



INSA TOULOUSE

PROJECT RELEASE REPORT

---

## Autonomous Drone Guide

---

*Engineers:*

Mathieu OTHACEHE  
Florian VOLKMANN  
Duc Ngoc HUYNH  
Fadi HALLOUMI

*Supervisors:*

Mr.Pierre-Emmanuel HLADIK  
Ms.Dinnah MCCARTHY

*Mentor :*

Mr.Didier LE BOTLAN

# Autonomous target-guided navigation in low obstacle enviroment

Halloumi Fadi

`halloumi@etud.insa-toulouse.fr`

Othacehe Mathieu

`othacehe@etud.insa-toulouse.fr`

Volkman Florian

`volkmann@etud.insa-toulouse.fr`

Huynh Duc Ngoc

`huynh@etud.insa-toulouse.fr`

January 11, 2013

### **Abstract**

Being fifth year students at INSA Toulouse, we saw that many visitors had difficulties to find the Electrical and Computer Engineering Department (DGEI) and they were late for the meeting. At the end of our study in Critical and Embedded System, we have an opportunity to work with a drone so we decided to make our project more useful. Our Pink team is working on developing a UAV that can go to DGEI wherever it was by searching a signal power of an Xbee emitter (or a GPS).

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>General project presentation</b>	<b>3</b>
2.1	Goal of our project . . . . .	3
2.2	Documentation & deliverables . . . . .	3
<b>3</b>	<b>Hardware considerations</b>	<b>4</b>
3.1	Parrot 2.0 . . . . .	4
3.1.1	Ressources . . . . .	4
3.1.2	Specification . . . . .	4
3.1.3	Hardware . . . . .	4
3.1.4	Software . . . . .	5
3.1.5	Development . . . . .	6
3.2	Arduino . . . . .	6
3.3	Android . . . . .	6
3.4	Ultrasonic Sensor . . . . .	6
3.5	Bluetooth . . . . .	6
<b>4</b>	<b>Software producted</b>	<b>7</b>
4.1	Repository architecture . . . . .	7
4.2	The auto thread . . . . .	10
4.2.1	Movement API . . . . .	10
4.2.2	Send an order . . . . .	10
4.2.3	Using of this API . . . . .	11
4.2.4	Storing navdatas . . . . .	12
4.3	The avoidance thread . . . . .	12
4.4	The receive_gps & gps_target threads . . . . .	15
4.5	The target thread . . . . .	16
4.5.1	Turn angle algorithm . . . . .	16
4.5.2	Convergence algorithm 1 . . . . .	17
4.5.3	Convergence algorithm 2 . . . . .	17
<b>5</b>	<b>Final results</b>	<b>19</b>
<b>6</b>	<b>Conclusion</b>	<b>20</b>

# 1 Introduction

Introduction

## 2 General project presentation

### 2.1 Goal of our project

Our project, proposed by our tutor D. Le Botlan is titled “Adptive Flight”. It is composed of two main parts.

The first one is about target detection and autonomous flight.  
The other one is obstacle avoidance.

Actually, our UAV must be able to detect a target, flight to this point and above all, avoid the human obstacles.

### 2.2 Documentation & deliverables

In the beginning of the project we have created a website on our project. It can be accessed at :

<https://sites.google.com/site/projetsecinsa/projets-2012-2013/projet-de-la-mission-pink>

We have also written technical documentation : PMP, SRS and STD. They can be downloaded in the “Sources” section of our website.

Regarding software production, we have used the versionning manager git. Our code is hosted at this url :

<https://github.com/insadrone/InsaQuadDrone>

Finally, various documents, unsorted, like mid-term presentations, hardware benchmarks can be seen in the included folder named “google\_drive\_dump”

## 3 Hardware considerations

### 3.1 Parrot 2.0

The Parrot AR.Drone is a radio controlled flying quadrotor helicopter built by the French company Parrot. The drone is designed to be controlled by any electronic device having wi-fi connection and sufficient ressource to run a control software. Only android, iOS and windows have officially distributed control applications. In other hand, only Windows and Linux OS are supported as a development platform. The choice of the department for this drone as a base for the project is due to the harmless character of this quadrotor. Two versions of the Parrot AR.Drone exists. For our project only the new version(Ar.Drone 2.0) is supported.

