 30º) and the distance from the drone to the current destination is bigger than 0.5 meters (isNearDestination == false). This causes the event E1 and the state machine changes to the state *Normal_Rotation*. In this state, the drone will rotate until its north direction and the current destination direction be aligned (angleDifference == 0º). When this occurs, it causes the event E2 and the state machine switches to the *Normal* state.

In the *Normal* state, the north direction of the drone and direction of the current destination are aligned, or nearly aligned. The drone only needs to go forward to achieve its destination, and eventually a little to the left or to the right. While in *Normal* state, the state machine sends commands to the drone to make it move in the north-south axis and to the left-right axis. The direction of the movement in each axis is defined by the distance from the drone to the destination in each axis. For example, if the distance from the drone to the destination in the north-south axis is positive, the drone must move forward. If the distance from the drone to the destination in the left-right axis is negative, the drone must move to the left. It is important to notice that the drone make movements in both axis at the same time (going forward and leftward at the same time, for example). The acceleration and the velocity in each of these axis is also defined by the distance in each axis and this will be discussed in the section 5.1.2.

The drone is moving toward the current destination (that is A) and the state machine continues in the state *Normal*. At some moment, the distance from the drone to the current destination is lower than 0.5 meters (isNearDestination == true) and the current destination is not the final destination of the flight plan, because there is B (validNextDestination == true). This causes the event E3 and the state machine switches to the state *Intermediate_Rotation*.

As the current destination is just an intermediary point in the flight plan, it's not necessary to go in the exact position of the current position. The state *Intermediary_Rotation* means "drone rotation in an intermediary destination of the flight plan". In this state, the drone will rotate to align its north direction with the direction of the next destination (the point B), while will still moving to the current destination (the point A). In other words, the drone will rotate toward the next destination while it stay near to the current destination of the flight plan. When the north direction is aligned (angleDifference2 == 0), this causes the event E4 and the drone state machine switches to state *In_Destination*.

The state *In_Destination* is a transitory state. It sends a message to the flight plan module to delete the current destination. When this is done (currentDestinationDeleted == true), this causes the event E5 and the state machine switches to the state *Waiting*.

In the state *Waiting*, the state machine will wait a "wake-up" message from autopilot monitor. As the flight plan was changed, the autopilot monitor will verify the flight plan. As the flight plan is not empty, it sends a "wake-up" message to the state machine. When this occurs (wakeUp == true), this cause the event E6 and the state machine swiches to the *Normal* state.

In the *Normal*, the state will verify its north direction is aligned with direction of the current destination (that now is B), and may change to the state *Normal_Rotation*, as already explained before. Continuing in the *Normal* state, the drone will move toward the current destination. This time the event E3 will not occur, because there is no next destination after the current (validNextDestination == false). The drone will stay in the *Normal* state moving toward the current destination. When the distance from the current destination is lower than 0.25 meters (isInDestinationRange == true) and the velocity of the drone is low enough to consider the drone stopped in the air (lowVelocity == true), these causes the event E7. Then the state machine will switch to the *In_Destination* state, will delete the current destination, and the state machine will switch to the state *Waiting*. As the flight plan has changed, the autopilot monitor will verify the flight plan. The flight plan will be empty and the drone monitor will send the command "land" to the drone and will turn off the state machine, that is in the *Waiting* state.

5.1.2 The Autopilot Acceleration Calculator

In this section is presented a little module used by the autopilot state machine to calculate the acceleration of the movements of the drone, the autopilot velocity calculator.

Before talking about the velocity calculator, its necessary to talk about the objectives of the present work. Make an high-performance and fast autopilot for the drone is a complex task and is not the objective of the autopilot presented here. It may involve measuring the time response of the sensors of the drone, study in detail the quality of the inertial unit present in the drone, create complex control loops based in differential equations, among other things. It may consume a lot of time and may ever not be

possible to do: as already said before, the drone sensors do not have a high quality. Instead of this, the objective of the autopilot described in the current work is to create a stable and simple autopilot, that demonstrates a implementation created using DiaSpec working in a real device and that works well. To achieve this objective, we will not permit the drone to have high inertial velocities and to do fast movements.

Through several tests with the drove, it was verified that with a inertial speed limit of 700 mm/s the drone can cancel the inertial velocity accumulated without causing too much oscillations, that could make the flight unstable. With an acceleration limit of 300 mm/s^2 the drone can move in any direction without acquiring significant inertial velocity too fast, that could cause also oscillations. These values were taken as limit values for the autopilot velocity calculator.

