ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Waypoint navigation with an MAV

Semester Project

Adrien Briod

Section Microtechnique

SPRING 2008

Assistants:
Severin Leven
Jean-Christophe Zufferey

Laboratory of Intelligent Systems (LIS)
Prof. Dario Floreano

EPFL
Faculte Sciences et Techniques de l'ingenieur (STI)
Institut d'ingenierie des systemes (I2S)

FACULTE SCIENCES ET TECHNIQUES DE L'INGENIEUR

LABORATORY OF INTELLIGENT SYSTEMS (LIS)
Station 11
CH-1015 LAUSANNE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

DIPLOMA PROJECT --- WINTER 2006 / 2007

Title :	Waypoint Navigation with an MAV		
Candidate :	Adrien Briod	**Section :**	Microtechnique
Professor :	Dario Floreano		
Assistant 1 :	Severin Leven	**Assistant 2 :**	Jean-Christophe Zufferey

Project Summary

The goal of the project is to implement a waypoint navigation using GPS as the position input, for an outdoor MAV. A waypoint navigation allows an operator to set a number of places (waypoints) where he wants the plane to pass. The waypoint navigation implemented for this project allows the user to fix a desired heading at every waypoint.

Navigation is done thanks to the control of the turning rate. First, a homing behavior (circular path) was developed and tested. It was shown experimentally that the GPS limits the possible turning rate that can be commanded in navigation because the GPS isn't precise enough and its update rate is too low.

Dubins' theory of optimal path is then used to generate intermediate homing waypoints as shown in fig 1, so that an arc-line-arc path makes the MAV go from one waypoint to the other. Trajectories obtained during navigation tests are shown in fig 2.

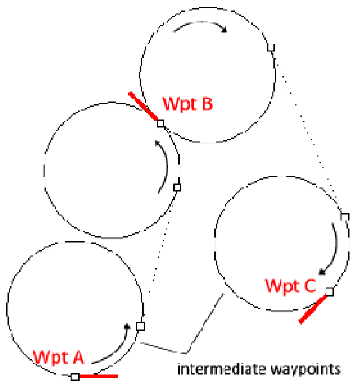


Figure 1: Intermediate homing waypoints created to build a path linking 3 waypoints.

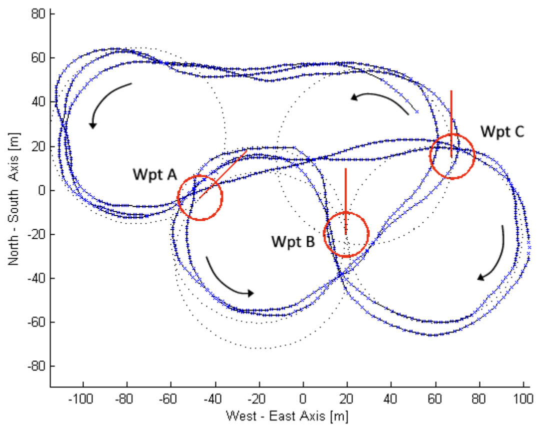


Figure 2: Example of a trajectory of the MAV given by the GPS. Sequence of 3 waypoints, with the intermediate Dubins waypoints (dashed). The MAV always reaches the waypoints with a precision lower than 10m, and the trajectory has a good repeatability.

In addition to that, landing on a precise waypoint was implemented. A line following is realized for a straight landing. The plane's waypoints can be entirely programmed from the user interface on the ground-station software to achieve autonomously a certain trajectory, and then land at a pre-defined spot. Navigation is also used in an emergency mode, making the plane go back home in case of loose of radio signal, occurring if the MAV goes out of the radio receiver's range.

Tests showed a 1s delay between the turning rate command and the realization of this command, because of the MAV's inertia. This delay is suspected to cause some oscillations. To cancel the effects of this delay, an estimation of the position 1s later is performed, so that the command can be done 1s in advance. Results are encouraging, but the method has to be tuned for an even better anticipation.

EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE LAUSANNE
POLITECNICO FEDERALE DI LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY LAUSANNE
FACULTE SCIENCES ET TECHNIQUES DE L'INGENIEUR
LABORATORY OF INTELLIGENT SYSTEMS (LIS)
CH - 1015 LAUSANNE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

PROJET DE SEMESTRE

Titre:	Waypoint Navigation with an MAV
Candidat(s):	Adrien Briod (MT)
Professeur:	Dario Floreano
Assistant 1:	Severin Leven
Assistant 2:	Jean-Christophe Zufferey

Donnée & travail demandé:

This semester project aims at implementing a waypoint navigation algorithm for a micro air vehicle (MAV) based on GPS position measurements

First, the candidate will study different strategies proposed in literature, compare them, and make a short evaluation of the most interesting algorithms with a given flight simulation.

The second and main task of the project is to implement a chosen waypoint algorithm on a real flying platform, which will be provided by the Lab. It already includes a basic electronic control system which directly permits the implementation of navigation algorithms. The second task also involves extensive flight testing and the analysis of recorded flight data.

The goal of the obtained flight controller should finally be to show an example of the existing platform's usefulness for educational purposes and to be used in an autonomous MAV competition in 2008.

Remarques:

Un plan de travail (Gantt chart) sera établi et présenté à l'assistant responsable avant la fin de la deuxième semaine.
Une présentation intermédiaire (8 minutes de présentation et 7 minutes de discussion) aura lieu le 31 mars 2008 à partir de 13 heures. Elle a pour objectifs de donner un rapide résumé du travail déjà effectué, de proposer un plan précis pour la suite du projet et d'en discuter les options principales.
Un rapport, comprenant en son début l'énoncé du travail (présent document), suivi d'un résumé d'une page, devra être rédigé. La page de résumé (A4, seulement recto) doit comporter, en plus de la description du projet et de 1 ou 2 figures représentatives, la date, le nom du laboratoire, le titre du projet, son type (projet de semestre), vos noms et prénoms ainsi que ceux du professeur responsable et des assistants. Dans le rapport, l'accent sera mis sur la description des expériences et la présentation des résultats obtenus. Une version préliminaire du rapport sera remise à l'assistant au plus tard une semaine avant la date de rendu final. La version finale sera remise à l'assistant responsable le 6 juin 2008 avant 12 heures en 2 exemplaires signés et datés.
Une défense de 20 minutes (environ 15 minutes de présentation et démonstration, plus 5 minutes de réponses aux questions) aura lieu le 11 juin 2008. Vous serez jugé sur les résultats de votre projet tels qu'ils seront présentés dans votre rapport et lors de votre défense.
Tous les documents en version informatique, y compris le rapport (en version source et en version PDF), la page de résumé au format PDF et le document de la présentation orale, ainsi que les sources des différents programmes doivent être gravés sur un CD-ROM et remis à l'assistant au plus tard lors de la défense finale.

Le professeur responsable:	L'assistant responsable:
Signature: [illegible] Dario Floreano	Signature: [illegible] Severin Leven

Lausanne, le 15 fevrier 2008

Acknowledgments
I first want to thank Prof. Dario Floreano for hosting this project at the LIS. I want to sincerely thank Severin Leven, for his expert advices, his devotion, and for his motivation he transmitted to our team. I also wish to thank Jean-Christophe Zufferey for his support, and for having encouraged the participation to a flying robot competition. Without his impulsion, our team participating to the EMAV'08 competition would not have been created. Thanks go to my team partners, Alex, David and Laurent, for their enthusiasm, the great ambiance, and their advices concerning this project. A special thank to Severin and Alex for piloting the MAV, without their help and patience the numerous outdoor tests would not have been possible. Thanks go to the LIS, for supporting and sponsoring our team's participation to the EMAV'08 competition, and to Anouk Hein, for helping us organizing the trip to Germany. Finally, I want to thank the LIS members in general for their friendship and the good atmosphere, that helped me and motivated me during my project.

Contents

1 Introduction

Micro Air Vehicles (MAVs) are used - or are to be used - in many applications, such as terrain exploration, surveillance, or assistance during disasters. In many applications, the MAVs tend to be autonomous so that many of them can be used simultaneously for example. Of course, one of the key feature is here the management of where the plane goes, that is to say: navigation.

Navigation can make the plane follow a particular way (follow waypoints for example), or accomplish different behaviors (turn around a place, land,...). To achieve this, at least one controller system's inputs has to contain information about the plane's position. The goal of the present semester project is to implement a waypoint navigation using GPS as the position input, for an outdoor MAV. A waypoint navigation allows an operator to set a number of places (waypoints) where he wants the plane to pass (see fig 1). The algorithms will be implemented and tested on the LIS' platform used in the SMAVNET project [6], and one application will be the use at the EMAV'08 competition.