#### 3.1.1 Ressources

Many ressources could be found on this UAV and this what make it one of the best choices:

- <http://www.parrot.com/fr>
- <http://ardrone.parrot.com/parrot-ar-drone/usa/>
- <http://projects.ardrone.org/>
- <http://devzone.parrot.com/>

#### 3.1.2 Specification

Here a quick overview of the general specification of the drone:

- Autonomy : Approximately 12 minutes, recharging time of 1h30.
- Maximum range : 50m average, 100m in a wide-open space with few Wi-Fi waves.
- Maximum altitude : 6m is the stability limitation, 50m the wi-fi limitation (which could be hacked with wi-fi booster to go up to 75m).
- Maximum additional supported weight : 80g is the limit of stability, 100g is the limit of motors propulsion.
- Maximum speed : 18 km/h.
- Maximum supported wind speed : 2km/h.

#### 3.1.3 Hardware

The hardware reference is as follow:

- \* AR.Drone 1.0 : carte Mykonos, processeur ARM926EJ-S rev 5 (v5l) Wi-Fi: AR6000 Memory: 128MB RAM
- \* AR.Drone 2.0 : Mykonos2 card, Processor OMAP 3640 1GHz 32 bit ARM Cortex A8 with a video DSP 800MHz TMS320DMC64x

### 3.1.4 Software

The board has an embedded Linux with these reference : //

- \* Linux myhost 2.6.27.47-parrot-01227-g93dde09 #1 preempt Fri Jul 2 15:23:06 CEST 2010 armv5tej  
GNU/Linux
- \* Linux 2.6.32 kernel: Linux uclibc 2.6.32.9-g0d605ac #1 preempt Fri Apr 6 12:01:59 CEST 2012 armv7l  
GNU/Linux

This embedded linux contains these basic packages :

- BusyBox
- Mtdutils
- zlib
- ethtool
- procpops
- udev
- dsp bridge
- lcms dsp codec
- wireless tools
- exif
- iptables
- usbmodeswitch
- lsusb
- alsa lib
- barry
- busydroid
- webkit

We added multiple modules in order to be able to communicate with usb port:

- cdc-acm
- usbserial
- ftdi\_sio

you need to cross-compile these modules you can use the following step :

1. Download an ARM Cross-compiler you can find one on the website of Mentor Graphics (successor of Code sourcery) <http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/arm-gnu-linux>
2. Install the cross-compiler :

```
# : chmod +x arm-2012.03-57-arm-none-linux-gnueabi.bin
# : ./arm-2012.03-57-arm-none-linux-gnueabi.bin
```
3. Compile the module :

```
# : /opt/CodeSourcery/Sourcery_CodeBench_Lite_for_ARM_GNU_Linux/bin/arm-
none-linux-gnueabi-gcc -march=armv7-a toto.c -o toto.elf
```



### 3.1.5 Development

## 3.2 Arduino

We have used the arduino board uno, without any shield or add ons.

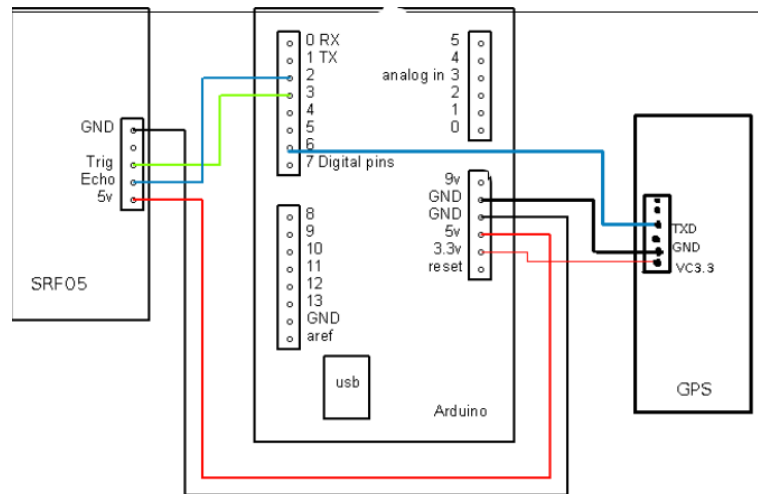


Figure 3.1: Arduino schematics

## 3.3 Android

## 3.4 Ultrasonic Sensor

## 3.5 Bluetooth

This part explains how to connect and read data from an android smartphone that sends information via bluetooth. ShareGPS, an application running on the phone is sending its GPS coordinate to the computer. The aim of the document is to give the process to get this coordinate.