To move in the left-right axis and in the north-south axis, the drone changes the values of the roll and pitch angles. When the drone incline itself, *it accelerates toward a direction*. It is very important to understand this: to make the drone move, we are passing a parameter (the angle of inclination) that means an acceleration. The commands go up / go down and rotate are different, as they works with a parameter that means velocity. For more information, see section 1.3.5.2, “The PCMD Commands”.

The equations of acceleration used to control the drone velocity are shown in the figure below. The transformation between acceleration in millimeters by seconds and inclination angles of the drone won't be demonstrated here. This transformation is not necessary to understand how the acceleration calculator works. It is important to notice that the drone moves in both left-right and north-south axis at the same time and that's why there are X and Y accelerations.

```
// First step of the calculation
nearDistance = 500;      // measured in millimeters
maxAcceleration = 300;   // in mm/s²
maxVelocity = 700;       // in mm/s

baseAccelerationX = maxAcceleration;
baseAccelerationY = maxAcceleration;

if (distanceX < nearDistance) {
      baseAccelerationX = maxAcceleration*(distanceX/nearDistance);
}
if (distanceY < nearDistance) {
      baseAccelerationY = maxAcceleration*(distanceY/nearDistance);
}

// Second step of calculation
velocityDecrementX = maxAcceleration*(velocityX/maxVelocity);
velocityDecrementY = maxAcceleration*(velocityY/maxVelocity);

accelerationX = baseAccelerationX - velocityDecrementX;
accelerationY = baseAccelerationY - velocityDecrementY;
```

Figure 5.2

To explain the calculation of the acceleration, we will split it in two steps. In the first step, the base value of the acceleration is calculated. In the second step, the final acceleration is calculated.

In the first step, the value of the base acceleration is always equal to max value. But

if the drone is near of the destination (“distance < nearDistance”) the drone must reduce the acceleration. Near of the destination, the acceleration will be reduce linearly (it will be a fraction of the max value, as the value of “distance/nearDistance” will always be between 0 and 1).

In the second step, the acceleration of the drone must be reduced or augmented if the drone already has an inertial velocity. For example, if the base acceleration is rightward at the max acceleration but the drone already is moving to the right at the max velocity, the resulting acceleration must be zero. In other case, if the base acceleration is rightward at the max acceleration but the drone is moving to the left at the max velocity, the final acceleration will be two times the max velocity rightward. Accelerations above the max value are only permitted to stop the drone inertial velocity.

There is also a special case. When the autopilot state machine is in the state *Normal_Rotation*, the drone must rotate in the air while stopped in the left-right and north-south axis. When the state machine is in this state, it sends a flag to the acceleration calculator. This flag makes the acceleration calculator output values to try to stop completely the inertial velocity of the drone. The equations to do this are shown in the figure 5.3.

```
maxAcceleration = 300;  // in mm/s²
maxVelocity = 700;      // in mm/s

accelerationX = (-velocityX / maxVelocity) * maxAcceleration;
accelerationY = (-velocityY / maxVelocity) * maxAcceleration;
```

Figure 5.3

The values of acceleration calculated in the figure 5.3 are always against the linear velocity of the drone.

5.2 DiaSpec Description

DiaSpec is a tool developed by the INRIA Phoenix research group affiliated with the University of Bordeaux I, LaBRI. This tool allows the automatic generation of a programming framework through the specification of the target system architecture. The specification of the system architecture is done using the DiaSpec description language.

The DiaSpec tool contains a compiler that takes as input a specification described in the DiaSpec description language and generates a set of code source files that contain the abstraction of each module from the specification and the implementation of the relationship between the modules described in the specification. The programmer of the system need to code the behavior of each module, but not the relationship between the modules, because DiaSpec already has generated it.

The modules of the architecture described in the specification are divided in three types: devices, contexts and controllers. Each device may make available sources of data. Also, each device has actions that it can do.

The contexts are like functions: it takes some data as input, do some calculation and output a result. The controllers are the only modules that can active actions in the devices. The figure 5.4 shows the interactions between devices, contexts and controllers.