Dealing with navigation for an MAV implies very different challenges from what one can be used to with mobile ground-robots. For example, on the ground, navigation has a lot to do with obstacle avoidance, or with terrain disturbances management. In the air, an MAV is most of the time higher than any obstacle, but has to deal with wind. Another

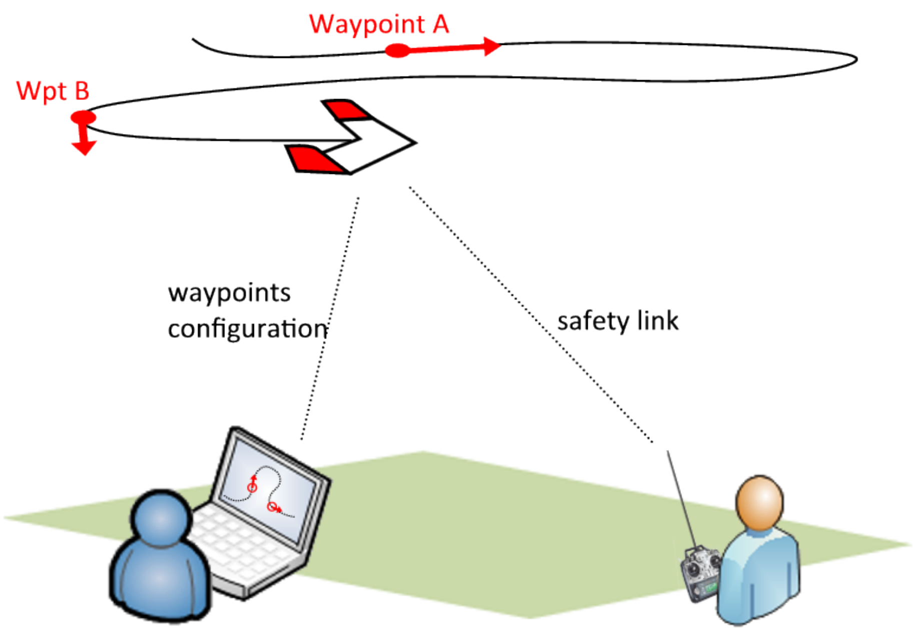


Figure 1: Principle of waypoint navigation : a sequence of waypoints is set from the ground station thanks to a wireless link, and the MAV follows the good path to reach the waypoints. The waypoints are configured with heading commands, which means that the plane has to have a certain heading when reaching the waypoint

main difference is the dynamics of the plane, which is very important to be taken into account: for example, an MAV has a turn radius limited by its geometry and by the need to remain stable. Plus, an MAV of course cannot stop in the air, to turn on itself, or while it's taking a decision, but it is constantly flying at a typical speed of 10-15 m/s.

The waypoint navigation we want to implement for this project should allow the user to fix a desired heading at every waypoint. This is useful in many situations, such as when we want the MAV to be aligned in the good direction for a straight landing. Plus, the possibility to fix a heading constraint on waypoints allows to program a trajectory quite precisely, even if the user has not the possibility to draw the full trajectory.

2 The Platform

The platform used in this project for the implementation of navigation is briefly presented in the following sections. It is developed by Severin Leven for the SMAVNET project [6].

2.1 Physical frame

The platform (see fig 2) is a 80cm fixed-wing plane, made of EPP. A Hyperion HP-Z2205 brushless motor drives the unique propeller, and two E-flight servos activate the ailerons present on each side of the wing, for altitude and direction control. A 35 MHz radio receiver on the plane permits the reception of signals from a remote controller. These signals are treated by the flight controller (FC), and can be either transmitted directly as commands to the actuators (motor and servos), so that the plane is commanded manually, or the remote controller can activate an automatic mode so that the plane is commanded by the FC. A switch on the remote controller can make the plane go in manual or automatic mode, so that the pilot can command the plane manually at anytime by actuating the switch.

The FC comprises a printed circuit board with a dsPIC33 microcontroller as the main processor, to which a few sensors are connected (see 2.2). A 3 cell Lithium Polymer battery powers all the components, and provides an autonomy of about 30 min. The total weight of the plane is around 280g, depending on what component is added or not.

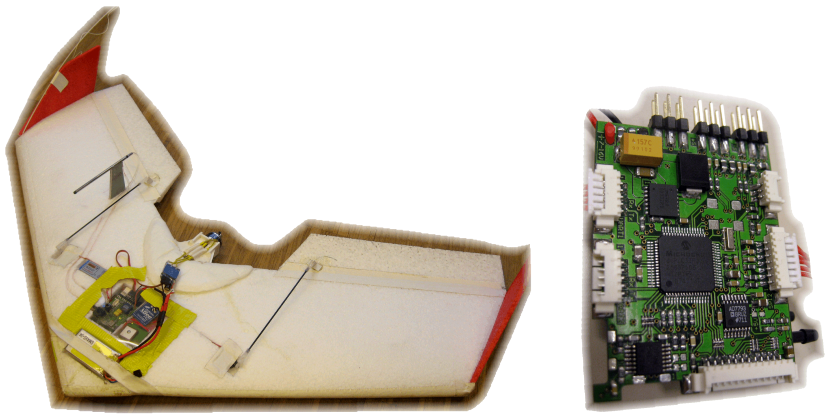


Figure 2: left: EPP flying wing, equipped with the actuators, all sensors, and the flight controller; right: Flight controller with dsPIC.

2.2 Low-level sensors

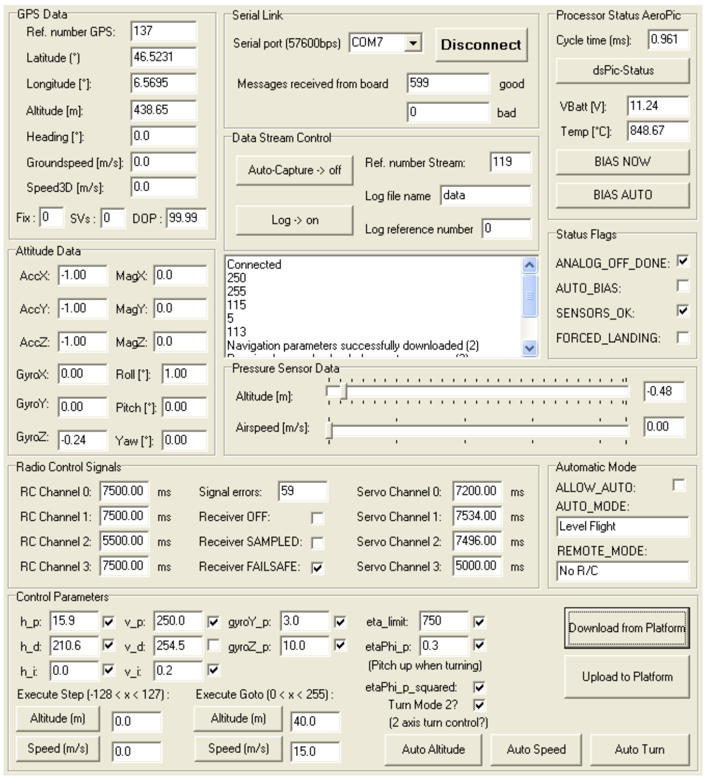
The sensors used by the FC for the low-level control are the following: 2 gyroscopes for the Y (pitch) and Z (yaw) axis, a static pressure sensor, and a dynamic pressure sensor. Both gyroscopes are used for the turn control. The static pressure sensor is used for the altitude control, and the dynamic pressure sensor for the speed control. The altitude and speed controls are PIDs, whose gains were tuned manually. The turn control is more complex, and relies on the roll angle estimation.

2.3 GPS

The GPS module contains an U-blox Antaris LEA-4H, that can provide four position messages per second. The GPS measures its position only once per second, and the 4Hz rate is obtained by interpolation. The GPS is configured to send to the FC messages providing information on position, speed and precision of measure. The GPS antenna is a small 15x15mm ceramic patch.

2.4 Communication

The FC can communicate with a ground station thanks to a 2.4GHz Xbee module, establishing a UART wireless link. A monitor, programmed in C++, provides a few functionalities, such as the possibility to send commands to the FC or receive messages. For example, a functionality activates the continuous sending of different values, so that flight data can be logged on the computer and analyzed afterward. Or flight parameters can be edited in the monitor, and uploaded to the FC, to test different settings.