First step: with the smartphone

- Turn on the bluetooth and the GPS
- Launch ShareGPS and share coordinate via bluetooth
- Make the bluetooth visible by other devices

Second step: with the computer on Linux

- Turn on the bluetooth and open a terminal (you may need to be root) Scan devices with: "hcitool scan" -> Write down the MAC address of the corresponding device
- Find the good channel which gives GPS coordinate with: "sdptool records MAC\_ADDR" -> This command displays all channels and their functions -> Look for the one called "ShareGPS" and write down the corresponding channel

- Create the connection with: “rfcomm bind X MAC\_ADDR CH” -> X: positive integer corresponding to the rfcomm you want to bind -> MAC\_ADDR: MAC address of the smartphone -> CH: channel of the ShareGPS application

You can now check that /dev/rfcommX exists. The last step is to read data. To kill it use the command “rfcomm release rfcommX” or “rfcomm release all”.

Third step: read data

- Using PuTTY: connect via serial to /dev/rfcommX with a 9600 baudrate
- You can also use a self-made program

Source: [http://www.thinkwiki.org/wiki/How\\_to\\_setup\\_Bluetooth](http://www.thinkwiki.org/wiki/How_to_setup_Bluetooth)

## 4 Software produced

### 4.1 Repository architecture

The general architecture of our code repository is explained in this tree :

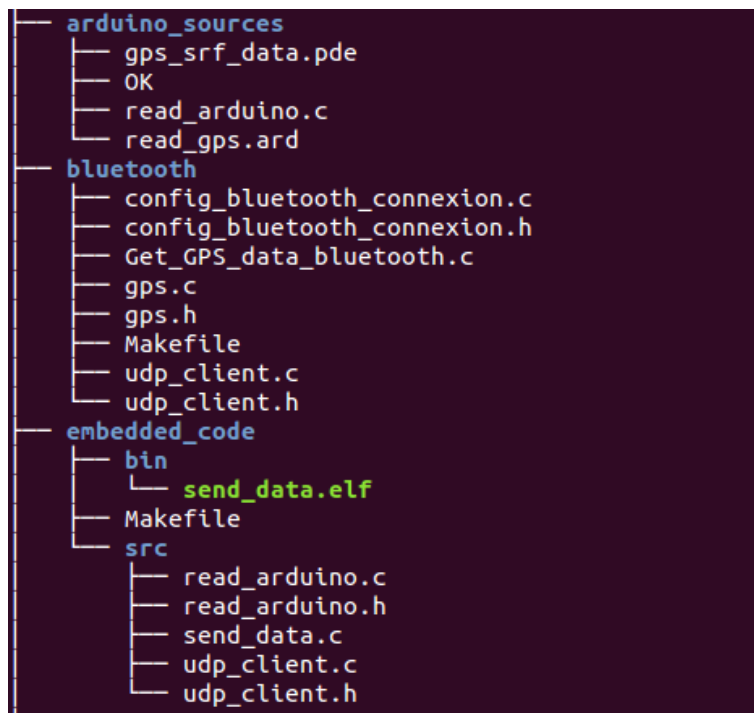


Figure 4.1: 1st part of the repository

In `arduino_sources` folder, all the code related to the arduino board can be found. The software used to program the arduino is expecting `.pde` or `.ard` files.

In the following parts, we will provide more details for each files.

The `bluetooth` folder contains C sources et headers used on the target to catch informations sent by the smartphone.

The `embedded code` section includes all the programs we created to run on the UAV embedded linux.

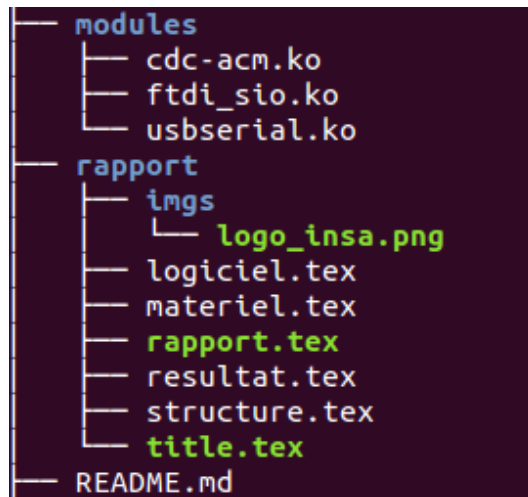


Figure 4.2: 2nd part of the repository

All the compiled modules we added on the embedded linux can be found in the module folder.