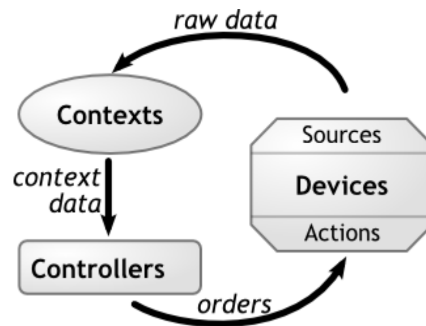


Figure 5.4

To illustrate the description of an architecture using DiaSpec, consider the system shown in the figure 5.5.

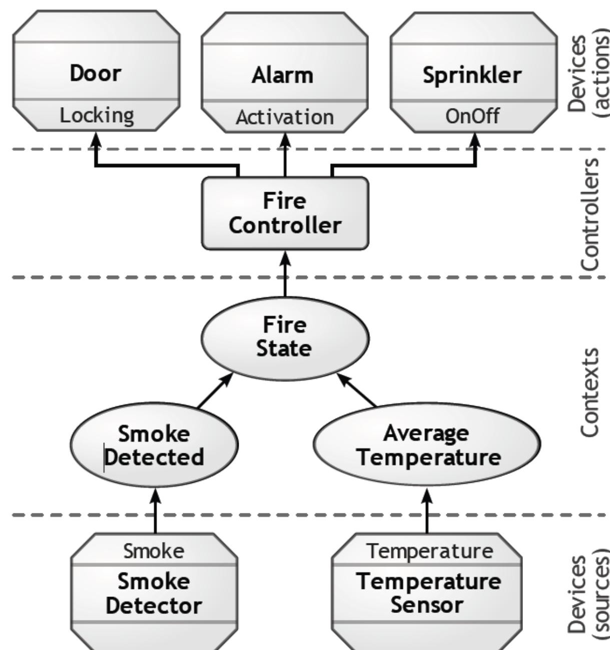


Figure 5.5

The figure 5.5 shows the architecture of a fire management system. The devices Smoke Detector and Temperature Sensor make available sources of data about the smoke and the temperature in the place. The contexts Smoke Detected and Average Temperature process the data from the sources and output a conclusion to the context Fire State. The context Fire State activates the controller Fire Controller. The controller Fire Controller do the right action in the devices Door, Alarm and Sprinkler.

Some application domains of DiaSpec are: multimedia communication services, home/building automation and avionics (e.g., flight management). A complete description of DiaSpec can be found in Cassou (2009).

5.3 Autopilot Architecture

The figure 5.6 shows the architecture of the drone autopilot described using DiaSpec language.