GPS Data
Ref. number GPS: 137
Latitude [°]: 46.5231
Longitude [°]: 6.5695
Altitude [m]: 438.65
Heading [°]: 0.0
Groundspeed [m/s]: 0.0
Speed3D [m/s]: 0.0
Fix : 0 SVs : 0 DOP : 99.99

Serial Link
Serial port (57600bps) COM7
Messages received from board 599 good
0 bad

Data Stream Control
Ref. number Stream: 119
Log file name data
Log reference number 0

```
Connected
250
255
115
5
113
Navigation parameters successfully downloaded (2)
```

Processor Status AeroPic
Cycle time [ms]: 0.961
VBatt [V]: 11.24
Temp [°C]: 848.67

Status Flags
ANALOG_OFF_DONE: ✓
AUTO_BIAS: ☐
SENSORS_OK: ✓
FORCED_LANDING: ☐

Attitude Data
AccX: -1.00 MagX: 0.0
AccY: -1.00 MagY: 0.0
AccZ: -1.00 MagZ: 0.0
GyroX: 0.00 Roll [°]: 1.00
GyroY: 0.00 Pitch [°]: 0.00
GyroZ: -0.24 Yaw [°]: 0.00

Pressure Sensor Data
Altitude [m]: -0.48
Airspeed [m/s]: 0.00

Radio Control Signals
RC Channel 0: 7500.00 ms Signal errors: 59 Servo Channel 0: 7200.00 ms
RC Channel 1: 7500.00 ms Receiver OFF: ☐ Servo Channel 1: 7534.00 ms
RC Channel 2: 5500.00 ms Receiver SAMPLED: ☐ Servo Channel 2: 7496.00 ms
RC Channel 3: 7500.00 ms Receiver FAILSAFE: ✓ Servo Channel 3: 5000.00 ms

Automatic Mode
ALLOW_AUTO: ☐
AUTO_MODE: Level Flight
REMOTE_MODE: No R/C

Control Parameters
h_p: 15.9 ✓ v_p: 250.0 ✓ gyroY_p: 3.0 ✓ eta_limit: 750 ✓
h_d: 210.6 ✓ v_d: 254.5 ☐ gyroZ_p: 10.0 ✓ etaPhi_p: 0.3 ✓
h_i: 0.0 ✓ v_i: 0.2 ✓ (Pitch up when turning)
etaPhi_p_squared: ✓
Turn Mode 2? ✓ (2 axis turn control?)
Execute Step (-128 < x < 127) : Altitude (m) 0.0 Speed (m/s) 0.0
Execute Goto (0 < x < 255) : Altitude (m) 40.0 Speed (m/s) 15.0

Figure 3: Original ground station program, used to monitor and log the flight data, and to set the low-level control parameters

3 Navigation

Before navigation is introduced, the general control of the MAV is described. Then, solutions of basic navigation tasks are presented, and a navigation solution is described. Finally the Matlab tool created to simulate the MAV control and test the navigation algorithsm is presented.

3.1 MAV control

The control is separated in two types: low-level and high-level control. The role of high-level control is to drive the plane from a place to another, make it land or achieve a task. It sets the low-level's commands in order to achieve these tasks. The available low-level commands are the following:

- h_c: altitude command (in m)

- v_c: speed command (in m/s)

- $\dot{\Psi}_c$: turning rate command (in °/s)

The low-level control, which manages to make the plane reach these commands, is already present on the platform and the present project mostly deals with the high-level part. More precisely, the focus is really on navigation - the control of the plane's trajectory - although other high-level controls for different tasks (such as landing) are demonstrated.

The navigation is achieved thanks to the command of the turning rate ($\dot{\Psi}$) only, for the trajectory control. To sum up, the navigation algorithm has the GPS data as input, and must return $\dot{\Psi}_c$. Fig 4 shows the complete control loop.

3.2 Trajectory control

The trajectory of the MAV shown in fig 5 is constrained by the following equations, describing the non-holonomic displacement:

$$\begin{aligned} \dot{x} &= V \cdot \sin \Psi && (3.1) \\ \dot{y} &= V \cdot \cos \Psi && (3.2) \\ \dot{\psi} &= \omega && (3.3) \end{aligned}$$

The turning rate ω is limited by the dynamic capability of the MAV. The turning rate in function of the roll angle Φ and the speed V can be approximatively described by the following formula:

$$\omega = \frac{g}{V} \cdot \tan \Phi \qquad (3.4)$$

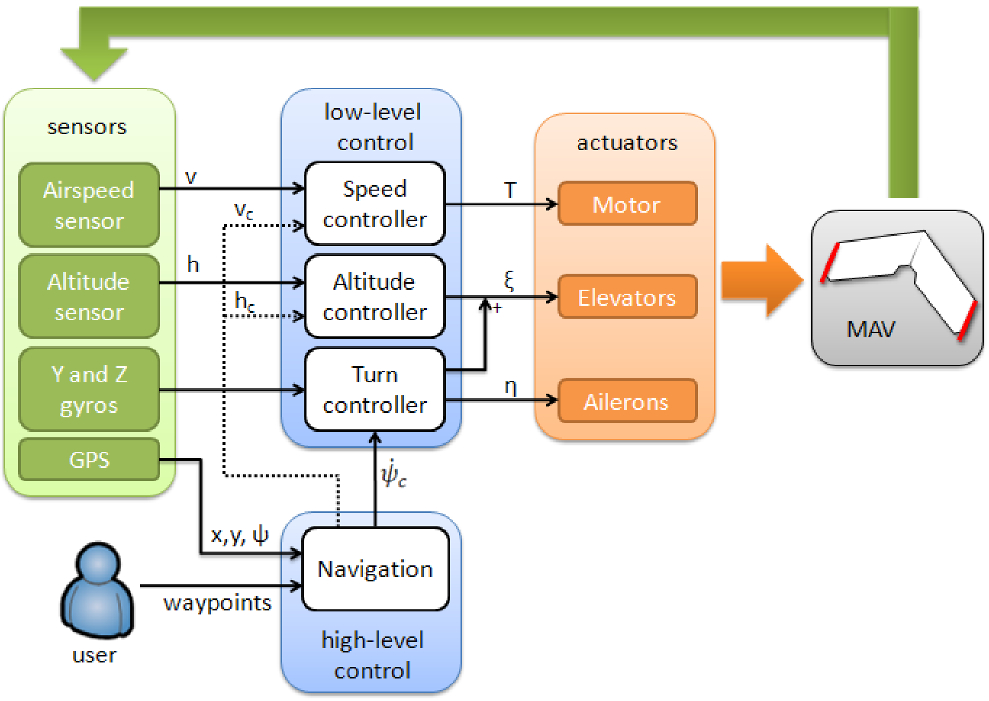


Figure 4: Block-diagram of the control loop implemented in the flight controller. High-level control comprises navigation, which calculates a turning rate command $\dot{\psi}_c$ for trajectory control. Speed and altitude commands (v_c, h_c) may be set as well. The commands are treated by the low-level control.

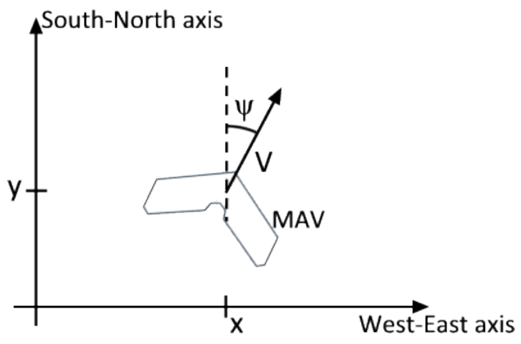


Figure 5: The plane's parameters for the trajectory's dynamics are the position (x,y), the velocity (V) and the heading (Ψ). The effect of wind is not developed here.

And the turn radius is dependent on the turning rate in the following manner:

$$R = \frac{V}{\omega} \tag{3.5}$$

The combination of equations 3.4 and 3.5 gives:

$$R = \frac{V^2}{g \cdot \tan \Phi} \tag{3.6}$$

To maintain a good stability, the maximum acceptable roll angle Φ was determined to be of about 45°. This allows a maximum turning rate of $\omega_{max} \cong 56\,^\circ/s$ at $V = 10m/s$ or $\omega_{max} \cong 37\,^\circ/s$ at $V = 15m/s$. The respective maximum turn radiuses are $R \cong 10m$ at $V = 10m/s$ or $R \cong 23m$ at $V = 15m/s$.

3.3 Use of GPS

A simple GPS module is used in navigation to provide the position (x,y) of the MAV and its heading (Ψ). Another module providing such information could have been an IMU (Inertial measurement unit). An IMU includes 9 sensors and often a GPS. It can estimate much more precisely and faster the position, speed and orientation of the plane, thanks to sensors fusion (often a Kalman filter). The biggest disadvantages of the IMU are its cost and weight, and that is why only fewer sensors are used for this project's platform, and only GPS will be used for navigation. This choice implies a slower position update (every 250ms), a precision of a few meters, and a certain delay (the provided position and heading are the ones the plane had a few ms before). Therefore, the control has to deal with these limitations, especially with the 0.25s sampling rate.

3.4 Homing solutions

Before the full waypoint navigation is presented, a few navigation functions are presented, like homing or line following. These simpler behaviors are useful in many cases, for example homing can be used to make the plane stay on a certain place for observation or before it receives instructions. In addition to that, these basic functions are important building blocks as they will be used to achieve more complex trajectories.