The “rapport” directory contains the sources of the documentation you are reading !

Finally, the most important folder is the one named `sdk_apps`. It contains one application named `auto_flight`. Like most of the c projects, you can find a `Makefile`, a `bin/` and a `src/` folder. The `Makefile` is configured to create the application binary and to move it to the `bin/` folder. You can find additionnal informations about the `Makefile` in the appedices.

About the `src/` folder :

The most important C file is named `ardrone_testing_tool.c`. This file is making the link between our application and the ARdrone library. Moreover, it is launching and joining all our threads.

The `Auto/` folder contains code related to the `auto_control` thread.

The `Avoidance/` folder contains code related to the `avoidance` thread.

The `Comm/` folder contains code related to the `receive_gps` thread.

The `Comm_target/` folder contains code related to the `gps_target` thread.

The `Control/` folder contains an small library we wrote to handle UAV travelling.

The `GPS/` folder contains algorithms used to manipulate GPS strings and make distance and angle calculation.

The `Navdata/` folder contains the fonctions used to read and store navdata sent by the UAV.

The `STMachine/` folder contains the fonctions generated by SCADÉ KCG compiler.

The `Target/` folder contains code related to the `target` thread.

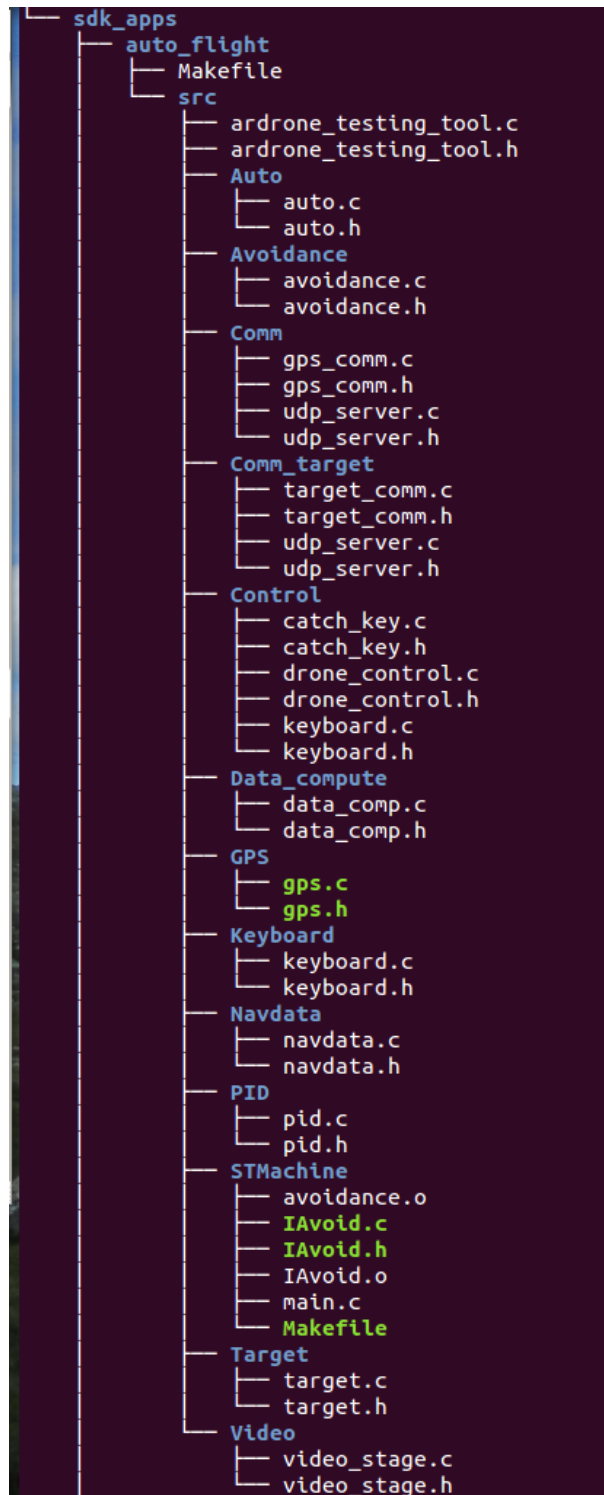


Figure 4.3: 3rd part of the repository

Thread name	Period	Description
ardrone_control	2	Library related thread
navdata_update	20	Library related thread
auto_control	20	Thread allowing manual command of the UAV
receive_gps	50	Thread used to received gps coordinated sent by the UAV
avoidance	60	Thread used to perform autonomous obstacle avoidance
gps_target	50	Thread used to received gps coordinated sent by the target
gps_target	50	Thread used to perform autonomous convergence to the target

This table summarizes all the thread launched in our application. In the following sections, we will detail the operations of each threads.