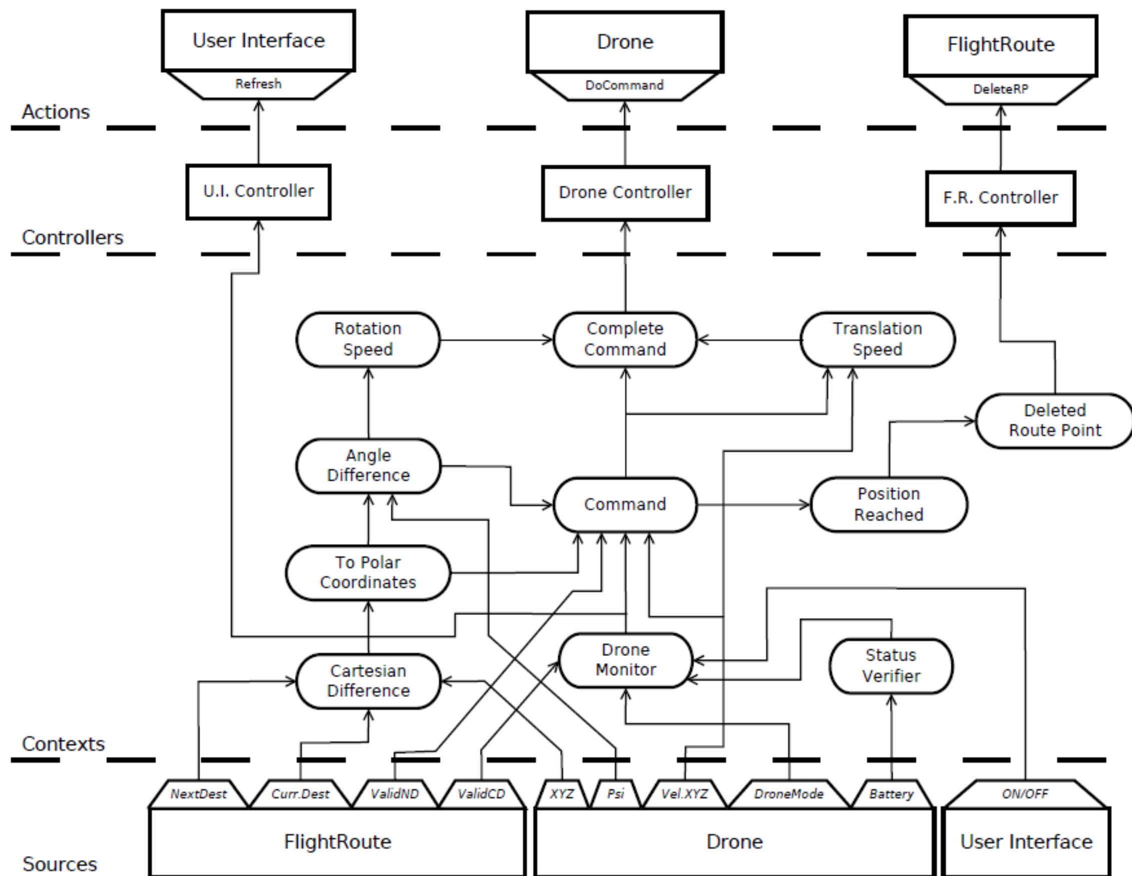


Figure 5.6

This architecture is data-flow oriented. The diagram shows all the used data and where it is used. There is three devices: Flight Route, Drone and User Interface.

The Flight Route device has four sources: NextDest (the destination after the current destination), CurrDest (current destination), ValidND (boolean variable that says if NextDest is valid) and ValidCD (boolean variable that says if CurrDest is valid). These sources come from the Flight Plan module (see section 4.4). The Drone device has five sources: XYZ (the position of the drone in the space), Psi (the value of the Yaw angle of the drone), VelXYZ (the current velocity of the drone), DroneMode (the current mode of the drone, like “taking off”, “landing” or “flying”) and Battery (the battery level). These sources come from the Drone Current Data module (see section 4.3). The module User Interface has only one source: ON/OFF (the switch that turn the autopilot on and off).

The context Drone Monitor contains the implementation of the autopilot monitor discussed in the section 5.1.1. It monitors the current state of the drone, checks if the flight plan is empty, if the battery low and if the user interface turned on or off the autopilot. As output, this context sends the commands “take off” and “land” (that passes through the context Command), it turns on and off the autopilot state machine (that is in the context Command), it send “wake up” massages to the autopilot state machine, it stops and land the drone when the battery level is low and it sends some information to the user interface about what the autopilot is doing. The context Status Verifier just verifies if the battery level is low and outputs a conclusion to the context Drone Monitor.

The contexts Cartesian Difference, To Polar Coordinates and Angle Difference only

do calculations. The context Cartesian Difference calculates the difference vector between the position of the drone and the current destination of the flight route and the difference vector between the position of the drone and the next destination (the destination after the current). The context To Polar Coordinates transform the vectors of Cartesian Difference from the cartesian coordinates system to the polar coordinates system. Each vector is represented in the polar coordinates system by a module and an angle. The Angle Difference calculates the current angle difference between north direction of the drone and the current destination direction. It calculates also the current angle difference between north direction of the drone and the next destination direction. To do this, the Angle Difference context takes as input the angles outputted by the To Polar Coordinates context and the source Psi of the drone (that is the Yaw angle).

The context Command contains the implementation of the state machine discussed in the section 5.1.1. This is why it needs as input the information about if the next destination is valid (the source ValidND), the module of the distance between the drone and the current and next destinations (outputted by the context To Polar Coordinates), the angle difference between the drone and the current and next destinations (outputted by the context Angle Difference), the current velocity of the drone (source VelXYZ) and the output of the context Drone Monitor (that controls globally the state machine). As output, this chooses the right command to send to the drone and sends to the context Translation Speed the current distance from the drone to the current destination in the drone horizontal and north-south axis (that rarely correspond to the X and Y axis of the position system).

The context Translation Speed contains the implementation of the autopilot acceleration calculator discussed in the section 5.1.2. This is why it takes as input the current drone velocity (the source VelXYZ) and the distance from the drone to the destination outputted by the context Command. As output, it sends the values of the forward-backward acceleration and leftward-rightward acceleration.