Homing is the action of turning in circles around a fixed point (or home). The navigation has to make the MAV follow a circle of radius R. To achieve that goal, many control strategies exist. The first one presented is simply computing a desired heading (Ψ_d) for the plane, and the turning rate command ($\dot{\Psi}_c$) is proportional to the error with the actual heading (Ψ), with a gain K that has to be tuned:

$$\dot{\Psi}_c = K \cdot (\Psi_d - \Psi) \tag{3.7}$$

The second solution presented computes a desired heading and a desired turning rate ($\dot{\Psi}_d$). The turning rate command is calculated as follow:

$$\dot{\Psi}_c = \dot{\Psi}_d + K \cdot (\Psi_d - \Psi) \tag{3.8}$$

This last solution is a way to generate a kind of a priori control. For example, when the plane is right on the circle, $\dot{\Psi}_d$ is equal to $\frac{V}{R}$, and in theory, the plane will follow the circle without the need of heading corrections from the $K \cdot (\Psi_d - \Psi)$ term. This solution is useful in the case of circle following, because from the trajectory control point of view, it is almost like receiving a ramp command, and without an a priori control a steady-state error appears.

3.4.1 Heading along tangent

This first approach is an intuitive control, that calculates a desired heading for the plane so that its direction is along the tangent of the circle to follow, as shown in fig 6. Equations to calculate Ψ_d are described in annexe 8.1.1. Control law 3.7 is then used to calculate the turning rate command.

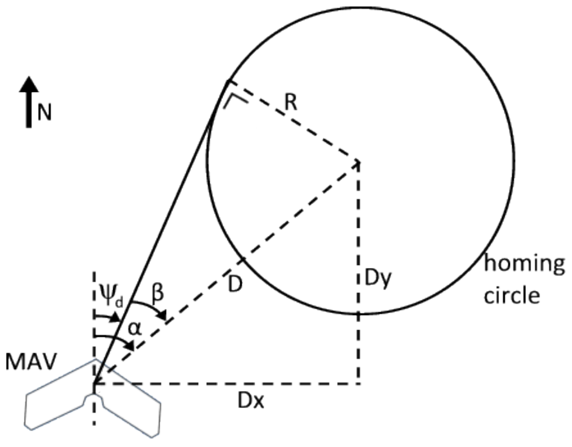


Figure 6: Ψ_d is the desired heading, directed along the tangent of the homing circle

3.4.2 Vector field

This method, described in [1], calculates a vector field in the whole space, giving a desired heading (Ψ_d) for the MAV so that it asymptotically reaches the circle and stays on it. In addition to that, the desired heading field is derived on the whole space to give a desired turning rate ($\dot{\Psi}_d$) for the plane. The formulas are presented in annexe 8.1.2. Figure 7 shows the resulting vector field, and figure 8 shows the a priori turning rate ($\dot{\Psi}_d$), in function of the distance from circle's center.

3.5 Line following solution

When the plane has to fly along straight lines, a line following algorithm is used. For example, it is useful for the approach phase of a straight landing.

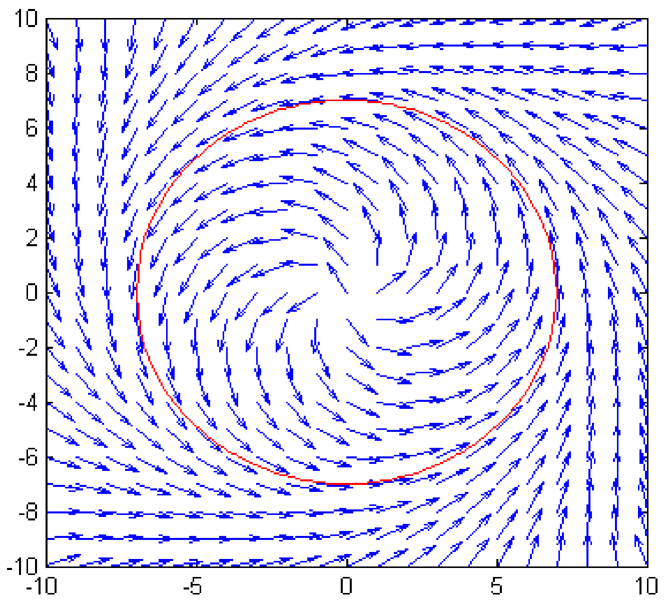


Figure 7: Vector field indicating the desired heading of the plane for every point of the space, to make the trajectory tend to a circle

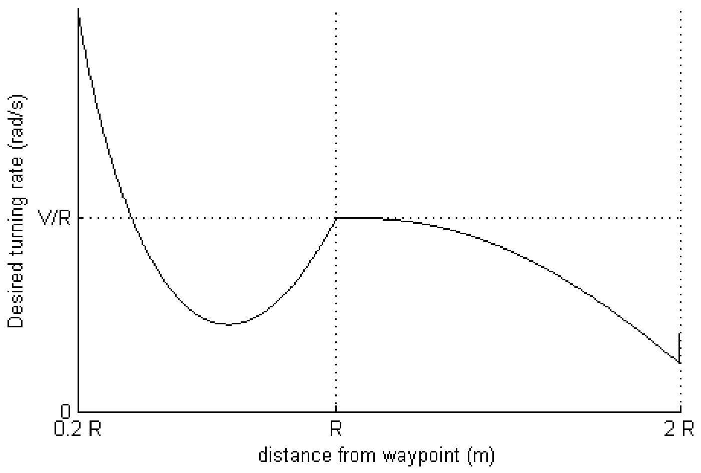


Figure 8: A priori desired turning rate $|\dot{\Psi}_d|$ in function of distance D from center of circle of radius R, calculated by derivating the vector field shown in fig 7. $\dot{\Psi}_d$ is positive for a clockwise turn, and negative for a counter-clockwise turn. We remark that when the MAV is on the circle $(d = R)$, the desired turning rate is $fracVR$.

Like for homing (see §3.4.2), an algorithm generating a vector field, described in [1], is used. It makes the plane asymptotically fly along the line. The formulas are presented in annexe 8.1.3 and fig 9 shows the resulting vector field.

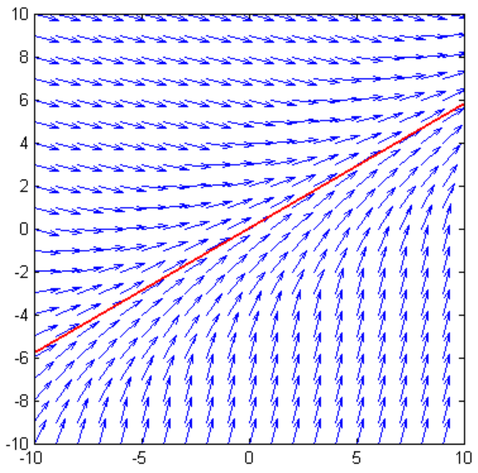


Figure 9: Vector field indicating the desired heading of the plane for every point of the space, to make the trajectory of the MAV tend to a line

3.6 Waypoint navigation solutions

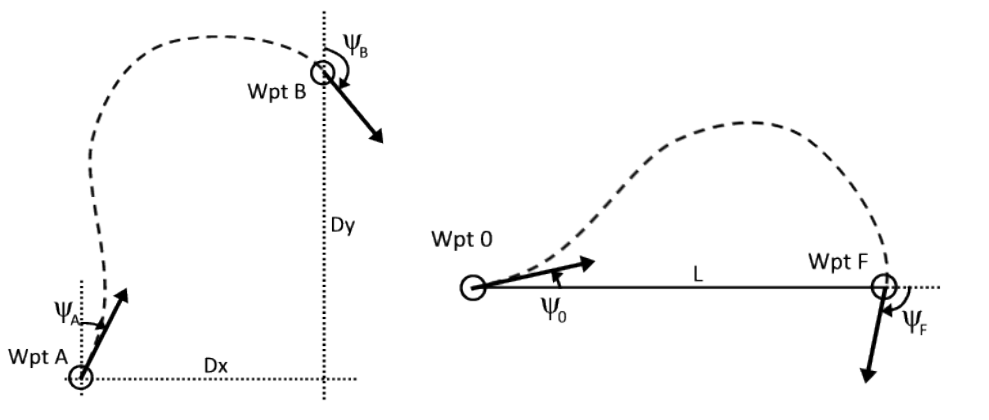
The waypoint navigation we decided to implement in this project implies to find a trajectory so that the plane has a predetermined heading when passing on the waypoint. To synthesize, a coordinate change is realized to have only three variables describing the problem. The coordinate change is described in annexe 8.1.4, and fig 10(b) shows the new situation. The problem is therefore described by:

- **L:** Distance between the waypoints
- Ψ_0**:** Heading of waypoint 0, relatively to the line between both waypoints
- Ψ_F**:** Heading of waypoint F, relatively to the line between both waypoints

3.6.1 The different approaches

To make the MAV go from one waypoint to another, many different approaches exist. They can usually be classified into two categories

1. The computation of the command is made in function of the instantaneous position and condition
2. An initial trajectory planning is done, then the commands try to achieve the trajectory following



(a) General case (b) Three variables problem, after coordinate change

Figure 10: Example of a trajectory, linking two waypoints (A and B) having pre-defined headings (ψ_A, ψ_B)

The first category can be qualified of behavior-based control, because these are often simple rules that determine which direction (or behavior) has to be taken. For example, [2] describes a control strategy choosing between turning right or left, in function of the waypoint's position relatively to the MAV, as described in fig 11. The advantage of the behavior-based approach is its robustness, because it can adapt to any situation, and accept strong disturbance. Indeed, if wind pushes the MAV further than anticipated, the plane reacts with a behavior adapted to its new position, since the control is done only thanks to the immediate inputs. The counter part is a sometimes unpredictable comportment, and therefore a loss of control for the user.

