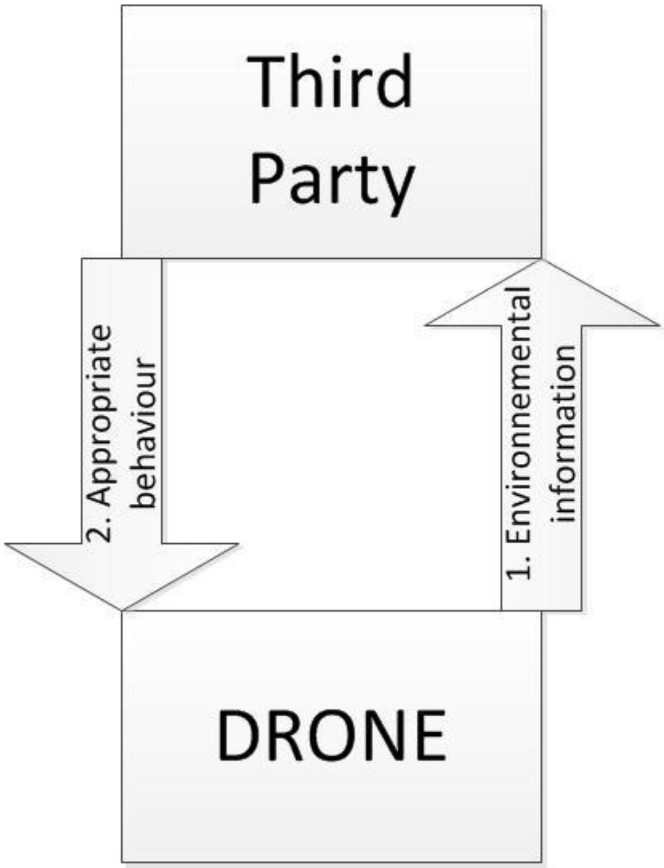
Report : Early Image Processing Tests on UAV (Cont.)

on the 21/02/2103 by Jean de Campredon

Index

I. History of the project

I.a. Main goal

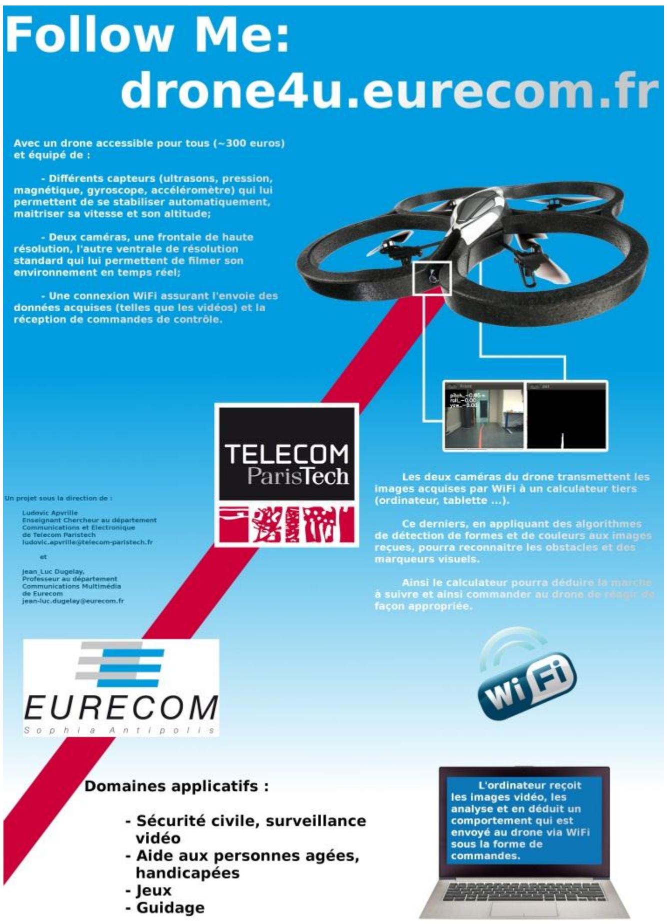


The main idea is to allow drone to auto-navigate thanks to the **environmental information** it can gather by itself.

The drone should be able to transfer those information to a third party that could compute with an **appropriate behaviour** for the drone to follow.

Eventually the third part should send some simple commands to the drone.

In our case the drone is low cost, the environmental are the balance of the drone and the images captured.



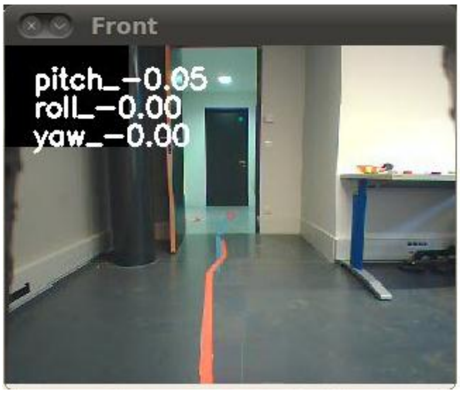
Thus, such a presentation was used during presentations, science fairs and other events we could present our project.

I.b. The pre-existing project

The project was first led on the version 1 of the Drone Parrot.

But Parrot issued recently the version 2 of their Drone. The main differences between those two drones are:

- On the hardware
    - The resolutions of the cameras were multiplied by 4
    - The magnetic sensors allowing the drone to measure its orientation
- On the software
    - A new SDK incompatible with the previous version
    - Communications simplified
    - An intern environment of the drone locked



The previous project issued an auto-pilot capable of:

- Detecting and following a line
- Detecting two symbols
- Turning right
- Turning left

But this project was relying on a weak and limited structure, a heavy image processing and an accumulation of modules unplanned.

II. The mission

Because of the incompatibility of the SDKs and the underlined defaults of the previous project, we decided to restart from “scratch” the project.

Also we define our new mission with two main objectives:

Rebuild the project

- Compatibility with Drone version 2
- Develop a scalable Drone

The functionalities of the new auto-pilot

- Detect and follow lines
- Detect symbols
- Turn right and left
- New imaginative functionalities

III. The actual program

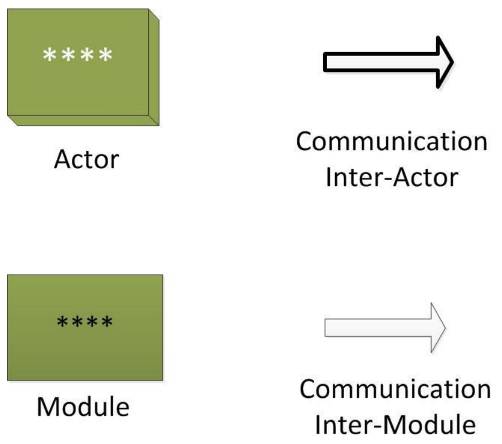
Before starting, we had to fix a guide line to follow. Obviously, the auto-pilot would need multiple module operating more or less separately:

- A center of communication: to communicate with the drone
- A center of image processing: to extract the valuable information from the captured images
- A center of command: To determine the appropriate behaviour
- A user Interface

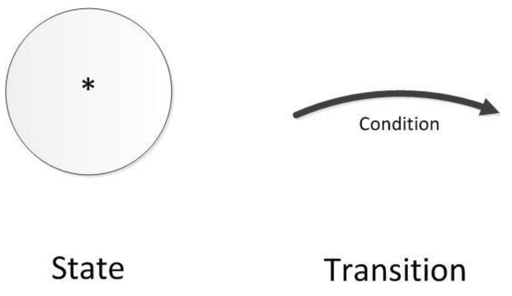
Also we have ended up with the following structure.

N.B: The following schema can be explained by

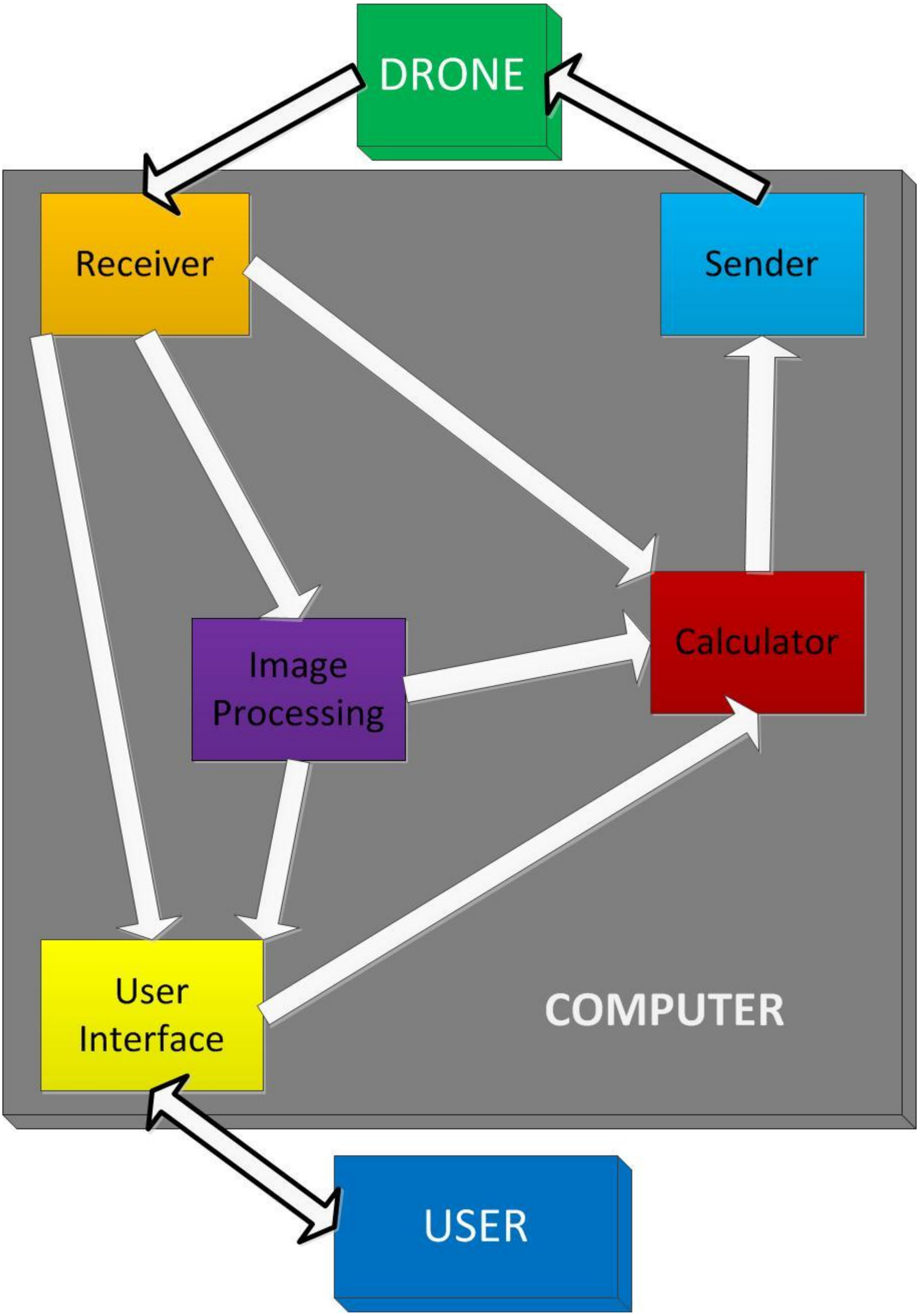
Legends 1:



Legends 2:



III.a. Global structure



Global Structure

Legends 1

Each module is composed of at least one thread, is in charge of a very specific task and has its own source files in directories with the same name.