The context Rotation Speed defines the rotation speed of the drone. To do this, it takes as input the two values of angle outputted by the context Angle Difference. The drone only rotates when the autopilot needs to align its north direction with the direction of the current or next destinations. So, for example, if angle difference between the north direction and the current destination is positive, the drone needs to turn right. If it is negative, the drone need to turn left. Moreover, the drone accepts as parameter of rotation a value of angular speed. This makes the control much easier: to make the drone rotate right, the parameter must be positive; to make the drone rotate left, the parameter must be negative; to make the drone stop rotating, we send the value zero as parameter. If the absolute value of the parameter is high, the drone rotates fast. If it is low, the drone rotates slowly. There is no "inertial angular velocity" to control, as in the case of the forward-backward and leftward-rightward movements. The context Rotation Speed context output two values of velocity: one if the drone need to rotate toward the current destination, and another if the drone needs to rotate toward the next destination.

The context Complete Command takes as input the values outputted by the contexts Rotation Speed and Translation Speed and the type of command chosen by the context command. As output, it sends to the Drone Controller the complete specification of the command that must be executed by the drone.

This autopilot architecture works in a completely reactive way: the actions are activated by the sources. The contexts are activated every time that they receive a new value of input. As new values for the sources of the device Drone are published every

30ms, the autopilot system will output a new command to the drone every 30ms. This is the recommended frequency of new commands for the Parrot A.R. Drone.

6 CONCLUSION

The creation of the drone autopilot involved many challenges. The first major one was the creation of the decoder for the video stream sent by the drone. The implementation of a decoder must be done very carefully, because any little error will ruin the result. Moreover, the entropy encoding and the DCT steps in the video image decoding are not easily understandable. The second major challenge was to create a way of extract the position of the drone in the space using the image from a camera and special markers. The calculation of the distance between the drone and the special marker involved a lot of trigonometric calculations. The detection of the special marker involved the creation of customized image processing algorithms. Finally, the third major challenge was to create control loops for the autopilot itself.

The system described in this work was successfully implemented in Java. The implementation is completely object oriented and permitted the author of this work to improve his programming skills.

During the stage period, many tests and demonstration of the autopilot system have been done. The first major demonstration was after the implementation of the first and second layers of the system. It was demonstrated to the internship tutor the drone flying (manually controlled) while sending its position in real-time, using also the special markers (the tags) for this. The second major demonstration was after the complete implementation of the system. It was demonstrated to the internship tutor and to the research director of INRIA the drone flying in a predefined route while being entirely controlled by the autopilot.

The drone autopilot system developed shows also power of the DiaSpec tool. A architecture described in DiaSpec is easily understandable and flexible. The implementation of the autopilot was faster because the DiaSpec tool generated a lot of code automatically.

A video has been created to demonstrate the autopilot working. It can be found in the link: http://www.youtube.com/watch?v=9l8tRbya4vU .

REFERENCES

CASSOU, D. et al. **A Generative Programming Approach to Developing Pervasive Computing Systems**. Proceedings of the 8th international conference on Generative Programming and Component Engineering, Denver (United States), ACM, pp. 137-146, oct. 2009.

PEREZ, J.M. et al. **JavaVis: open source code for computer vision subjects**. Proceedings of the First Free/Libre/Open Source Systems International Conference (FLOSS 2007), Cádiz (Spain), pp. 106-115, Mar 2007.

ABRAMOFF, M.D.; MAGALHAES, P.J.; RAM, S.J. **Image Processing with ImageJ**. Biophotonics International, volume 11, issue 7, pp. 36-42, 2004.

FERREIRA, T.; RASBAND, W. **ImageJ User Guide IJ 1.45m**. [S.l.:s.n], aug 2011. Available at: http://rsbweb.nih.gov/ij/docs/index.html. Last access: nov. 2011.

PISKORSKI, S.; BRULEZ, N. **AR. Drone Developer Guide SDK 1.6**. [S.l.:s.n], feb. 2011. Available at: http://projects.ardrone.org/. Last access: jun. 2011.

RASBAND, W. **ImageJ library**. National Institutes of Health, United States, v1.45c, Mar. 2011. Available at: http://rsbweb.nih.gov/ij/download.html. Last access: nov. 2011.

JAVAVIS library. [S.l.:s.n], v3.51, Dec. 2005. Available at: http://sourceforge.net/projects/javavis/. Last access: nov. 2011.

ZXING library. [S.l.:s.n], v1.7, June 2011. Available at: http://code.google.com/p/zxing/. Last access: nov. 2011.