The second category is more complicated to implement, because a feasible path from the plane to the waypoint has first to be computed and stored in memory. Then the control has to determine where on the path is the MAV, and compute the commands so that it follows the path. This way of doing is more complicated, but probably more powerful, in the sense it allows to create more complex trajectories than the first approach. Often, more constraints than in the simple case shown in fig 11 are present, and slightly complex trajectories are pretty hard to be achieved with only simple rules based on the direct inputs whereas, in some cases, using path planning is easier.

A way of mixing the advantage of both approaches is a constant re-calculation of the path, to account for the disturbances endured by the plane and therefore have a good robustness, with a still high level of complexity possible for the trajectory. In this case, Wpt 0 of fig 10(b) represents the plane, and the trajectory is constantly planned so that it links the plane to the next waypoint.

For the waypoint navigation implemented in this project, because of the heading constraints fixed at each waypoint, the best approach is path planning. Indeed, it is very difficult to classify different situations and find rules for each combination of the three variables shown in fig 10(b) (assuming that Wpt 0 represents the plane).

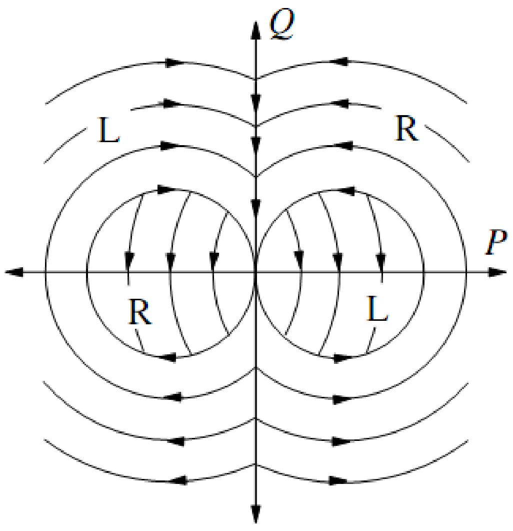


Figure 11: Phase portrait of the control strategy found in [2]. In function of the waypoint position (coordinates P, Q) relatively to the plane (center of figure, heading directed upward), the turn direction is chosen (R for right, L for left), to make the plane eventually reach the waypoint.

To find the best path linking two waypoints with heading constraints, a few solutions exist and are presented in the next sections.

3.6.2 Shortpath and Lowturning

Paper [3] proposes an approach limiting the path's length as well as the turning amplitude. For this, a trajectory formed by one line and one arc is calculated. Detailed forumlas are described in annexe 8.1.5, and examples of trajectories are shown in fig 12.

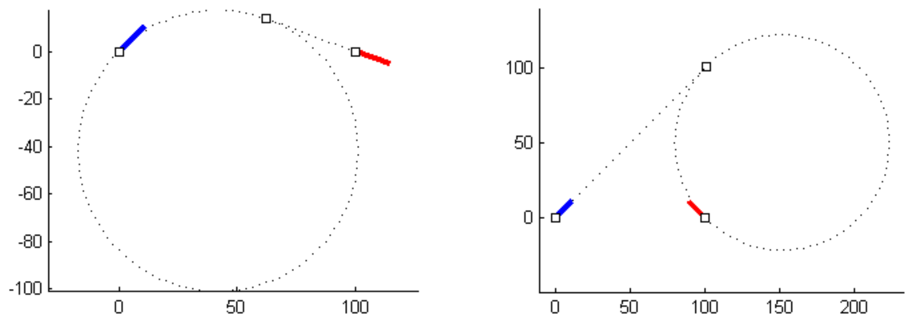


Figure 12: Examples of arc-line and line-arc trajectories, using theory of [3]

3.6.3 Dubins' solution

Dubins' theorem [5] establishes that the minimum time optimal path is either an arc-line-arc, either an arc-arc-arc solution. In [4], this theory is used to construct the optimal path. The idea is to build two pairs of circles tangent to the waypoints' directions, with acceptable circle radiuses for a good flight dynamic (see §3.2). Z_0 and Y_0 are the counterclockwise and clockwise oriented circles tangent to first waypoint's direction, and Z_F and Y_F are the counterclockwise and clockwise oriented circles tangent to second waypoint's direction. Then, 2 to 4 solutions for trajectories linking Z_0 or Y_0 to Z_F or Y_F exist, and the optimal path is the shortest of these solutions, as shown in fig 13.

In some particular cases when the waypoints are quite close from each other, the optimal path is an arc-arc-arc solution (as in fig 14 for example). These cases are not treated here, because the arc-line-arc solution is most of the time optimal, and always provides at least one solution. The equations used to calculate the optimal trajectory are described in annexe 8.1.6.

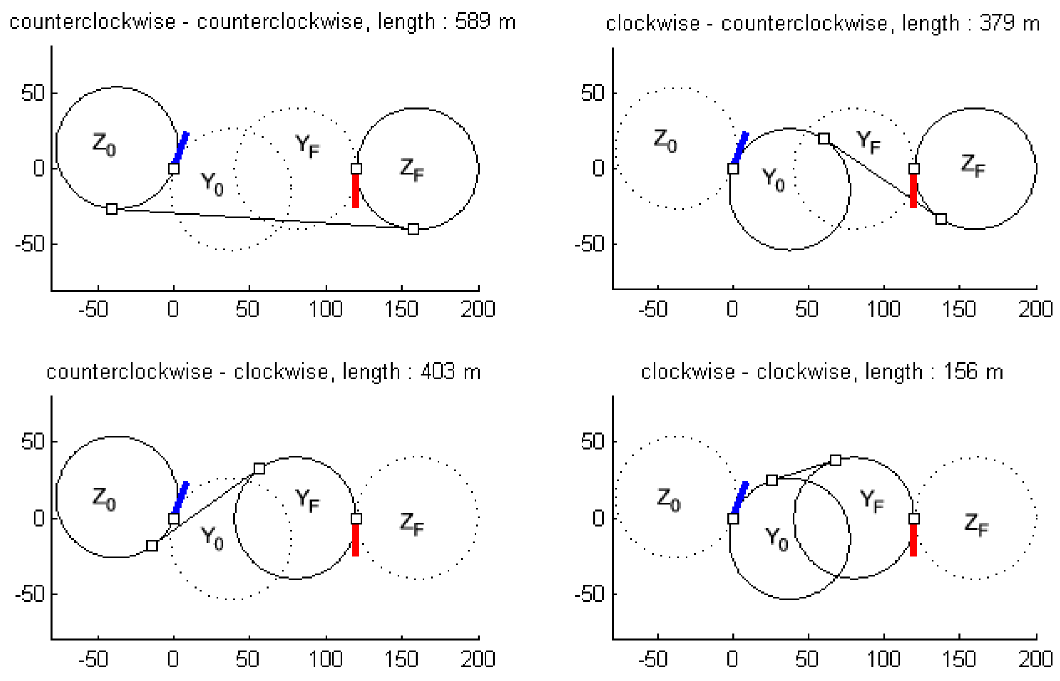


Figure 13: Example of the 4 arc-line-arc solutions for one configuration of waypoints ($\Psi_0 = 70°, \Psi_F = -90°, L = 120m$), with optimal path corresponding to the shortest solution (clockwise-clockwise in this example)

3.6.4 Final solution

Initially, the Shortpath and Lowturning approach described in section 3.6.2 was chosen for it's simpleness. But in many cases, bad trajectories are proposed because this solution

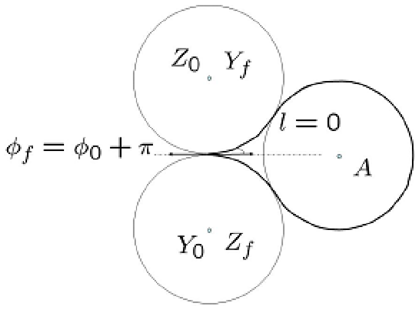


Figure 14: Image taken from [4]: example of arc-arc-arc optimal solution

is limited to certain placements of waypoints, as it can be seen in fig 15. An adaptation is proposed in [3] to treat the down-left example of fig 15, but other improvements should have been realized to treat correctly all cases.

On the other hand, Dubins' solution always shows good and feasible trajectories. This is why this solution was finally chosen and implemented for the waypoint navigation. Homing solutions presented in section 3.4 can be used to achieve the circular parts of the trajectory, as well as the line following solution of section 3.5 for the straight part.