The Receiver:

Module of communications.

Receives images captured by the drone and current balance of the drone through a WiFi connection.

Image Processing:

Module of image processing.

Extracts valuable information (post signals, obstacles, lines, etc…) from the images captured by the drone.

User Interface:

Module of communications and images displayer.

Displays images and interprets user command.

Calculator:

Module of decisions.

Finite state automaton that can compute an appropriated behaviour for the drone.

Sender:

Module of communication.

Transmit commands to the drones.

Global operation:

The Images are captured by the bottom camera of the **drone** and sent to the **computer**.

They are received and spread by the **Receiver** to the **User Interface** and the **Image Processing**.

The **Image Processing** processes the images, send the extracted information to the **Calculator and** the images processed to the **User Interface**.

The **User Interface** displays all the images and transfer the eventual user order to the Calculator.

The **Calculator** compute an appropriate behaviour for the drone that he sends to the **Sender**.

The **Sender** transmit the commands to the drone.

III.b. Inter-module Communication

The modules communicate through extern variables protected by mutex.

All variables are declared in 6 declaration files gathered in the directory “common”:

- “command.h”
- “images.h”
- “imageInterpretation.h”
- “record.h”
- “settings.h”
- “state.h”

Command.h:

Protected by the mutex: **commandUpdate_lock**

Declares type:

```
typedef struct commands{
            int typeControl;
           float roll;
           float pitch;
           float gaz;
           float yaw;
}CommandsList;


typedef enum{
           WITHOUT,
           AUTO,
           MANUAL
}TypeControls;
```

Variable	Type	Function
newCommand	bool	Is true if a new command computed is still not sent
Takeoff	bool	Signal to take off
Land	bool	Signal to land
isFlying	bool	Is true if the drone is flying
currentTypeControls	TypeControls	Type of control
comparaisonAngle	float	Maximum angle allowed
CommandsToSend	CommandsList	New Command to be sent
CommandsSent	CommandsList	Command sent

images.h:

Protected by the mutex: **imagesUpdate_lock**

Declares type:

```
typedef struct{
        IplImage *image;
        int number;
        char *path;
        float roll;
        float pitch;
} imageNumbered;
```

Variable	Type	Function
imageBottomCamera	imageNumbered	Image captured by the bottom camera
imageBottomRecieved	bool	Is true if a new bottom image has been received
imageBottomRecievedUndisplayed	bool	Is true if a new bottom image has been received and is still not displayed
imageFrontCamera	imageNumbered	Image captured by the front camera
imageFrontRecieved	bool	Is true if a new front image has been received
imageFrontRecievedUndisplayed	bool	Is true if a new front image has been received and is still not displayed
processedFrontCamera	imageNumbered	Image captured by the front camera once processed
imageFrontProcessedUndisplayed	bool	Is true if a new front image has been processed and is still not displayed
processedBottomCamera	imageNumbered	Image captured by the bottom camera once processed
imageBottomProcessedUndisplayed	bool	Is true if a new bottom image has been processed and is still not displayed

imageInterpretation.h:

Protected by the mutex: **processedInformationUpdate_lock**

Declares type:

```
typedef enum{
         NOSYMBOL,
         SYMBOLTURNRIGHT,
         SYMBOLTURNLEFT,
         SYMBOLRAISE,
         SYMBOLLAND,
         SYMBOLSPEED,
        SYMBOLALTITUDE
} symbolType;
```

Variable	Type	Function
imageBottomProcessed	bool	Is true if a new bottom image has been processed
imageFrontProcessed	bool	Is true if a new front image has been processed

bottomThetaLine	float	Angle of the probable main line (polar coordinates)
bottomThetaLine2	float	Angle of the probable second main line (polar coordinates)
bottomRhoLine	float	Rho of the probable main line (polar coordinates)
bottomRoll	float	Roll of the Drone when the image was captured
bottomPitch	float	Pitch of the Drone when the image was captured
bottomIntersection	CvPoint*	Coordinate of an eventual intersection between two lines
lostBottomFrames	int	Number of picture since the line has been lost
lostFrontFrames	int	Number of picture since the line has been lost
Symbol	symbolType	Symbol detected
symbolValue	int	Value of the symbol
theNumber	int	Reception rank of the image that we process

record.h:

Protected by the mutex: **recordUpdate_lock**

Declares type:

```
typedef struct elmt{
  char line[100];
  bool written;
} commentLine;

typedef struct el{
  commentLine lines[6];
} commentText;
```

Variable	Type	Function
record	bool	Is true if a report is asked
records	commentText*	Is true if a new image has been processed
number	int	Number of images received
directory	Char*	Path of the directory where the images and the report will be registered
startTime	clock_t	Date f the beginning of the program
fileReport	FILE*	File where the report will be written

setting.h:

Protected by the mutex: **settingsUpdate_lock**

Variable	Type	Function
colorLine	int	Color of the line
colorSignals	int	Color of the Signal
colorReperes	int	Color of the Benchmark
thresholdLowBottomCamera	int	Low threshold of the Hough transform for the bottom camera(when the drone is high)

thresholdHighBottomCamera	int	High threshold of the Hough transform for the bottom camera (when the drone is low)
thresholdFrontCamera	int	Threshold of the hough transform for the front camera
intensityLineForFrontCamera	float	Coefficient for color recognition
intensityLineForBottomCamera	float	Coefficient for color recognition
intensitySignals	float	Coefficient for color recognition
intensityReperes	float	Coefficient for color recognition
intensityHighLineForBottomCamera	float	Coefficient for color recognition
intensityHighSignals	float	Coefficient for color recognition
intensityHighReperes	float	Coefficient for color recognition
luminanceUpperBound	int	Above this limit, pixel are considered white
luminanceHighBound	int	Above this limit, pixel are considered lightful
luminanceLowerBound	int	Under this limit, pixel are considered black
toleranceAngleRectangle	float	Tolerance for angle measures
toleranceLengthRectangle	float	Tolerance for length measures
limitNumberPixelForSignal	int	Number of pixel required to detect a point
limitDifferenceAngleBetweenTwoLines	Float	Minimal angle between two lines
limitNbPixelsReperes	int	Number of pixel required to detect symbol

state.h:

Protected by the mutex: **stateUpdate_lock**

Variable	Type	Function
controlState	int	Control State
batteryLevel	int	Battery Level
balanceTheta	float	Pitch of the drone
balancePhi	Float	Roll of the drone
balancePsi	Float	Yaw of the drone
Altitude	Int	Altitude

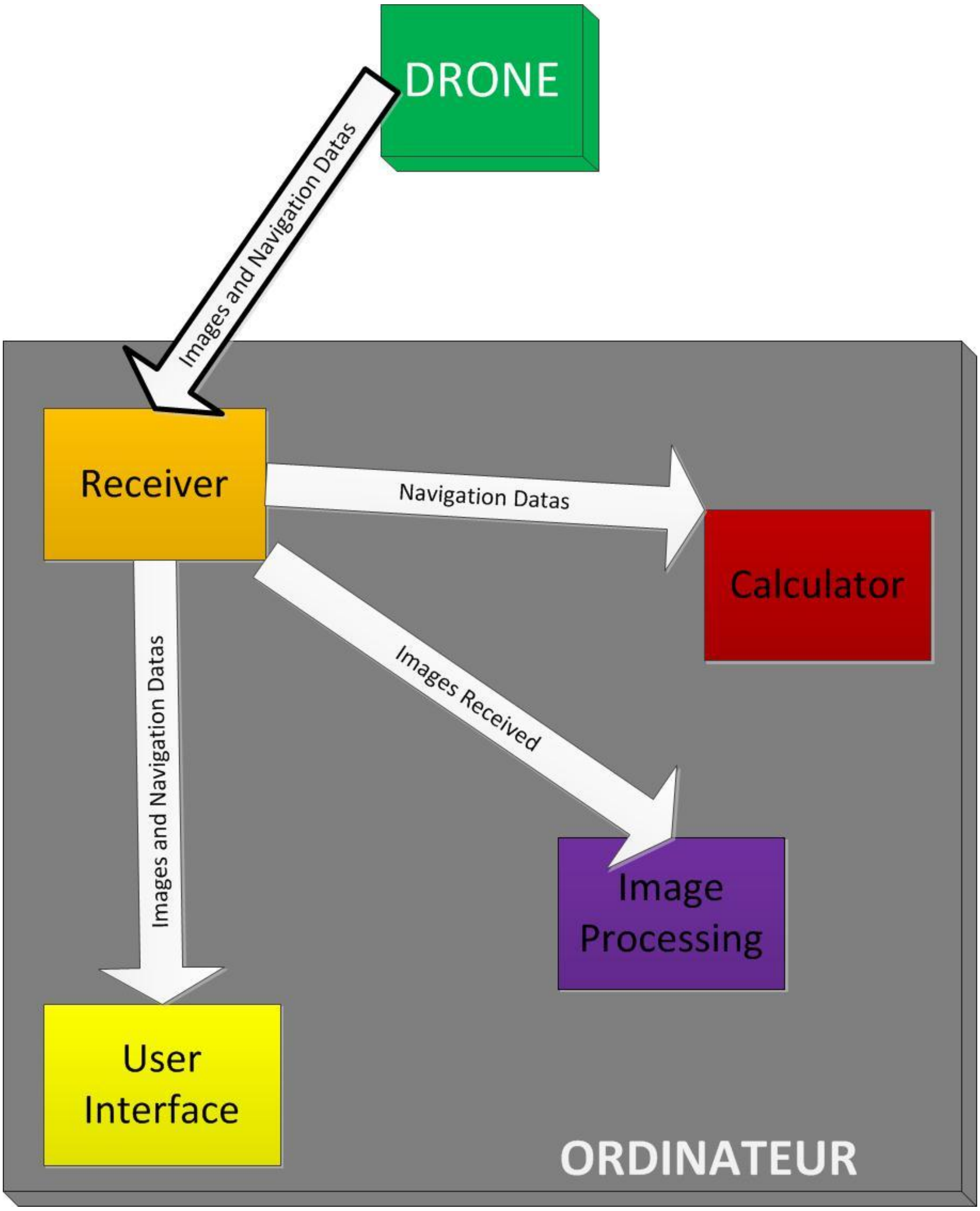
Each mutex is ordered such as a mutex A whose rank is inferior to another mutex B can't be locked if the muxtex B is already locked:

1. **imagesUpdate_lock**
2. **processedInformationUpdate_lock**
3. **commandUpdate_lock**
4. **stateUpdate_lock**
5. **recordUpdate_lock**
6. **settingsUpdate_lock** (not used actually)

III.c. Module by Module

The Receiver:

The



Receiver's communications

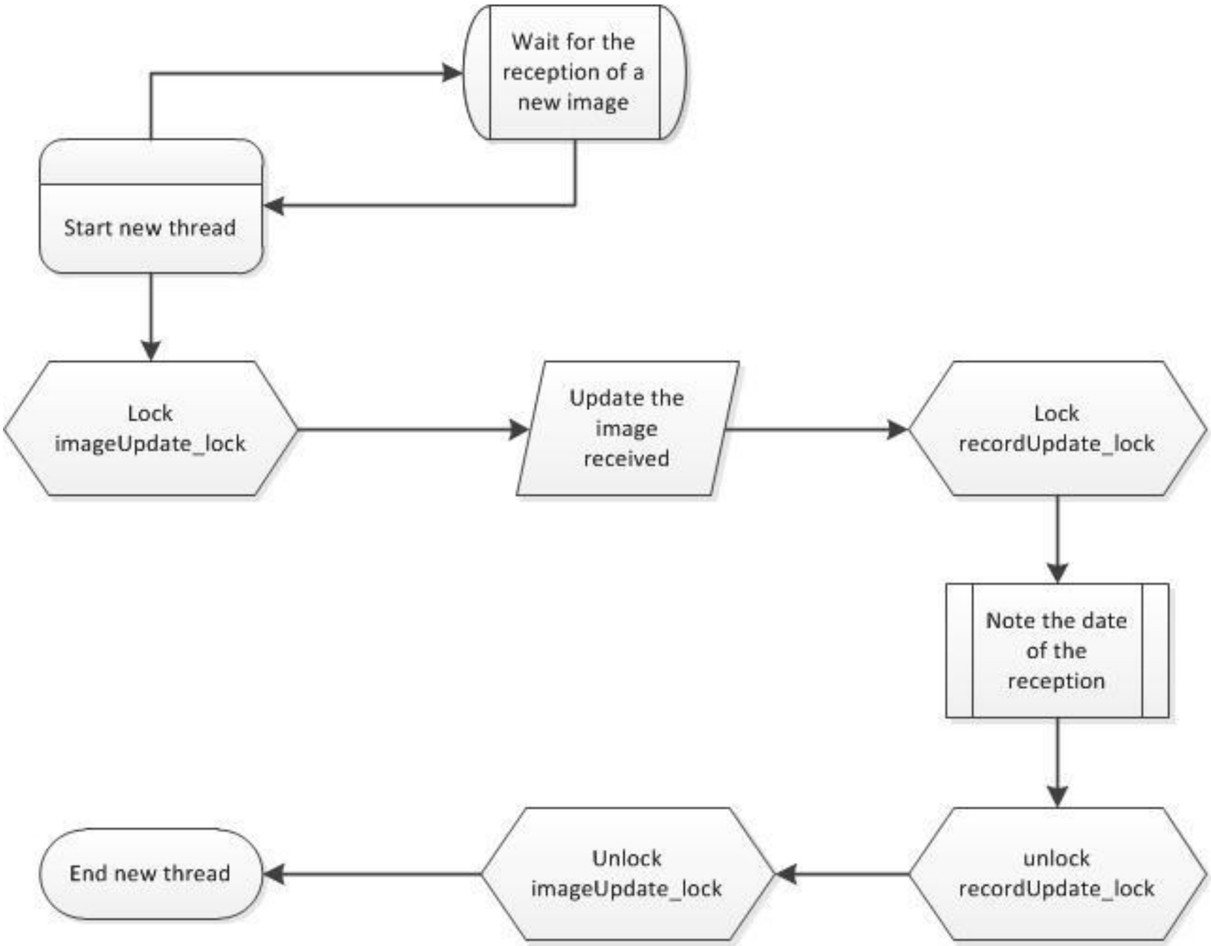
Legend 1

Actually, the receiver receives only the bottom cameras pictures. Indeed, we can switch the source of the picture captured (front or bottom camera) and check the camera in use.
However the Drone communication channel is designed such a way that images source is not consultable at the reception of an image and the drone bufferize the images before sending them (an intern buffer that cannot be accessed).
Also we can't receive images from both cameras without ending up confusing them.

The receiver is composed of two thread mainly written by Parrots' engineers:

- One to receive images (video)
- One to receive navigation data (navdata)

Both of them relies on a similar operation structure. Here is the structure of the video receiver:

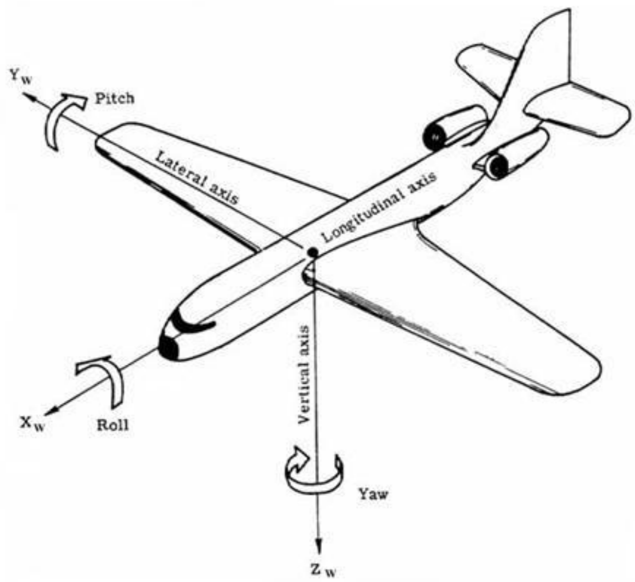


The Receiver's operation structure

Legends 1

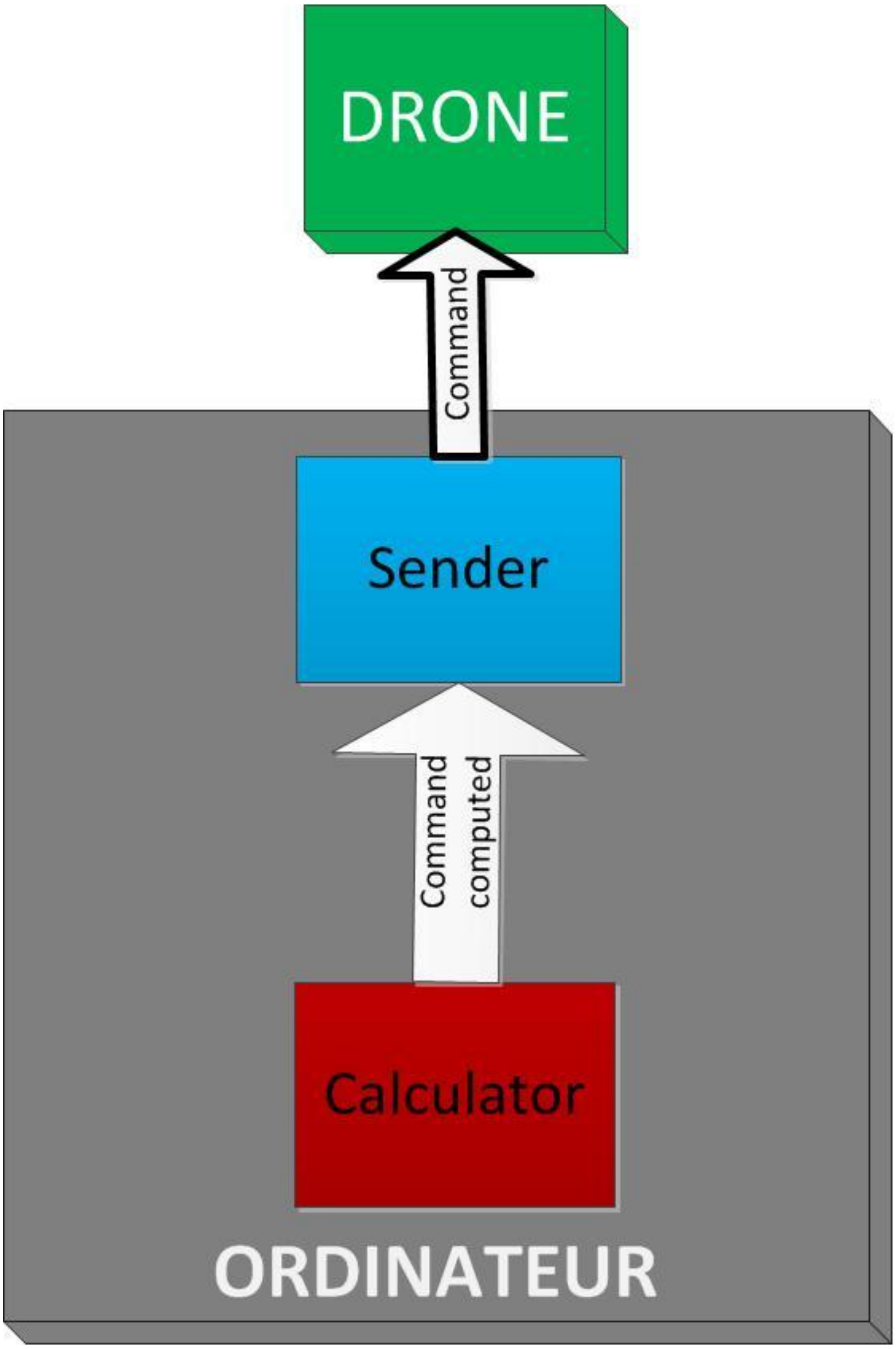
The navigation data are mainly composed of:

- The Altitude of the drone
- Its pitch
- Its roll
- Its yaw
- Its vertical speed
- Its battery level