## 4.2 The auto thread

The auto thread is mainly based on the movement API we developped with the Brown team.

### 4.2.1 Movement API

This API is an overlay of the Parrot API, it allows more intuitive UAV control. It also provide movement quatification, which means you can quantify an order by speed, distance or duration. For example, you can emit orders like “go forward on 300 cm”.

#### Les mouvements élémentaires

Les commandes de déplacement sont définis dans le fichier Control/drone\_control.c, elles ont toutes le même format :

The travelling commands are all defined in the file Control/drone\_control.c, they have the same format :

```
C_RESULT ordre (void *arg)
```

The list of available travellings :

- turn\_left
- turn\_right
- forward
- backward
- up
- down
- right
- left
- stop

### 4.2.2 Send an order

The above order can be passed to both functions depending on whether you want to make a move that will be termed “elementary”, or a longer trip by specifying the distance.

#### Elementary move

An elementary movement allows for a sudden displacement of around 10 cm. You must use the following function:

```
C_RESULT small_move(ORDER* order)
```

Sample code :

```
small_move(turn_right);
```

## Long move

All orders to send move commands accept an argument of type `void *`. This argument must be cast to `void *` but available commands manage only the type arguments `mov_t` which includes different arguments :

```
typedef struct mov_t{
    int32_t power;      //engine power between 0 and 100
    int32_t distance;   //distance in cm
    int32_t time;       //time in usec
}mov;
```

Before sending orders, it's important to fill correctly this structure. Unused fields must be initialized to -1. To send an order, this function must be used :

```
C_RESULT send_order(ORDER* order, void *arg)
```

Sample code :

```
/* 30% on 70 cm*/
mov mv = {30, 70, -1};
send_order(backward, &mv);
```

### 4.2.3 Using of this API

In the auto thread, we need to control the UAV with the keyboard of the station. This automatic control is really useful in case of problems occuring during the automatic control.

Basically, we are just running a `scanf` in an infinite loop. Depeding on the key pressed, we just have to call the wright function of the API.

```
while (1) {
    usleep(100000);
    scanf("%c", &c);
    printf("%c\n", c);
    switch(c){
        case 'f':
            small_move(forward);
            break;
        case 'b':
            small_move(backward);
            break;
        case 'u':
            small_move(up);
            break;
        case 'd':
            small_move(down);
            break;
        case 'o':
            //small_move(left);
            printf("Batt␣:%d\n", sauv_ndata.bat_level_current);
            break;
```

Very helpful functionalities are battery control, by pressing the “o” key, and recover from emergency mode by pressing “x”. All the other command are pretty basic (“land”, “go up”, “go down”).

#### 4.2.4 Storing navdatas

We also use this thread to store the navdata received. To achieve this goal, it is necessary to declare three functions, defined in the ARDrone library :

```
/* Initialization local variables before event loop */
inline C_RESULT auto_navdata_client_init( void* data )

/* Receving navdata during the event loop */
inline C_RESULT auto_navdata_client_process( const navdata_unpacked_t*
    const navdata )

/* Relinquish the local resources after the event loop exit */
inline C_RESULT auto_navdata_client_release( void )
```

We just keep the informations we have to use later, like battery level, control state, altitude or psi angle.

```
sauv_ndata.psi_current = nd->psi / 1000;
sauv_ndata.bat_level_current = nd->vbat_flying_percentage;
sauv_ndata.ctrl_state_current = nd->ctrl_state;
sauv_ndata.tag_detected = nv->nb_detected;
sauv_ndata.tag_tab = nv->camera_source;
sauv_ndata.alt = nd->altitude / 1000.0;
```

We store all these informations in a global structure that will be accessed by the other threads.