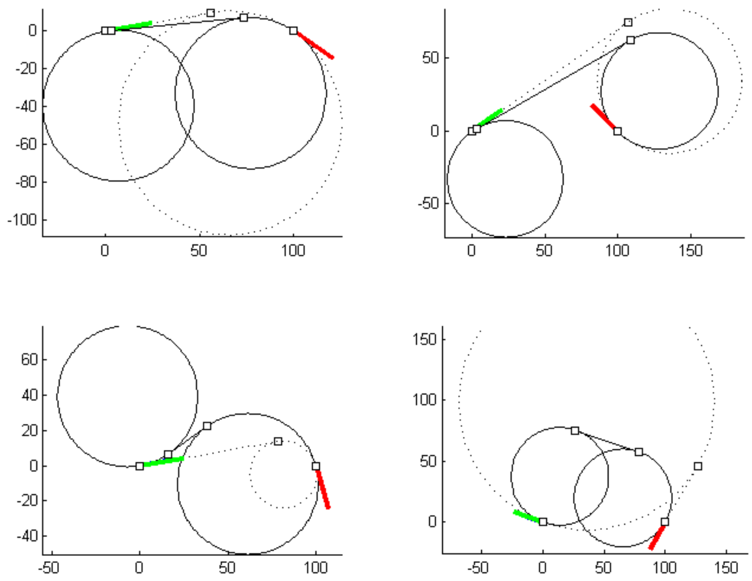


Figure 15: Comparison between waypoint navigation solutions 3.6.2 and 3.6.3. The Shortpath and Lowturning solution (dashed lines) shows appropriate results in both first examples (top), but a too small radius is generated for third case (down left) and the path's length of last example (down right) is really not optimal. We can see that Dubins' solution (continuous line) shows good results in all cases.

3.7 Matlab implementation

All solutions implemented in the flight controller for navigation were first tested in Matlab. The first reason is because debugging is much easier on a PC than on the microcontroller, the second reason is for the possibility to visualize the generated vector fields or trajectories, to see if the algorithms really match the initial idea. The figures of the present chapter showing the algorithm's generated vector fields, or generated paths demonstrate the use of a Matlab implementation for visualization purposes.

Efforts were done so that a very simple transition was possible from Matlab to the C program running the flight controller. For example, the functions generating the path are tested in Matlab by plotting the intermediate waypoints in many configurations, and the same functions are then used in the flight controller. The same variable names are used, and the algorithms written in Matlab have to suffer of only little changes to be C-compatible. For example, accolades have to be added to if/else loops, variables have to be declared, and *cos()* functions transform into *cosf()* functions. This way of doing could limit the flight controller debug time.

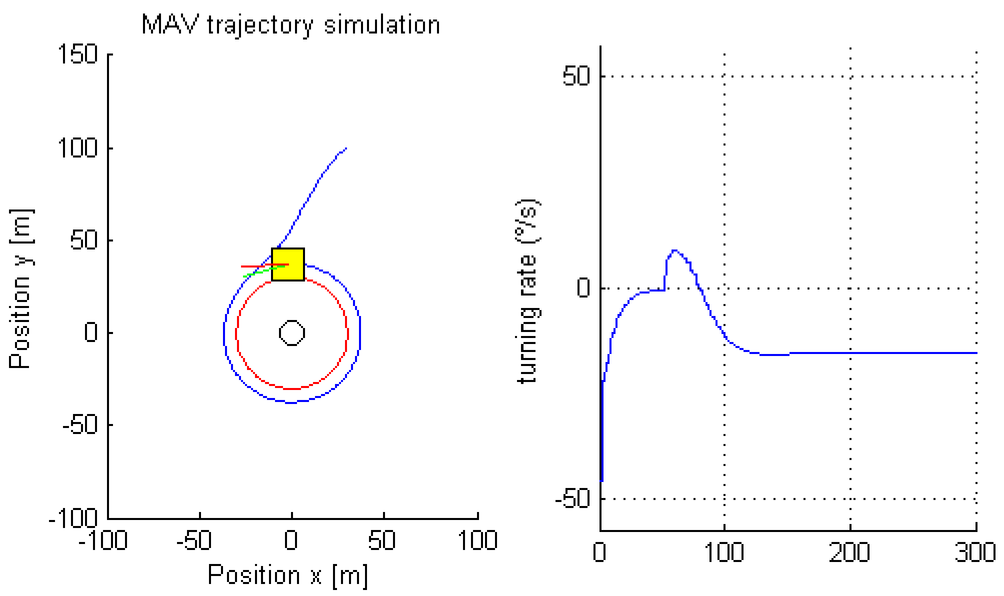
In addition to that, a small Matlab program was developed to simulate the trajectory of the MAV, in function of the turning rate command and the wind (W_x and W_y). The simulation is based on the following equations :

$$\Psi = \Psi + \dot{\Psi}_c \Delta t \tag{3.9}$$

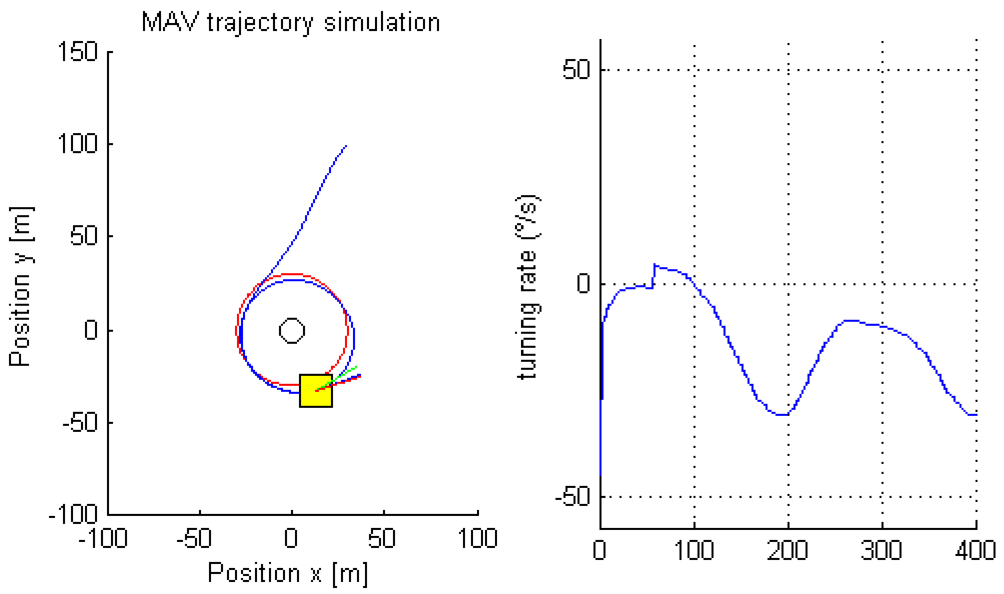
$$x = x + V\sin(\Psi)\Delta t + W_x \tag{3.10}$$

$$y = y + V\cos(\Psi)\Delta t + W_y \tag{3.11}$$

With Δt a small time period. Fig 16 shows examples of simulations that can be done, to test algorithms, or to understand effects of parameters.



(a) Homing without the a priori $\dot{\Psi}_d$ command. The steady state error is visible, and it is easy to understand the role of the a priori term



(b) Homing with the a priori $\dot{\Psi}_d$ command, and 3m/s wind in East direction.

Figure 16: Example of trajectory simulation with different parameters or algorithms, with plot of the commanded turning rate. These simulations allow to understand quickly the effect of different configurations.

4 Embedded programming

The navigation had to be integrated in the existing flight controller, which had to be modified in some points. One important aspect of navigation is of course the user input, when one wants to set the waypoints for example. This is why modifications relative to communication between the flight controller and the ground station had to be realized, involving the modules *usart1.c* and *dma.c* (see annexe 8.2.1). Configuration of navigation's algorithms and the sending of flight data for navigation debug were also made easier. The high-level control (in *navigation.c*) is called just before the low-level control (in *flightcontrol.c*) and sets the desired commands at every GPS update (4Hz). For clarity, separated modules were created to manage the waypoints (*waypoint.c*) and Dubins' path planning (*Dubins.c*). Figure 17 sums up the program's organization.

Navigation's main objective is to achieve the route by going through all waypoints. Standard waypoints are of type "pass" or "homing", which are used to realize the trajectories (passing over or turning around a waypoint). But some more functionalities were implemented as well, such as landing on a specific location, realizing an action when arriving on the waypoint (dropping an object), or an emergency mode where the plane comes back home if no more radio signal is received.