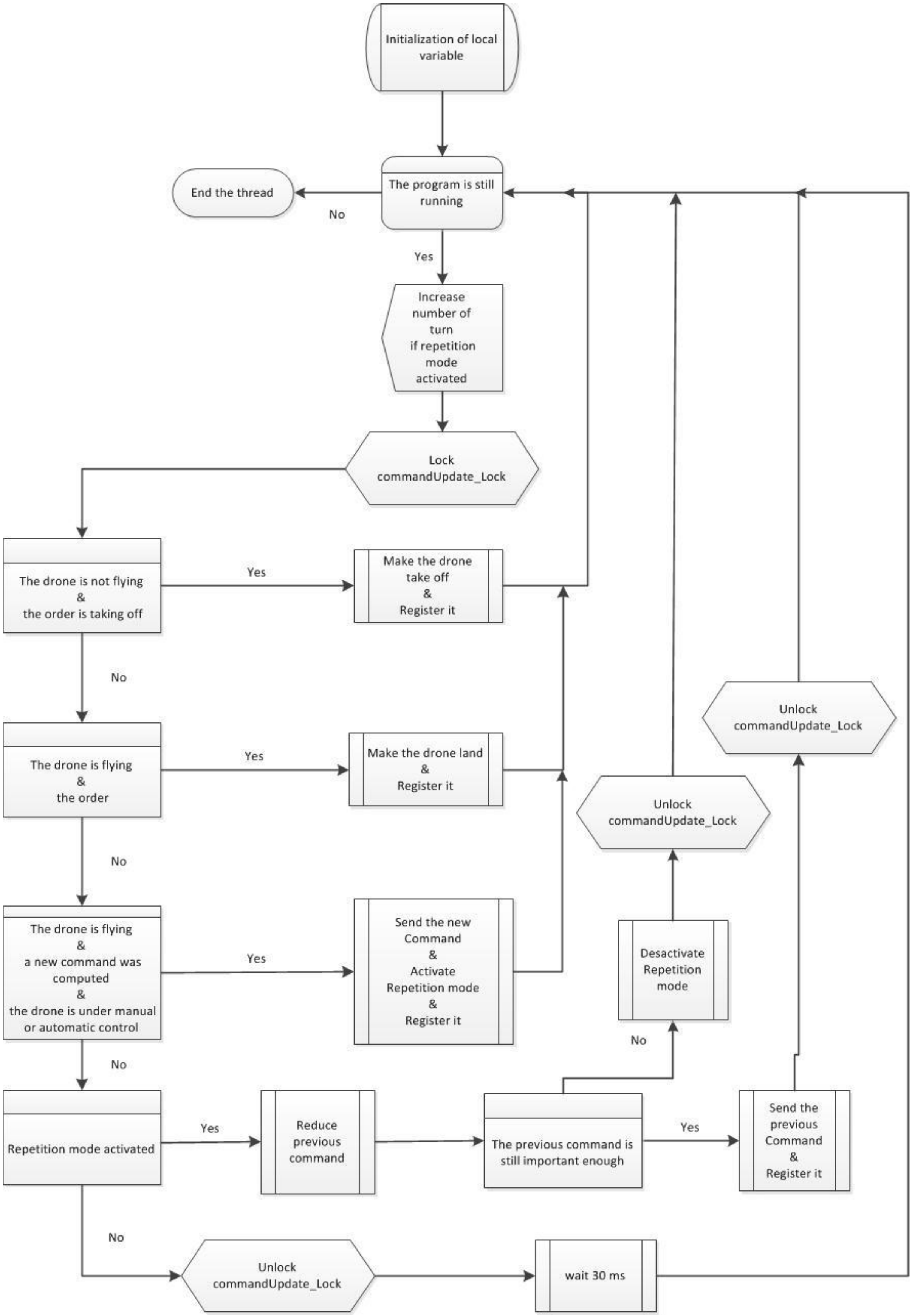
Balance of the Drone

The Sender:



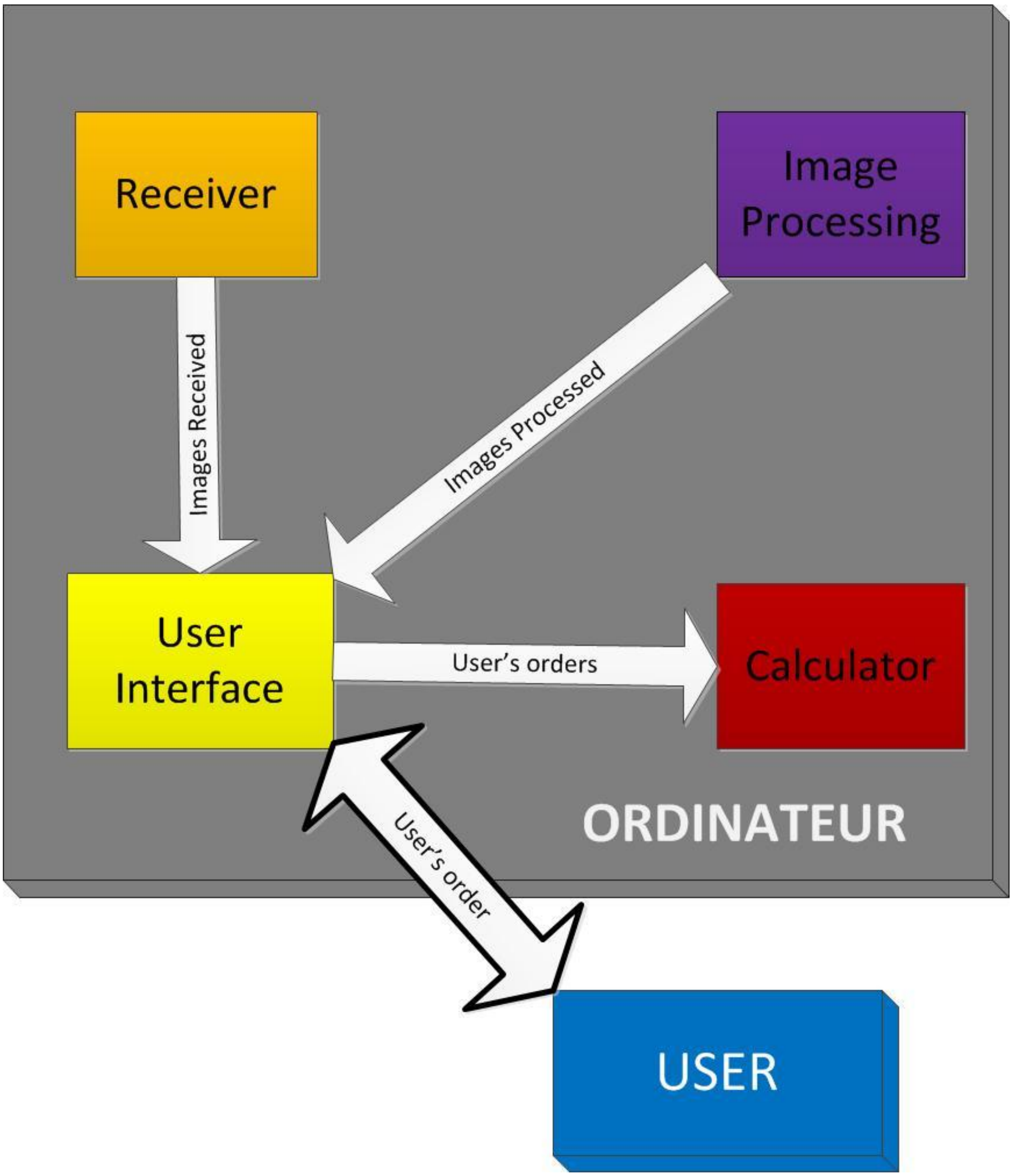
The Sender's communications
Legend 1

The sender should have been simple structure waiting for a new order and transmitting it but, due to the new SDK, the sender has been the trickiest part. Indeed, it has to find out a fragile balance between flooding the drone by orders and repeating not sufficiently or not fast enough the command to make the drone move:



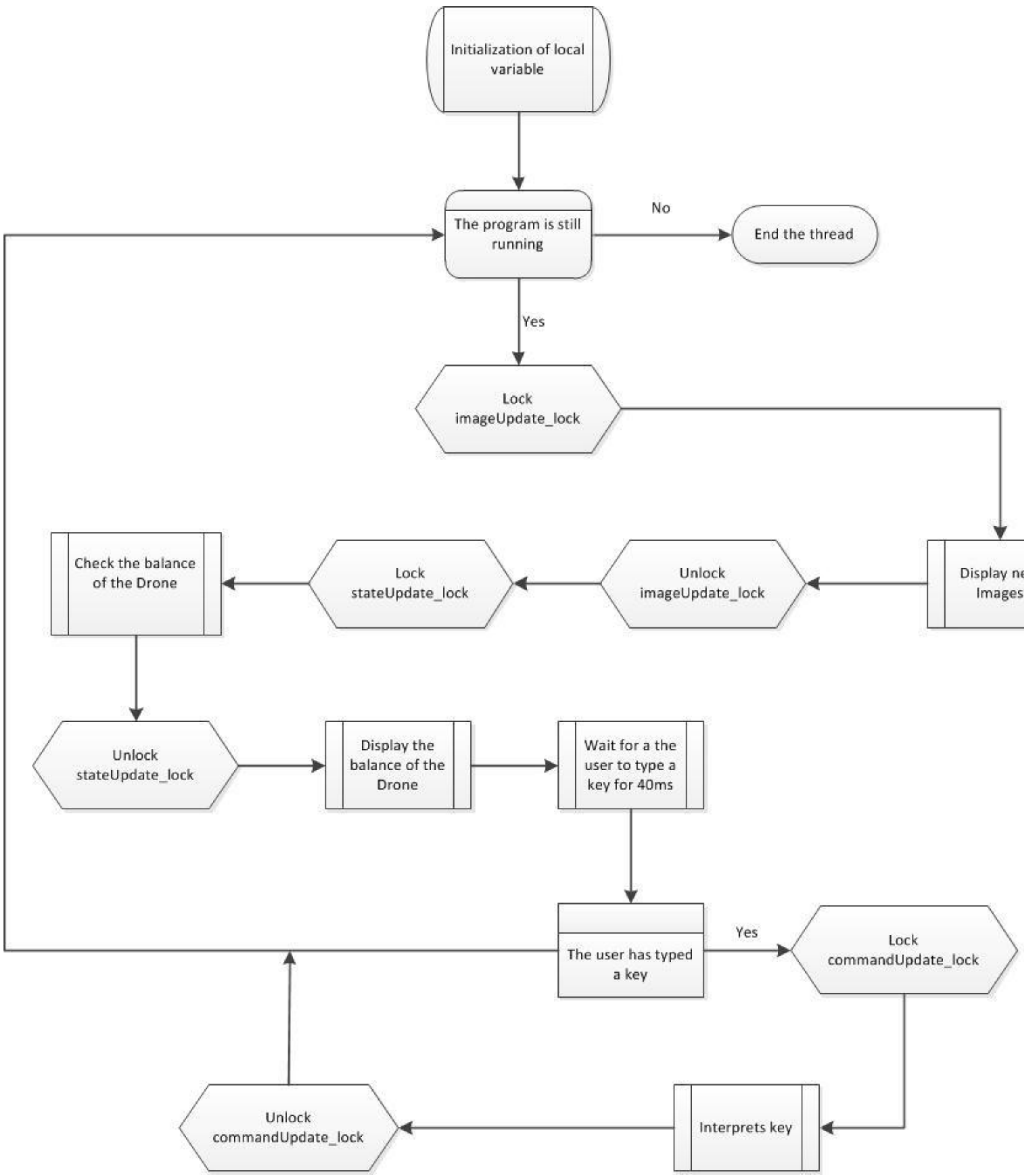
The sender's operation

The User Interface:



The user Interface's communication
Legend 1

The user interface is a simple loop:

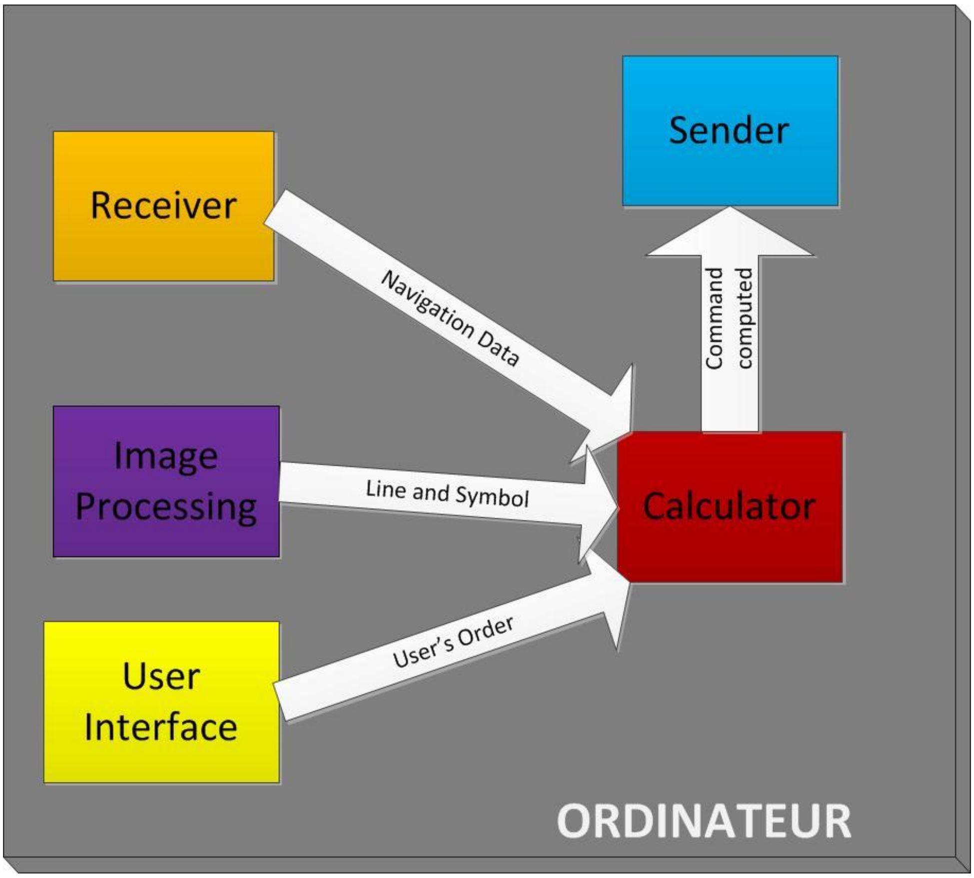


User interface operation

The key that can be used are:

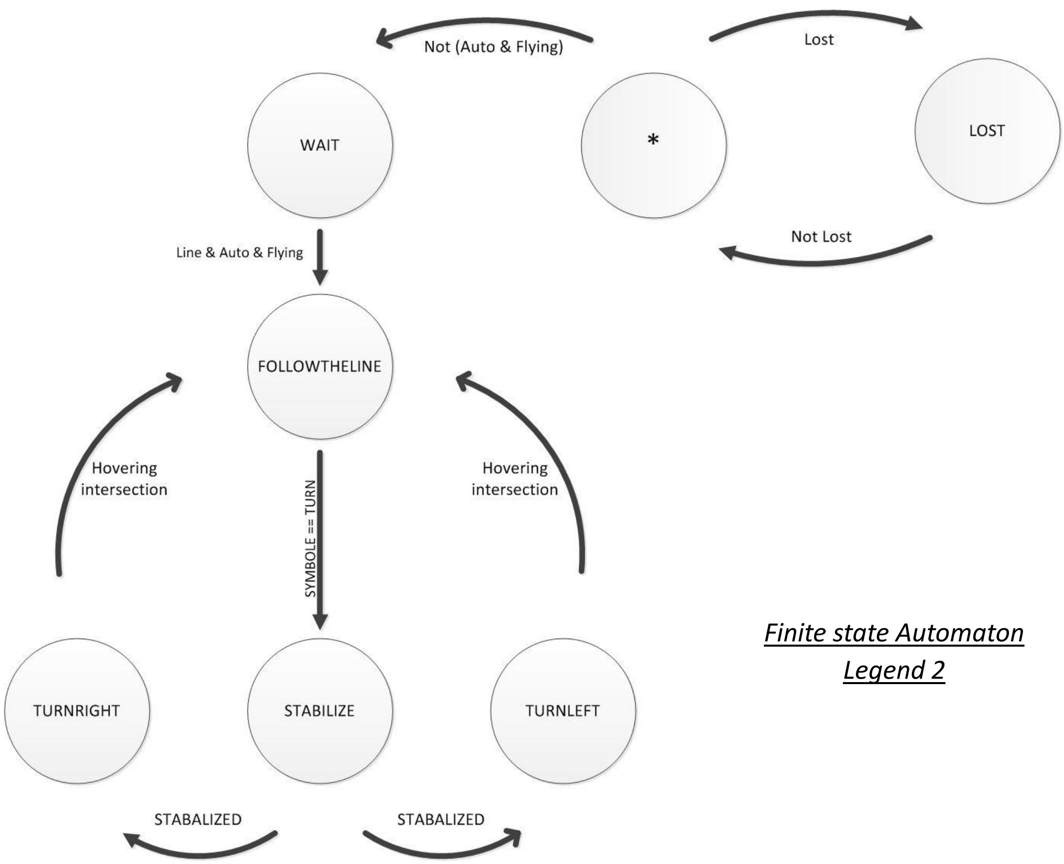
Key	Action
!	Forces the euler_angle_max to 0.52
t	Makes the drone take off
l	Makes the drone land
a	Puts the drone under an automatic control
m	Puts the drone under an manual control
w	Frees the drone from any control
c	Ends the program
⇧	In manual control, increases altitude
⇩	In manual control, decreases altitude
⇨	In manual control, increases yaw
⇦	In manual control, decreases yaw
z	In manual control, decreases pitch
s	In manual control, increases pitch
q	In manual control, decreases roll
d	In manual control, increases roll

The Calculator:



Calculator's communication
Legend 1

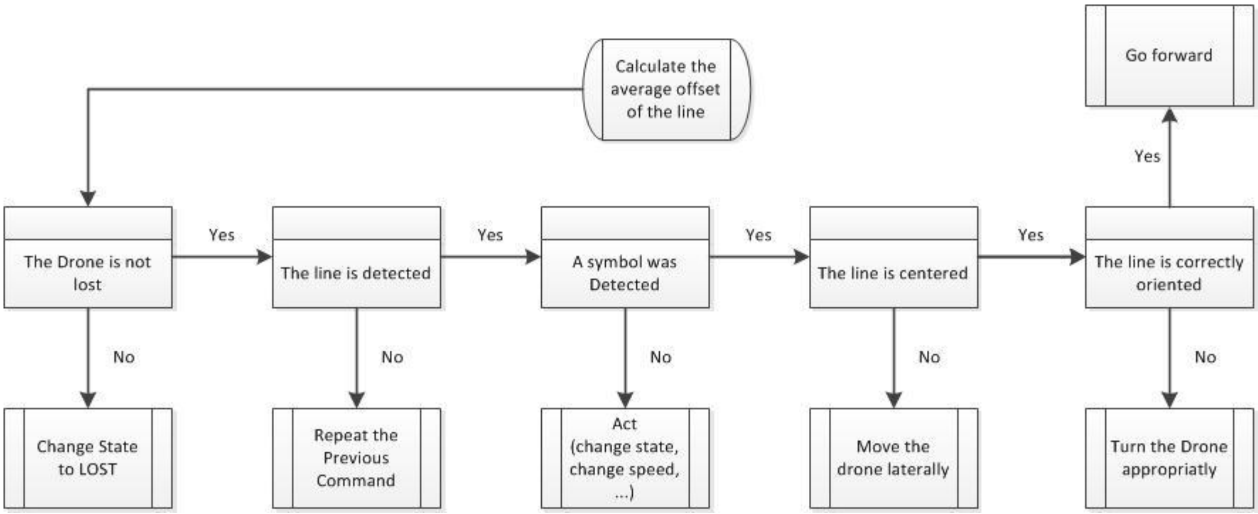
The calculator is a finite state automaton sparked by the reception of new valuable information which can be described by the following schema:



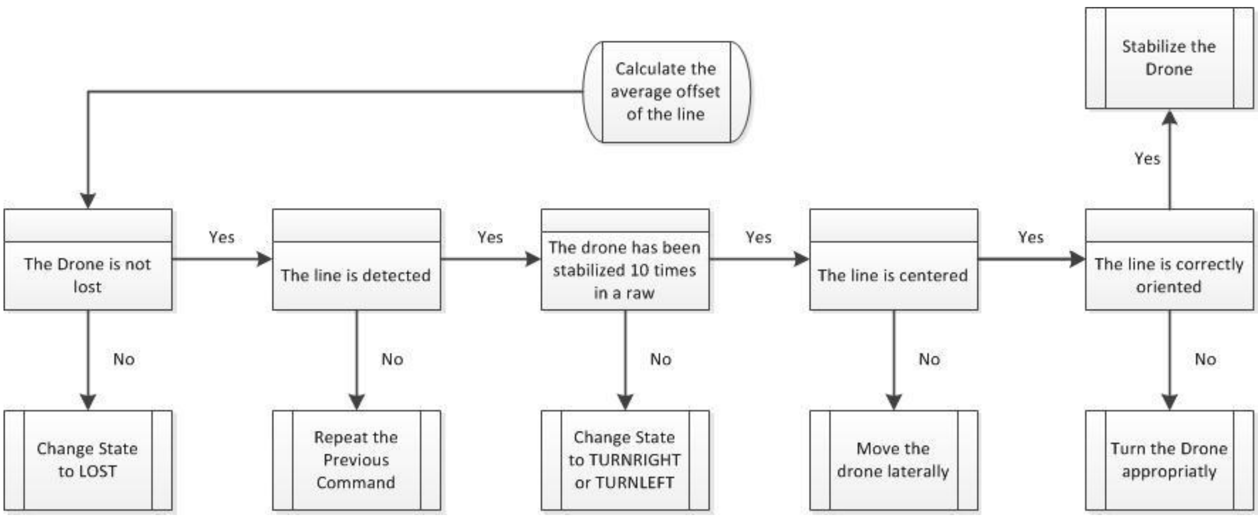
Finite state Automaton
Legend 2

Each state has its own operation. But you could notice some similitude:

FOLLOWTHELINE operation:



STABILIZE operation:

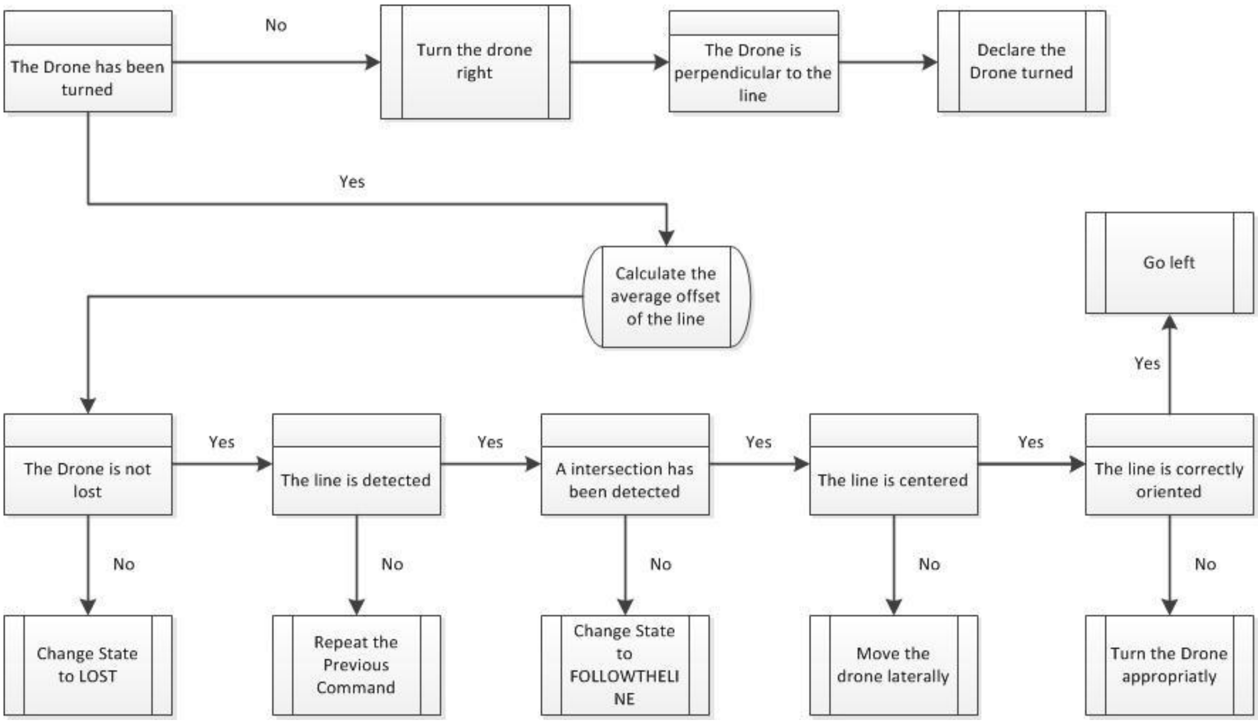


LOST operation:

When the Drone is lost, as long as the line is not detected the drone is going to look for the line:

- Getting higher
- Going backward
- Turning right and left

TURNRIGHT operation:



TURNLEFT operation:

Work the same way as turn right.

All the commands emitted by the calculator are defined according to the navigation data of the drone and a set of variables redefined for each state. Those variable are declared in a header file in the directory "calculator".

Variable	Type	Function
toleranceAngleDifferenceCommand	float	Tolerance for the angle between the line and the drone
pitchBoost	float	Reactivity for the changing of pitch
rollBoost	float	Reactivity for the changing of roll
desiredAltitude	int	Altitude to maintain
toleranceAltitude	int	Tolerance for the difference between the actual altitude and the desired one
coefficientAltitude	float	Reactivity for the changing of altitude
numberOfFrameBeforeLost	int	Number of frame without detecting the line before being lost

numberOfFrameBeforeTurningRight	int	Number of frame without detecting the line before turning right
numberOfFrameBeforeTurningLeft	int	Number of frame without detecting the line before turning left
numberOfFrameBeforeLanding	int	Number of frame without detecting the line before landing
offsetRight	int	Offset of the drone compared to the line
squareIntersection	int	Precision in the nearing of an intersection
limitOffset	float	Tolerance for the average offset of the line
precisionAngleToFollow	float	Precision for the angle between the line and the drone
rollCoefficient	float	Lateral speed
currentPitch	float	Maximal speed
minorPitch	float	Average Speed
yawCoefficient	float	Rotation speed
stableLimit	float	Stabilization condition
lastIntersection	CvPoint	Last intersection detected

The Image Processing:

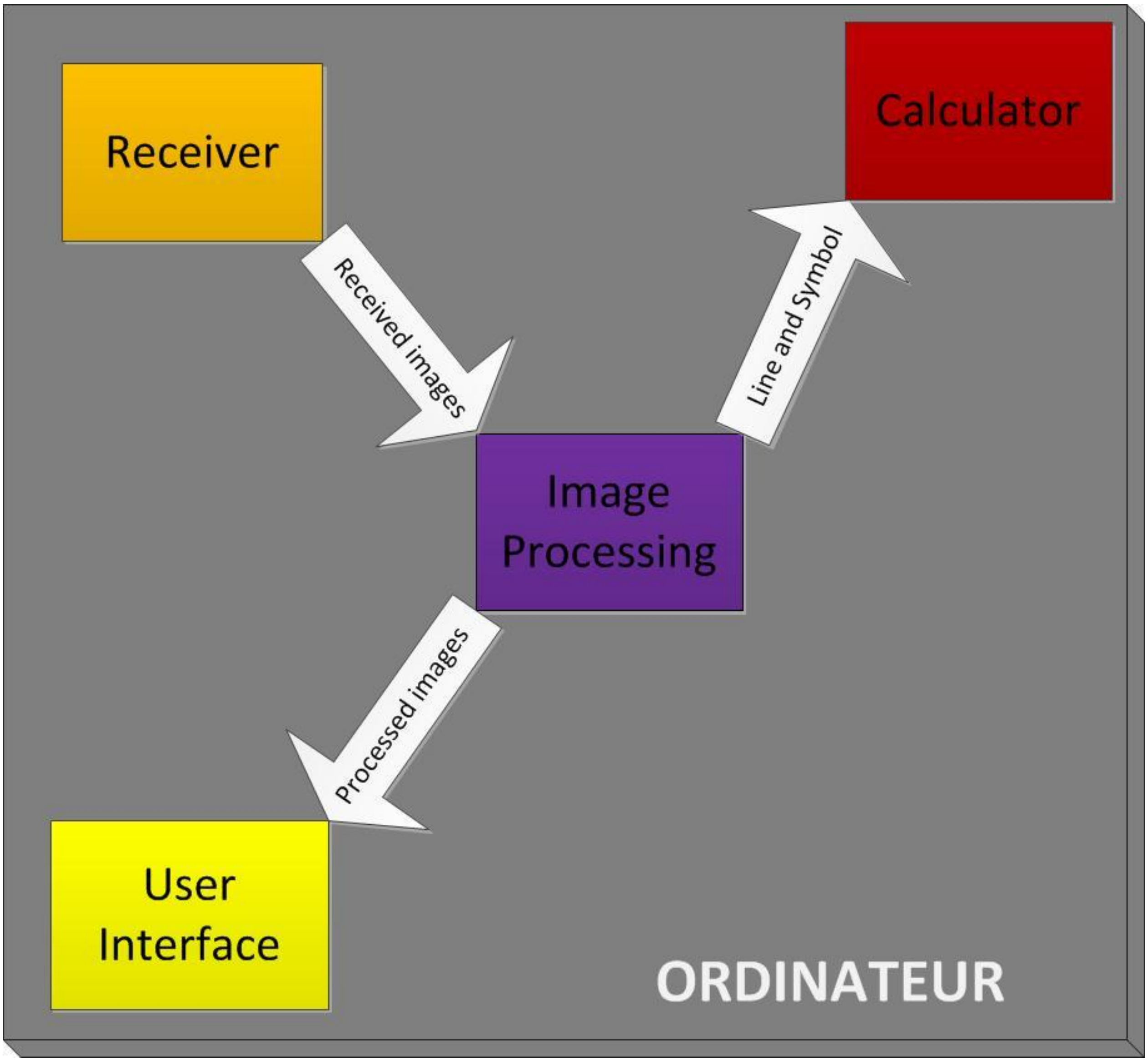


Image Processing's communication
Legend 1

To sum up, this module:

- Wait for a new image to process
- Look for the line
- Look for a symbol
- Transmit the valuable information