### 4.3 The avoidance thread

This thread is mainly based on SCADE generated code. The statechart modeling the avoidance is printed below :

Then, the KCG compiler gave us some independent code, that we included in our project.

```
switch (AvoidMachine_state_sel) {
    case SSM_st_Travelling_AvoidMachine :
        outC->AvoidMachine_reset_act = inC->obstacle_detected;
        break;
    case SSM_st_Hovering_AvoidMachine :
        if (br_1_guard_AvoidMachine_Hovering) {
            outC->AvoidMachine_reset_act = 1;
        }
        else {
            outC->AvoidMachine_reset_act = br_2_guard_AvoidMachine_Hovering;
        }
        outC->init = 0;
        break;
    case SSM_st_Avoidance_up_AvoidMachine :
        if (inC->obstacle_detected) {
            outC->AvoidMachine_reset_act = 1;
        }
        else {
            outC->AvoidMachine_reset_act = br_2_guard_AvoidMachine_Avoidance_up;
        }
        outC->init2 = 0;
        break;
```

The top-level function is defined in the file IAvoid.h.



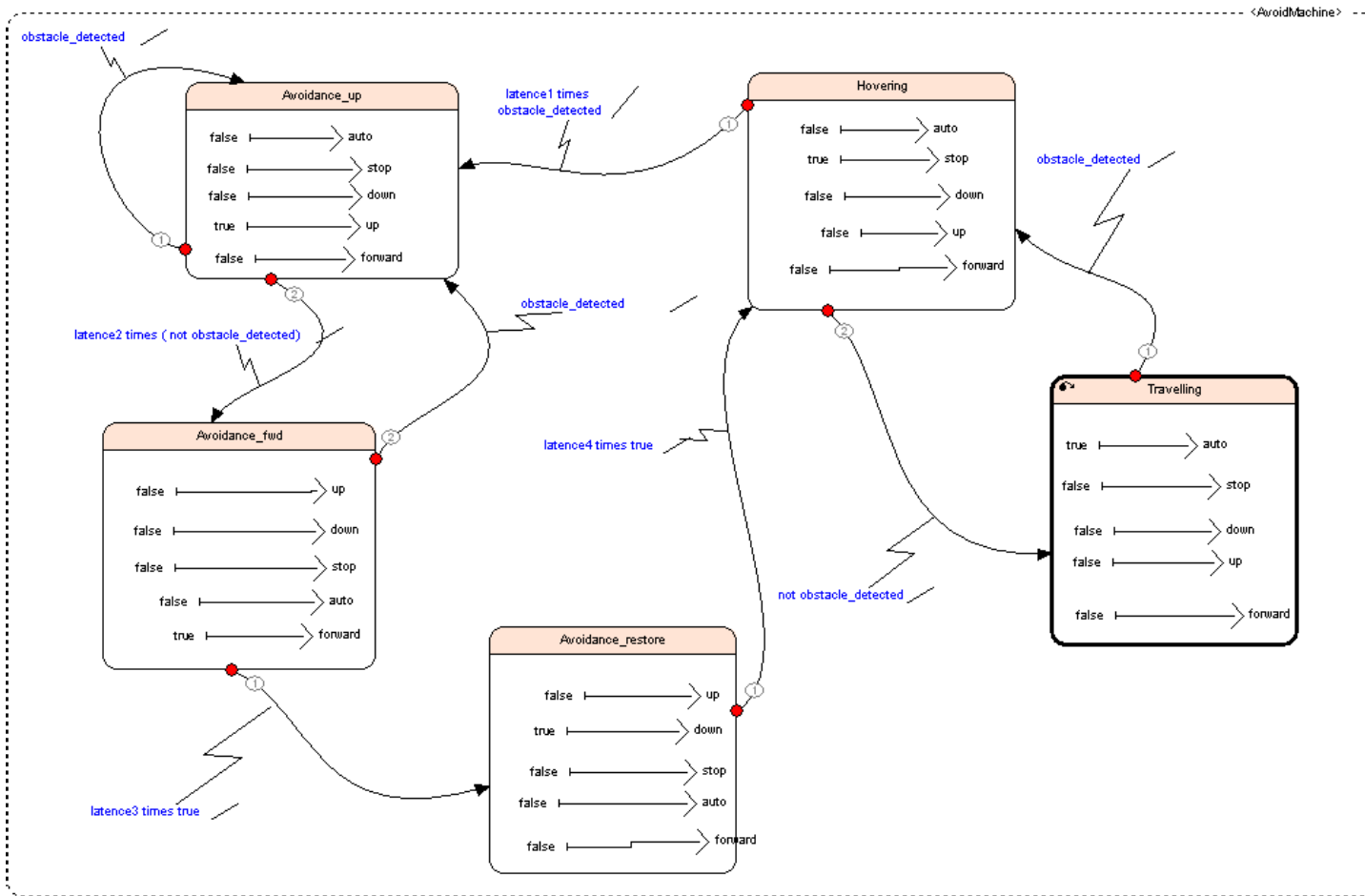


Figure 4.4: SCADE statechart

```
extern void IAvoid(inC_IAvoid *inC, outC_IAvoid *outC);
```

The inputs of this statechart are a boolean named “obstacle\_detected” (its value is 1 if there is an obstacle and 0 otherwise) and 4 integers describing latencies between transitions.