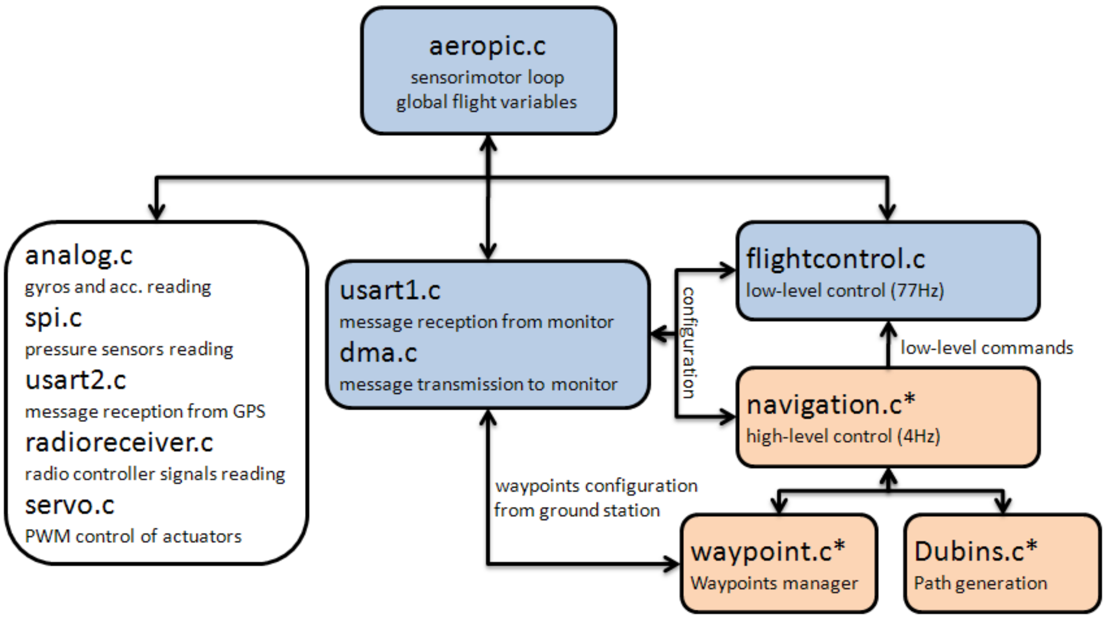


Figure 17: Diagram showing the hierarchical organization of the flight controller's software. The navigation blocks (with the *) were added to the original architecture. Mainly the blue blocks were modified during integration.

4.1 Waypoint manager

The waypoints are stored in structures, which can be modified thanks to the ground station interface. A table of 10 waypoint structures is stored in the flight controller, since it is not possible to allocate memory dynamically for a variable number of waypoints. The structure's variables are mainly the waypoints' parameters, which are modified by the user. The parameters of the main waypoint structure are the following:

- **id, nb_wpts:** The id and total number of waypoints, used to identify a waypoint when it is transmitted from or to ground station
- **type:** Indicates the type of waypoint:
  - **homing:** The waypoint is the center of a circle along which the MAV has to fly
  - **pass:** The waypoint has to be passed over by the MAV, with a given heading on waypoint
  - **land:** The waypoint is the place where the MAV has to land
  - **drop:** An object has to be dropped when the waypoint is passed over
- **alt_act, speed_act:** Tells if the values given for altitude and speed have to be commanded when the MAV is flying to the waypoint
- **turndir:** In case of homing, indicates the turning direction
- **wpt_act:** Tells if the waypoint is enabled or not (if it is included in the route or not)
- **next_auto:** Tells if the MAV should go to next waypoint automatically when this waypoint is achieved
- **Longitude, Latitude.** Coordinates of the waypoint
- **altitude, speed:** If activated (see *param*), these values are altitude or speed commands
- **radius:** In case of homing, gives the radius of the circle the plane should follow. In the other cases, this value gives the precision with which the plane should reach the waypoint.
- **heading:** This value gives the heading the plane should have when passing over the waypoint, landing or dropping the object (value not used in case of homing)

The configuration options offer quite a large customization, and an interface on the ground station was created so that waypoint edition could be achieved during flight (see section 4.5).

An internal variable of the waypoint manager indicates the id of the active waypoint, the one the MAV has to reach. At the beginning of the navigation process, the function

Wpt_update_status() is called. It checks how far the MAV is from the waypoint, and if it has reached it. Then, it decides if the MAV has to go to next waypoint. To decide when to go to next waypoint if the actual waypoint is homing, the heading that the next waypoint would command is computed. Next waypoint is activated only when the actual heading of the MAV is close to the computed "next" command, otherwise the plane keeps homing (see fig 18). If actual waypoint is not homing, next waypoint is activated as soon as the MAV has reached the waypoint.

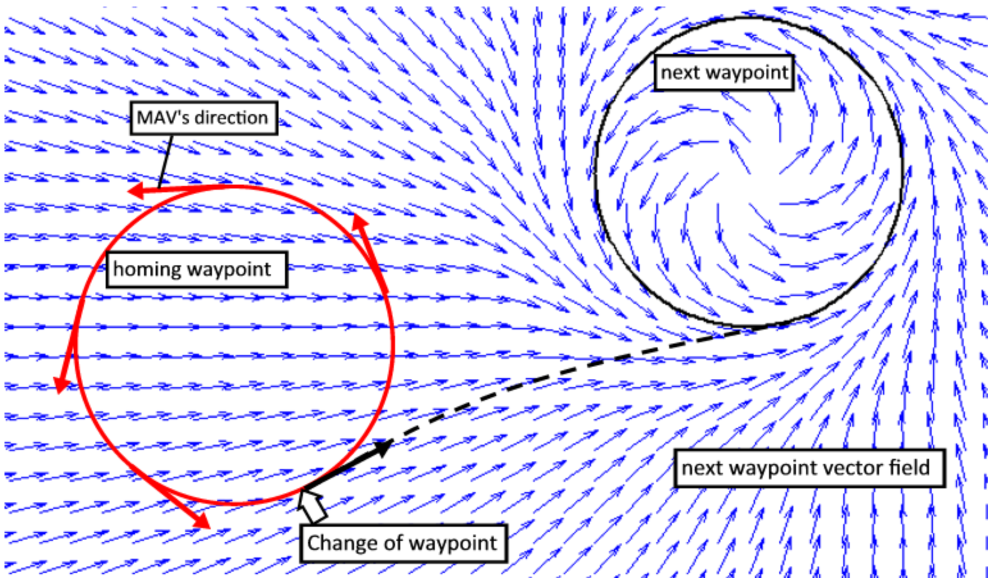


Figure 18: The MAV changes of homing waypoint when next commanded direction (blue vector field) fits the actual MAV direction

4.2 Navigation controller

The navigation controller's task is to set the variable *dpsi_c*, used afterward by the low-level control to achieve the desired turning rate. The computation of *dpsi_c* is realized in *navigation_commands()* and depends on the type of the waypoint the plane is flying to (the active waypoint is given by the waypoint manager):

- **homing**: The command is calculated using the vector field method, described in section 3.4.2, with the a priori term. The gain K of formula (3.8) can be set from the ground station, and is typically 0.75.
- **pass**: The command is calculated so that the plane follows Dubins' path, generated at waypoint activation (see section 4.3).
- **land, drop**: The command is calculated to achieve a line following, with the vector field method described in section 3.5.

An altitude command h_c is also generated, in function of the waypoint type:

- **homing, pass, drop**: (function *Nav_altitude_controller()*) If the waypoint's parameters indicate an altitude to respect at waypoint (*alt_act* is on), the commanded altitude is increased or decreased till the desired altitude is reached. The altitude variation rate can be set from the ground station, and is typically of $2m/s$ for a smooth ramp.
- **land**: (function *Nav_landing_controller()*) To land, a descending altitude is commanded. The altitude is computed in function of the distance to landing point, projected on the line to follow. This permits to set an angle of descent for a linear descent (see fig 19), but other curves could be used (with a stronger slope at the beginning, and a lighter just before landing for example). The typical approach angle is around 12.5°. For a smooth landing, the control is a bit tuned in the last 10 meters: the thrust is cut totally, and the altitude command is not decreased as long as the speed is not lower than a certain threshold, which makes the MAV glide at low speed during the last meters.

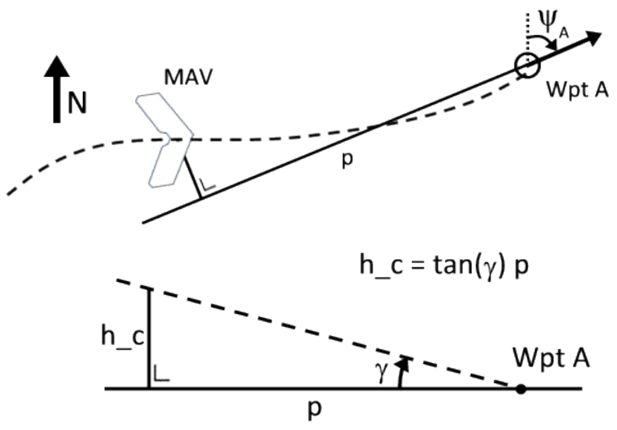


Figure 19: Up: shows the projection p of the distance to waypoint. Down: generated altitude command h_c in function of the projected distance to landing waypoint for a linear descent

4.3 Dubins path planning

Dubins path planning is used when the next waypoint is a "pass" waypoint. To generate Dubins' trajectory described in section 3.6.3, intermediate waypoints are created to achieve the arc-line-arc path. Two homing waypoints are used, with the same transition rule as explained in fig 18. In a sense, navigation is therefore realized thanks to a succession of homing waypoints well positioned.