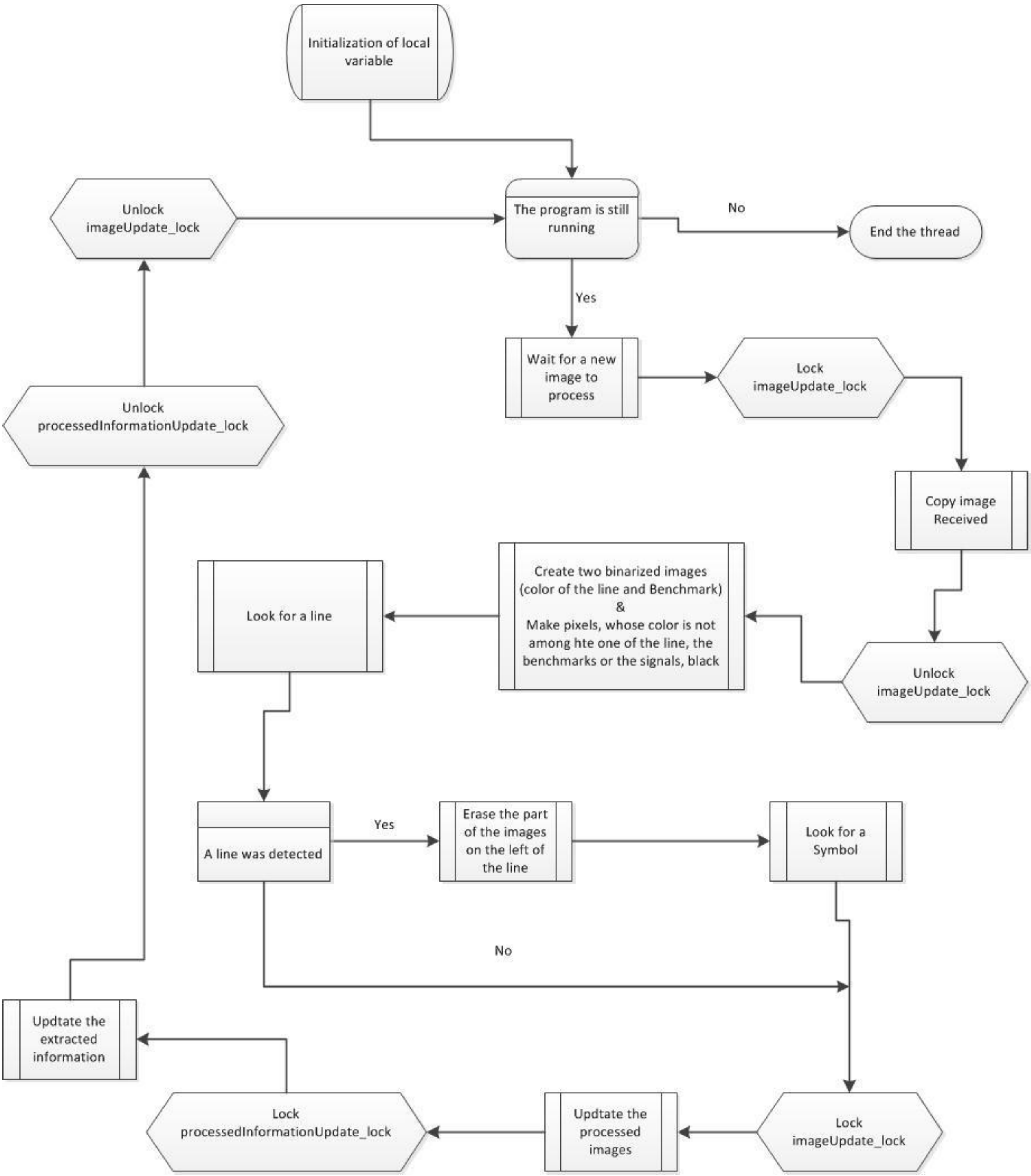


Image Processing's operation

III.d. Activities report

An option allows the program to register every images received and processed in a chosen directory.

Moreover, in this case, would be created a report in the very same report giving the date of reception and processing for each image and the order computed according to this image and the state of the Drone.

The report could put in light some latency issues. Indeed, in average, an command is computed with 150ms of delay.

Eventually, a testing program let the developer use a directory full of images to emulate the drone sending images and process the image and deduce an appropriate behavior.

III.e. How to use the program

The file source is composed of a:

- A Makefile
- A directory named **"myDemo"**

Download the latest version of the ARDrone SDK.

Extract him and go to the directory:

ARDrone_SDK_2_x/Examples/Linux

Extract inside the source files (note that you must replace the current make file)

In the directory **"myDemo"** you could find:

- A directory "Build" with the Makefile to build the project
- A directory "BuildTest" with the Makefile to build the image processing test
- A directory "images" destined to receive the report and the images recorded
- A directory "Release" with the applications once build
- A directory "Souces" with the source file

IV. The image processing functions

IV.a. Color detection

During this chapter, we will note luminance $l \epsilon [0,1]$ and a pixel in RGB-space $P = \begin{pmatrix} r \\ g \\ b \end{pmatrix}$.

The primary colors should have been obviously the easiest one to detect because of the RGB-pattern of the pixels.

We have measured the efficiency of the detection for six colors:

- Blue
- Red
- Green
- Cyan
- Yellow
- Magenta

The experience demonstrate that Red, Green and Cyan, probably due to their high luminance, are the easiest one to detect. Blue and yellow the most difficult ones.

Also a pixel P can be said red if:

- r > $\mu_{\sigma,\delta}$ (l)*g
- r > $\mu_{\sigma,\delta}$ (l)*b

where $\mu_{\sigma,\delta}$ is a function of the luminance that experimentally fit the best:

$$\mu_{\sigma,\delta}(x) = \begin{cases} +\infty, & l < 0.1 \\ \sigma - \sqrt[4]{\dfrac{l - 0.1}{0.5}}(\sigma - \delta), & l < 0,6 \\ \delta, & l < 0.85 \\ +\infty, & l \geq 0.85 \end{cases}$$

Where σ and δ are two constant usually between 1 and 2.

The same way we can detect Green pixel. For cyan pixel we first transform $P = \begin{pmatrix} r \\ g \\ b \end{pmatrix}$ in RGB-space to $P = \begin{pmatrix} g + b \\ r + b \\ r + g \end{pmatrix}$ in CyanMangentaYellow-space then apply the same comparisons.

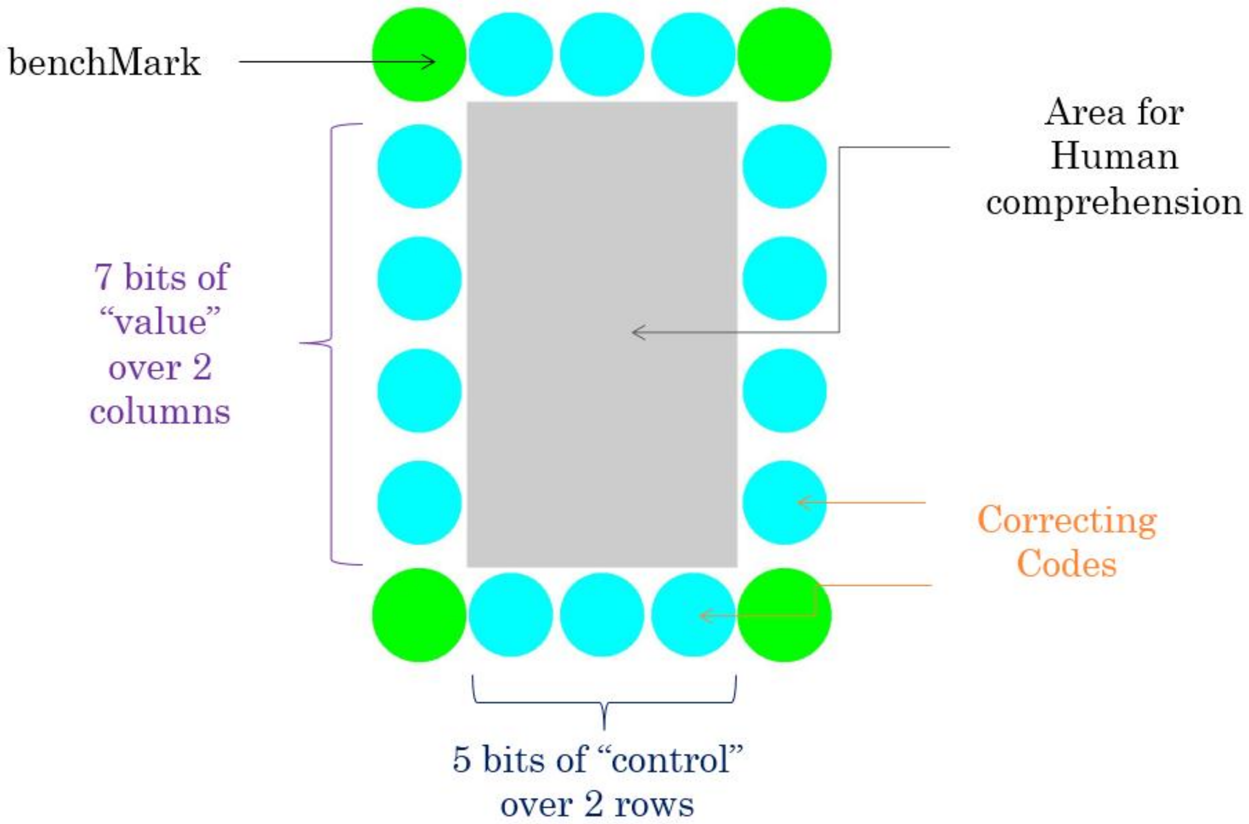
The efficacy of such a method allowed us to detect the line thanks to a Hough transform applied to an image binarized according whether a pixel is Red or whether it is not.

IV.b. Symbol detection

In order to define a set of symbol for the drone, we had to pay attention to the following aspects:

- Easily detectable by the Drone
- Understandable by Human
- New (cannot be confounded with any other signal posts already existing)

It came out that the easiest way to design the symbol was to design a two part symbol, one for the drone, one for the Human. We eventual fix our choice on the following one:



Symbol Common pattern

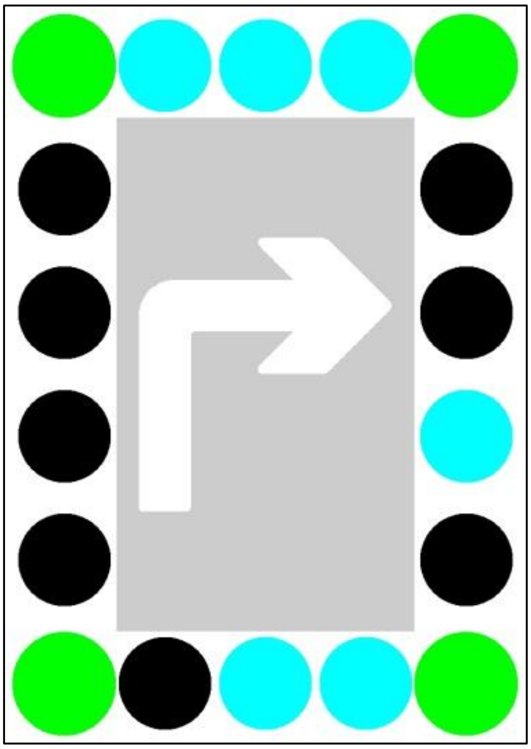
A simple connnexe component analysis, on a binarized picture according whether the pixel is green or not, allows us to determine the eventual benchmark as the four main connexe components.

The eventual benchmark must be the top of a rectangle. This verification allows us avoid most of the errors.