The outputs are 5 booleans : 4 orders (up, down, stop, forward) and a neutral order (if there is nothing to do).

Each call to the function represent a cycle. We have planned to execute a cycle each 10ms. The following code is illustrating the use of the auto-generated function :

```

DEFINE_THREAD_ROUTINE(avoidance, data) {

    comm_datas datas;
    double dangerThreshold=100;
    int detection;
    double average_left;
    int ret;

    init_array_obstacle_pos2();

    IAvoid_reset(&output);
  
```

```

input.obstacle_detected = 0;
input.latence1 = 100;
input.latence2 = 100;
input.latence3 = 300;
input.latence4 = 100;

while (1) {
    usleep(10);
    if (auto_ready) {
        //get srf datas
        datas = get_comm_datas();

        ret = average_obstacle_pos2(&datas.srfr, &average_left);
        printf("Moyenne capteur(%d): %f\n", ret, average_left);

        //check threshold
        if (average_left < dangerThreshold) {
            detection = 1;
        } else {
            detection = 0;
        }

        input.obstacle_detected = detection;
        IAVoid(&input, &output);
        command(output);
        usleep(10000);
    }
}

return (THREAD_RET) 0;
}

```

And the command function is :

```

void command(outC_IAvoid comm) {
    mov speed;

    if (comm.up) {
        speed.power = 8;
        send_fast_order(up, (void *)&speed);
    } else if (comm.down) {
        speed.power = 3;
        send_fast_order(down, (void *)&speed);
    } else if (comm.stop) {
        send_fast_order(stop, NULL);
    } else if (comm.auto1) {
        speed.power = 1;
        printf("avance\n");
        send_fast_order(forward, (void *)&speed);
    } else if (comm.forward) {
        speed.power = 1;
        printf("avance\n");
        send_fast_order(forward, (void *)&speed);
    }
}

```

## 4.4 The receive\_gps & gps\_target threads

Those threads are designed to receive and store the GPS strings sent by UAV and Target. The receive\_gps thread also handle SRF datas sent by the ultrasonic sensor.

We have created a small library with functions to receive udp datas. This library is described in the udp\_server.h file.

This function is reading udp datas on port 6444 (sent by the UAV).

```
/*
    Start listening the buffers sent from uav
*/
int start_comm(void)
{
    udp_struct udp_uav;
    int msglen_uav;

    if(udpserver_init(&udp_uav,UDP_UAV,1)) diep("udp_UAV_init");

    while (start_listen) {

        do {
            msglen_uav = udpserver_receive(&udp_uav, buf_uav, 512);
        } while(msglen_uav<=0);

        record_data(buf_uav);

    }

    udpserver_close(&udp_uav);
    return 0;
}
```

Then, depending on whether it is a sensor or GPS data, we fill a structure.

```
int record_data(char *buf) {
    char *gprmc_begin = "$GPRMC";
    char *gpgga_begin = "$GPGGA";
    char *srfl_begin = "$SRFL";
    char *srfr_begin = "$SRFR";

    int ret_val = 0;

    if (!strcmp(gprmc_begin,buf,6)) {
        strncpy(ret_datas.gprmc_string,buf_uav,sizeof(ret_datas.gprmc_string));
        ret_val = 1;
    } else if (!strcmp(gpgga_begin,buf,6)) {
        strncpy(ret_datas.gpgga_string,buf_uav,sizeof(ret_datas.gpgga_string));
    } else if (!strcmp(srfl_begin,buf,5)) {
        ret_datas.srfl = atof(buf+6*sizeof(char));
    } else if (!strcmp(srfr_begin,buf,5)) {
        ret_datas.srfr = atof(buf+6*sizeof(char));
    } else {
```

```

    printf("ERROR_NO_STRING_DETECTED\n");
}
return ret_val;
}

```

The other thread (receiving from target), uses similar functions.

## 4.5 The target thread

This thread contains functions related to auto-convergence. We have implemted different algorithms, described thereafter.