GOCR library. [S.l.:s.n], v0.49, Set. 2010. Available at: http://jocr.sourceforge.net/. Last access: nov. 2011.

TESSERACT-OCR library. [S.l.:s.n], v3.0, Nov. 2010. Available at: http://code.google.com/p/tesseract-ocr/. Last access: nov. 2011.

ASPRISE OCR SDK library. [S.l.:s.n], v4.0, Nov. 2007. Available at: http://asprise.com/product/ocr/selector.php. Last access: nov. 2011.

CAMELLIA library. [S.l.:s.n], v2.7.0, Jan. 2008. Available at: http://camellia.sourceforge.net/. Last access: nov. 2011.

OPENCV library. [S.l.:s.n], v2.2, Dec. 2010. Available at: http://opencv.willowgarage.com/wiki/. Last access: nov. 2011.

OPENSURF library. [S.l.:s.n], v18/03/2010, Mar. 2010. Available at: http://code.google.com/p/opensurf1/. Last access: nov. 2011.

APPENDIX A DRONE TECHNICAL SPECIFICATIONS

The specifications of the A.R. Drone from Parrot:

Embedded Computer System
* ARM9 468 MHz
* DDR 128 MB at 200MHz
* Wifi b/g
* USB high speed
* Linux OS

Inertial Guidance Systems
* 3 axis accelerometer
* 2 axis gyrometer
* 1 axis yaw precision gyrometer

Physical Features
* Max running speed: 5 m/s; 18 km/h
* Weight:
    - 380 g with outdoor hull
    - 420 g with indoor hull

Safety System
* EPP hull for indoor flight
* Automatic locking of propellers in the event of contact
* Control interface with emergency button to stop the motors

Aeronautic Structure
* High-efficiency propellers
* Carbon-fiber tube structure

Motors and Energy
* 4 brushless motors, (35,000 rpm, power: 15W)
* Lithium polymer battery (3 cells, 11,1V, 1000 mAh)
* Discharge capacity: 10C
* Battery charging time: 90 minutes
* Max flying time (battery capacity): about 12 minutes

Front Camera: Wide Angle Camera

* 93° wide-angle diagonal lens camera, CMOS sensor
* Camera resolution 640x480 pixels (VGA)

Ultrasound Altimeter

* Emission frequency: 40kHz
* Range 6 meters vertical stabilization

Vertical Camera: High Speed Camera

* 64° diagonal lens, CMOS sensor
* Video frequency: 60 fps
* Allows stabilization even with a light wind

APPENDIX B THE SQUARE FINDER ALGORITHM

This appendix shows a way to determine, in a set of points in the Cartesian space, how many combinations of four points forms a square. To find a square, we will use the following rule:

Consider four points in a bi-dimensional space: A, B, C and D. Consider the following distances: AB (from A to B), BC (from B to C), CD (from C to D), DA (from D to A), AC (from A to C) and BD (from B to D). The points A, B, C and D are in different positions in the space (AB, BC, CD, DA, AC and BD have non-zero positive values). If AB=BC=CD=DA and AC=BD, these points are arranged as a square in this space. Moreover, in this case, the condition AC=BD>AB=BC=CD=DA is always true.

The two conditions AB=BC=CD=DA and AC=BD and the supposition that the points are in different positions are enough to determine a square in a ideal case, but we will accept approximations of this (AB≈BC≈CD≈DA and AC≈BD). The use of approximations may introduce some bizarre cases. The figure B.1 shows one of these cases.

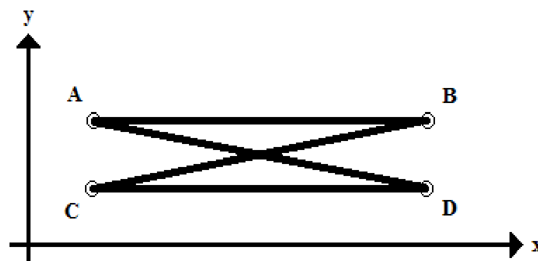


Figure B.1: AB≈BC≈CD≈DA and AC≈BD, but it's not arranged as a square

The avoid them, we will use the condition *AC=BD>AB=BC=CD=DA*. The algorithm that implement this logic is shown below.

```
Compute the distances between all points
Compute all the combination of 4 points
For each combination {
      // Consider A, B, C and D the points of the combination
      if ((AC≈BD) and (AB≈BC≈CD≈DA)) and
           (average_value(AC,BD) > average_value(AB,BC,CD,DA)) {
                  Add the combination to the list of squares
      }
}
```

Figure B.2: The algorithm to find squares

APPENDIX C THE POINTS ORDERING ALGORITHM

This appendix shows a way to determine, in a set of four points arranged as a square, what is the relative position of three points when we take one point as reference. To better understand the problem, consider the image C.1.