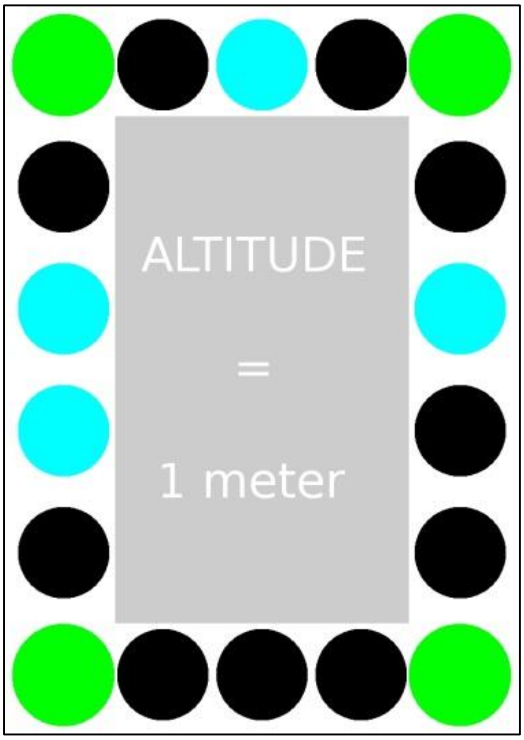
Eventually, since the positions of the benchmarks are known, we just need the existence of cyan pixel at the position of the correcting codes and of the bit of control and value to detect them.

The correcting codes guarantee us a reliable detection. The correcting codes guarantee the imparity of the number of cyan point on the rows and on the columns.

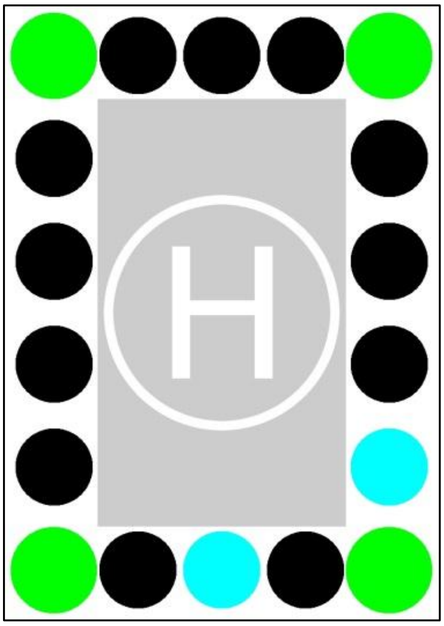
Example of symbols:



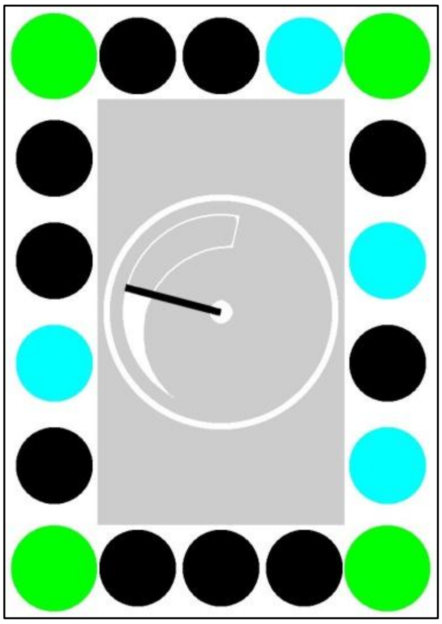
Turn right:
control=29; value=1



Turn left:
control = 8; value=50



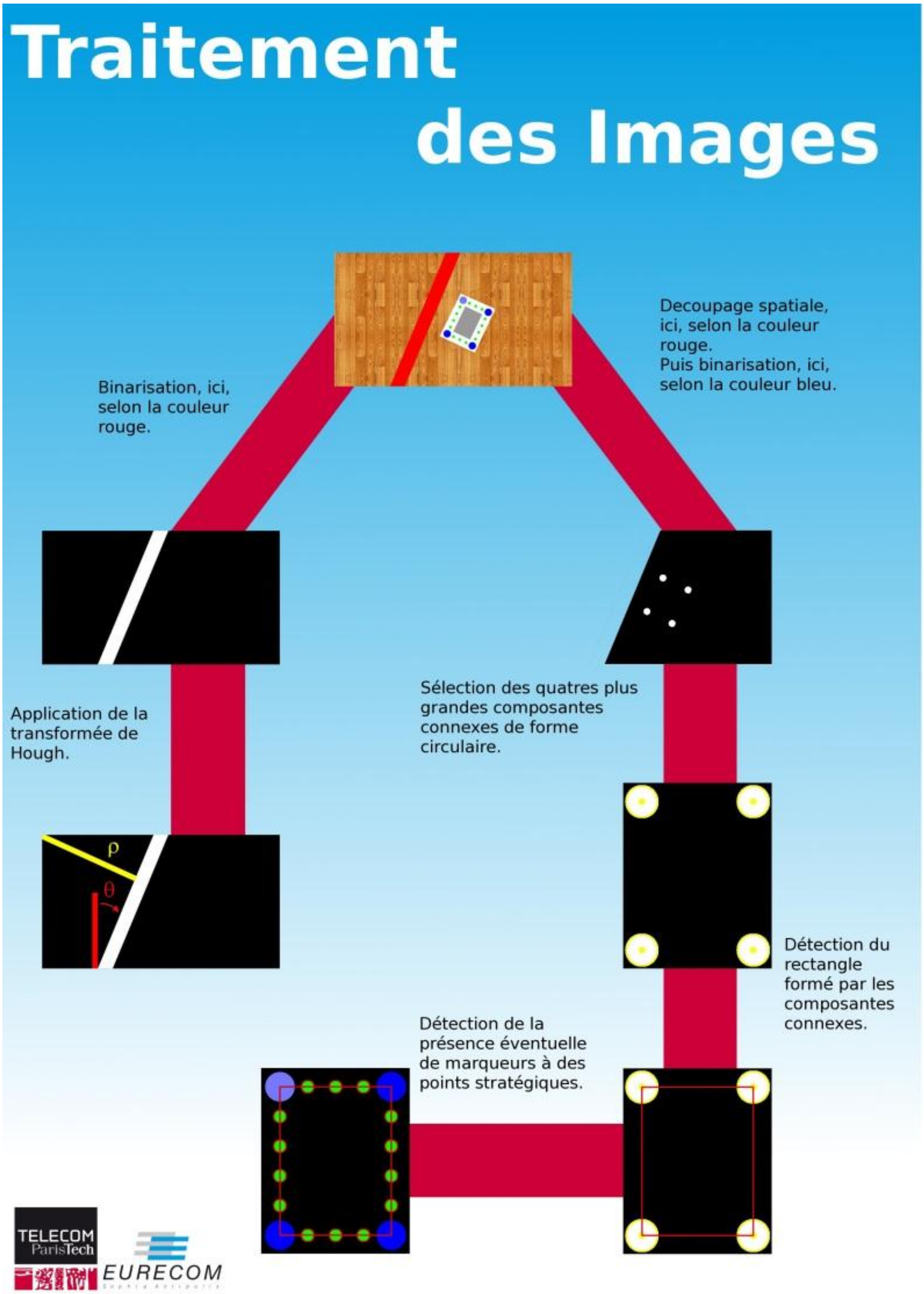
Land:
control=1; value=0



average speed:
control = 4; value=18

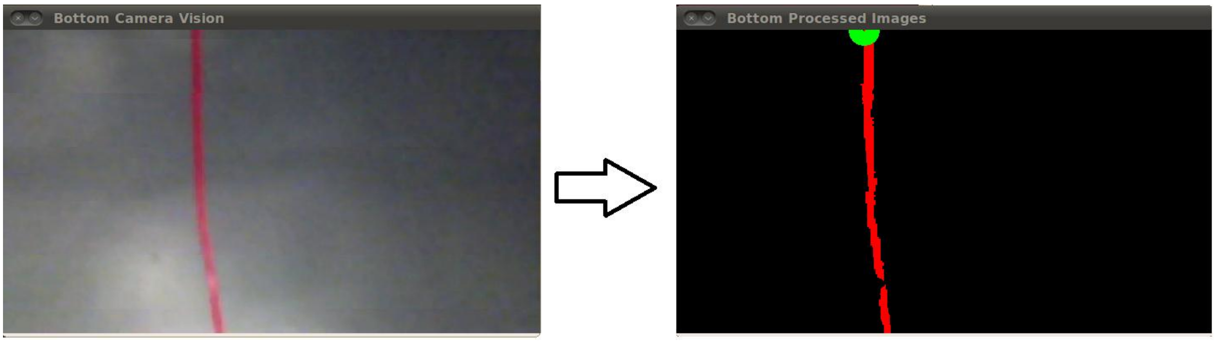
IV.c. To put it in a Nutshell

During demonstration we had to find a way to explain simply the image processing functions. Also we have created this king of flyer:



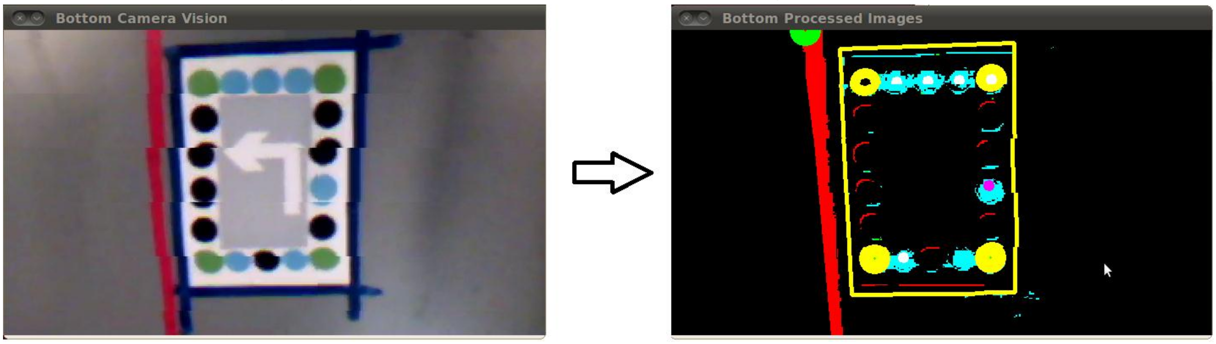
V. The outcomes

- The auto-pilot can make the Drone follow curved lines



- The auto-pilot can detect up to 4096 different symbols on flight

- The auto-pilot can make the drone turn 90° angles, change its speed, its altitude or even land consequently to the detection of the associated symbol



- The drone can be manualy piloted

- The video stream can be registered

- A report of the activities can be emitted and consulted

- The image processing can be tested independently of the use of a drone

Conclusion

The objectives of the mission were mainly reached.

The main difficulties remain the control of the intern environment of the drone.

The next step to ameliorate the project could be:

- Prefect the unitary tests
- Reduce the latency issues underlined by the report
- Get a better control through the sender
- Modify deeply the library to add a flag to the images pointing out the camera sources