### 4.5.1 Turn angle algorithm

When it converges to the target, the UAV need to turn to an absolute angle between 0 et 360 from north. Depending on the UAV angle, it can be better to turn right, or to turn left, to reduce the turning duration.

The UAV will turn to reach the angle given as an argument. We have defined a tolerance. Indeed, because of transmission delays, the UAV can keep turning even if the angle is ok. A tolerance of 5.0 degrees is acceptable and always working.

```

/*
  INPUT: angle    (float)
  tolerance (float)
  This function will make uav turn to the direction of target with an error
         of +- tolerance
*/
void turn_angle2(float target_angle, float tol) {

    fdata sauv_ndata = get_ndata();
    float angle_360 = sauv_ndata.psi_current;
    float angle_inf, angle_sup;

    angle_inf = target_angle - tol;
    angle_sup = target_angle + tol;

    if (sauv_ndata.psi_current < 0) angle_360 = 360 + sauv_ndata.psi_current;
    if (target_angle < 0) target_angle += 360;

    if (target_angle > angle_360) {
        (target_angle - angle_360) < 180 ? send_order(turn_left, NULL) :
            send_order(turn_right, NULL);
    } else {
        (angle_360 - target_angle) < 180 ? send_order(turn_right, NULL) :
            send_order(turn_left, NULL);
    }

    while (!(sauv_ndata.psi_current > angle_inf && sauv_ndata.psi_current <
        angle_sup)) {
        sauv_ndata = get_ndata();
        usleep(100);
    }

    send_order(stop, NULL);
    sleep(1);
}

```

```
}
```

### 4.5.2 Convergence algorithm 1

Let's describe the first algorithm :

1. Magneto calibration (the UAV has to be flying)
2. Receive GPS datas
3. Turn to wright angle
4. Go forward 2 seconds
5. If target reached, land otherwise, jump to step 3.

```
if ( (landed == 1) && (calibration == 0) ){
    printf("start_calibration\n");
    calibrate_magneto(NULL);
    sleep(4);
    calibration = 1;
    printf("Calibration_done\n");
}

if ( (landed == 1) && (calibration == 1) && (mission == 0) ) {
    datas = get_comm_datas();
    extract_coord(datas.gprmc_string,&depart);

    datas_target = get_comm_datas_target();

    if ((check_gps_coord_struct(&depart) > 0) && (check_gps_coord_struct(&
        datas_target.dest) > 0)) {
        navigation(&depart, &datas_target.dest, &distance, &angle, NULL); //&
            relatif_error

        turn_angle2(angle ,5.0);

        if (distance > 5.0){
            fprintf(redir_sortie,"La_distance_restante_est_%f\n",distance);
            fflush(redir_sortie);
            speed.power = 3;
            send_order(forward,(void *)&speed);
            sleep(3);
        } else {
            send_order(land,NULL);
            printf("LANDING\n");
            mission = 1;
        }
    }
}
```

### 4.5.3 Convergence algorithm 2

Let's describe the second algorithm :

1. Magneto calibration (the UAV has to be flying)
2. Receive GPS datas
3. Turn to wright angle
4. While distance is decreasing, go forward
5. If target reached land, otherwise go to step 3.

```

if ( (landed == 1) && (calibration == 0) ){
    printf("start_calibration\n");
    calibrate_magneto(NULL);
    sleep(4);
    calibration = 1;
    printf("Calibration_done\n");
}

if ( (landed == 1) && (calibration == 1) && (mission == 0) ) {
    datas = get_comm_datas();
    extract_coord(datas.gprmc_string,&depart);

    datas_target = get_comm_datas_target();

    if ((check_gps_coord_struct(&depart) > 0) && (check_gps_coord_struct(&
        datas_target.dest) > 0)) {
        navigation(&depart, &datas_target.dest, &distance, &angle, NULL); //&
            relatif_error

        turn_angle2(angle,5.0);

        while (last_distance > distance) {
            fprintf(redir_sortie,"La_distance_restante_est_%f\n",distance);
            fflush(redir_sortie);
            speed.power = 3;
            send_order(forward,(void *)&speed);
            last_distance = distance;
            sleep(1);
        }

        if (distance < 10.0){
            send_order(land,NULL);
            printf("LANDING\n");
            mission = 1;
            fflush(redir_sortie);
            exit(0);
        }
    }
}

```

## 4.6 GPS algorithms

## 5 Final results

## 6 Conclusion

Conclusion