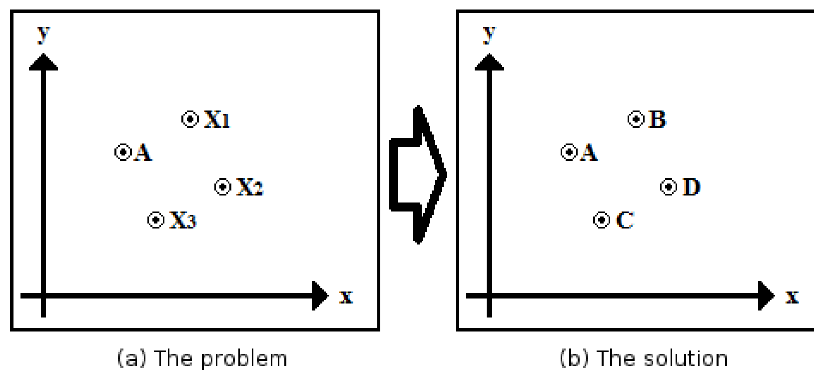


Figure C.1

The point A is the reference point. The point D will always be the point more distant of A. The point B will be the first point found when we search from A to D in a clockwise way. The point C is the first point found when we search from A to D in a anti-clockwise way. For a person, it's very easy to determine visually which is the point B, C and D when the point A is known, even when the figure is rotated. But how a computer do the same thing only knowing the position of A and the position of the other three unnamed points? Also, the computer must do this in a way as simple as possible. The algorithm that solves the problem is presented in the figure C.2. It is very optimized and avoid problematic singularities. It will be used to aid the decoding of a tag (see sections 4.1.2 and 4.1.3).

```
Compute the distances between A and the three points
Consider D the point with the biggest distance
Calculate deltaX = |xD - xA| and deltaY = |yD - yA|
If (deltaX > deltaY) {
      If (xD > xA) {
            Consider B the remaining point with the higher y
      }
      Else {
            Consider B the remaining point with the lower y
      }
}
Else {
      If (yD > yA) {
            Consider B the remaining point with the lower x
      }
      Else {
            Consider B the remaining point with the higher x
      }
}
Consider C the remaining point
```

Figure C.2: The points ordering algorithm

APPENDIX D DESCRIPTION OF THE WORK IN PORTUGUESE

Este trabalho consiste em um piloto automático implementado em Java para o A.R. Drone da fabricante Parrot. Este piloto automático foi desenvolvido no laboratório INRIA de Talence (França). Na implementação, foi usado o DiaSpec, que é uma ferramenta de geração automática de código a partir de um diagrama. O DiaSpec foi desenvolvido pela equipe Phoenix do laboratório INRIA de Talence.

O drone utilizado foi o A.R.Drone, que é um quadricóptero. Este quadricóptero é controlado remotamente através de uma conexão Wi-Fi em modo *ad-hoc*. Ele pode executar três procedimentos automáticos: decolar, aterrissar e pairar (o drone toma um ponto do solo como referência e tenta permanecer na mesma posição no espaço). O drone possui sensores que informam a orientação espacial do drone (através dos ângulos Pitch, Roll e Yaw) e a distância entre o drone e o chão. Há duas câmeras no drone: uma horizontal e uma vertical voltada para o chão.

A abordagem adotada para construir este piloto automático foi a de uma modelo de software em camadas. Este software, denominado "Drone Autopilot System", será executado em um computador (que daqui para diante será chamado dispositivo controlador) que possui uma máquina virtual java. A figura D.1 ilustra este modelo.

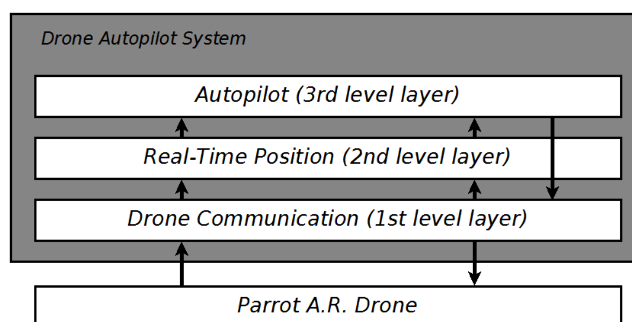


Figura D.1: Modelo em camadas

A primeira camada é a camada de comunicação, que recebe os dados enviados pelo drone e envia comandos. A segunda camada é a camada que calcula continuamente a posição do drone no espaço. A terceira e última camada é o piloto automático em si, que utiliza os recursos disponibilizados pelas outras duas camadas. Nos próximos, parágrafos as camadas deste modelo serão apresentadas com mais detalhes.

A camada "Drone Communication" trata três fluxos de dados: dados de navegação, dados de vídeo e dados de controle. Os dados de navegação são informações enviadas do drone para o dispositivo controlador e compreende o seguinte conjunto de dados:

nível de bateria, orientação do drone no espaço, distância do drone até o solo, posição XYZ do drone no espaço e a velocidade do drone no espaço. Os dados de vídeo são as imagens capturadas por uma das câmeras (o dispositivo controlador pode escolher qual câmera) e é, portanto, um fluxo enviado do drone para o dispositivo controlador. Os dados de controle são os comandos enviados pelo dispositivo controlador para o drone a fim de controlá-lo remotamente. A fim de executar suas funções, a camada "Drone Communication" foi dividida em três módulos: o módulo que recebe os dados de navegação ("Navigation Data Receiver"), um módulo que recebe e decodifica os dados de vídeo ("Video Decoder") e um módulo que envia os comandos para o drone ("Drone Controller").

A camada "Real-Time Position" é a camada que trata de calcular a posição do drone no espaço usando as informações disponibilizadas pela camada de comunicação. Como os sensores do drone têm baixa precisão e com o passar do tempo acumulam erros de medida, foi necessário fazer a recalibragem da posição do drone de tempos em tempos. A solução encontrada foi a recalibragem através de marcadores ("tags") colocados no solo. Estes marcadores aparecem nas imagens capturadas pela câmera vertical do drone e são detectadas no dispositivo controlador através de processamento de imagem. A fim de executar suas funções, a camada "Real-Time Position" foi dividida em quatro módulos: um módulo que detecta tags ("Tag Detector"), um módulo que calcula a posição do drone usando a saída do módulo detector de tags e os dados de navegação ("Position Calculator"), um módulo que contém o plano de vôo ("Flight Plan") e um módulo que contém todas as informações obtidas pela primeira e segunda camadas (Drone Current Data).

A camada "Autopilot" contém o piloto automático e a interface com o usuário. O piloto automático foi construído usando o DiaSpec, que é uma ferramenta que permite que uma descrição de software na forma de diagrama possa ser compilada para gerar código para os componentes do diagrama e para a interação entre os componentes. O piloto automático realiza o plano de vôo (que nada mais é do que uma série de pontos no espaço a serem percorridos) através de uma máquina de estados que decide o comando certo a ser enviado para o drone. A fim de executar suas funções, a camada "Autopilot" é dividida em dois módulos: um módulo de interface com o usuário ("User Interface") e um módulo que é o piloto automático em si (também chamado de "Autopilot").

A figura abaixo mostra os componentes do sistema e suas interações.

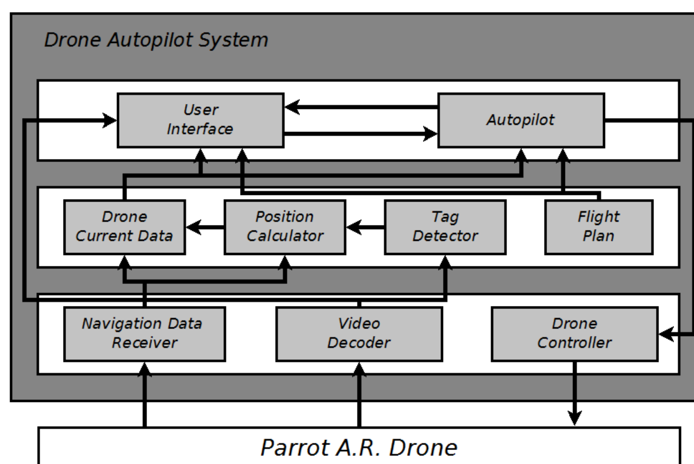


Figura D.2: Módulos do sistema