The function *Dubins_init_path()* in *Dubins.c* creates these two intermediate waypoints. Two cases can occur:

1. The path is initialized when the navigation is launched. In this case, the function creates the path with the starting point being the actual position of the MAV.

2. The path is initialized because a "pass" waypoint was just reached, and next waypoint is activated. In this case, the function creates the path with the starting point being the previous waypoint.

Both situations are shown in fig 20. The function *Dubins_update_status()* then checks if second intermediate waypoint should be activated, with the rule described in fig 18. The "pass" waypoint is achieved when the desired precision of position and heading are accomplished. If these conditions are not respected, the second intermediate homing waypoint stays active and the MAV turns around it till the conditions are respected.

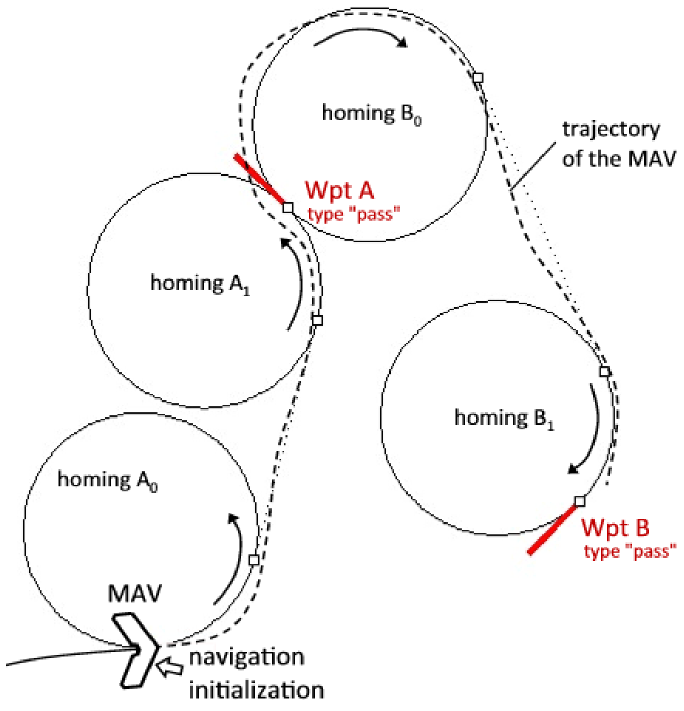


Figure 20: Dubins' path to Waypoint A is achieved thanks to intermediate waypoints A_0 and A_1. The path is generated with the starting point being the position of the MAV at navigation initialization. The path to Waypoint B is generated with the starting point being waypoint A.

4.4 Emergency mode

The emergency mode is activated when the MAV is flown in manual mode, and that the radio signal is lost. This can arrive if the pilot flies out of the radio controller's range, or if there is a problem with the radio controller. When in automatic mode, the emergency mode is not activated, because a route can possibly be programmed out of range, which is not a problem as long as the route is well programmed. The emergency mode, when activated, enables navigation and makes the MAV fly back home (home is by definition the first waypoint of the table), and homing around the home waypoint is realized. For proper work of the emergency mode, a modification concerning the trims management was realized (see annexe 8.2.2).

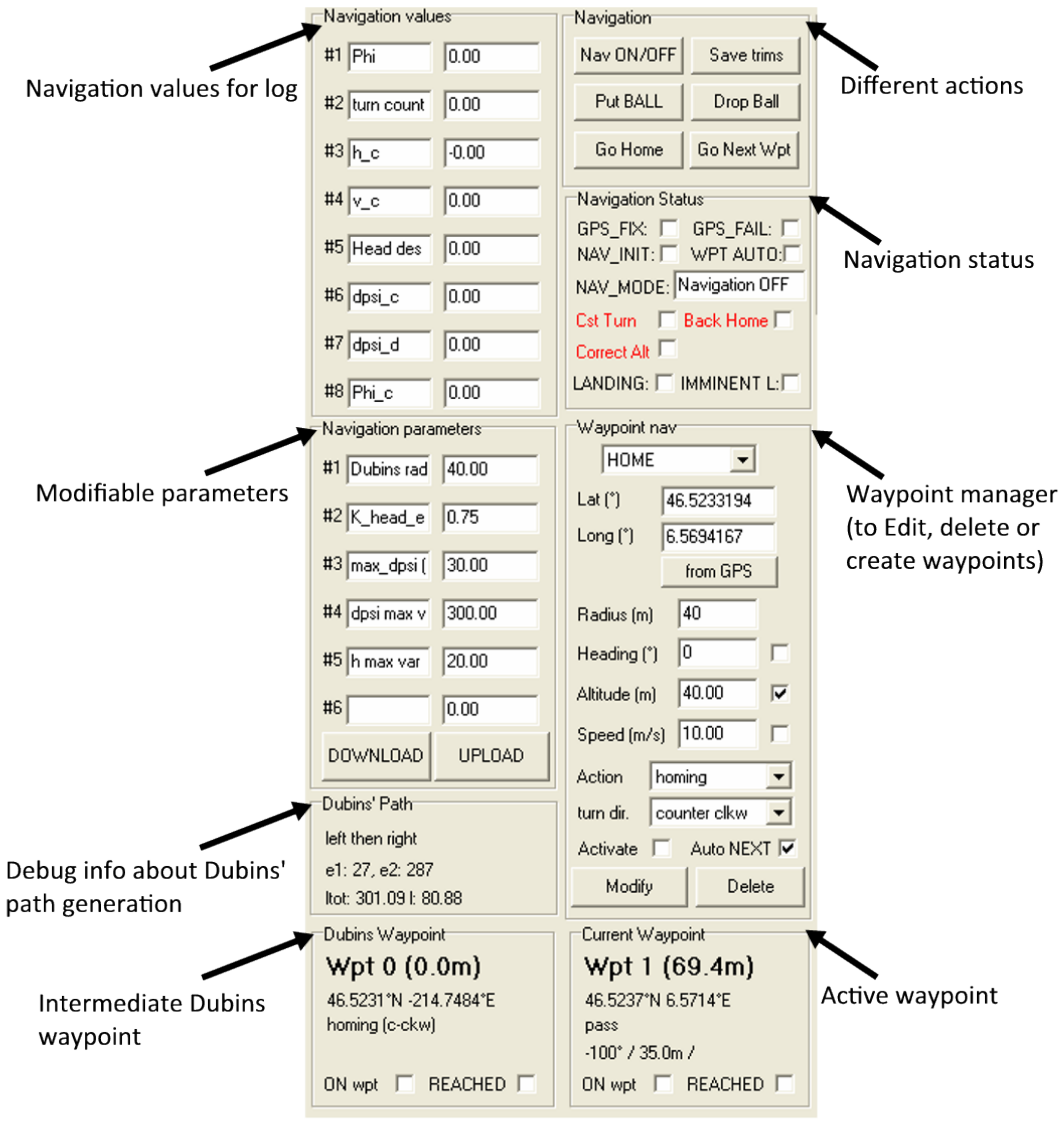
4.5 Monitor add-ons

To monitor the new modules, the initial ground-station software was extended (see fig 21). Almost the whole data transfer happens from the plane to the ground-station, for data logging and debug. Therefore the monitor extensions are mainly for display purposes, and most of the values received by the monitor are saved in the log file. Data is sent to the plane for configuration purposes, when uploading parameters, sending actions or editing waypoints. Waypoint edition is possible thanks to the id present in the structure, and one waypoint can be edited at a time. A drop-down menu allows to choose which waypoint to edit or delete.

When the appropriate option is enabled on the monitor (*Autocapture ON*), the flight controller sends 10 times per second to the ground station the data in different structures :

- **fdata** : (already on original platform) The sensors' values, the RC signals, the commands to the actuators, ...
- **navdata** : Some navigation values (generated commands, etc...), navigation status
- **curwpt** : Parameters of the waypoint the MAV is flying to
- **curpath** : Infos about the active Dubins' path (if next waypoint is of type ”pass”)
- **dubinswpt** : Parameters of the intermediate waypoint of Dubins' path the MAV is flying to

The transmission load could be lowered a bit, since some values do not change during many seconds and do not need to be sent at 10 Hz (current waypoint, path data, etc...), but enough bandwidth (56'000kbps) is provided by the Xbee, and a more powerful software to monitor the MAV (Ishtar) is in development and is probably going to be used in the future. However, some difficulties to upload data to the MAV were observed when the download volume is high, especially when the distance between the ground station and the MAV increases.



Navigation values

#1	Phi	0.00
#2	turn count	0.00
#3	h_c	-0.00
#4	v_c	0.00
#5	Head des	0.00
#6	dpsi_c	0.00
#7	dpsi_d	0.00
#8	Phi_c	0.00

Navigation Status

GPS_FIX: ☐ GPS_FAIL: ☐
NAV_INIT: ☐ WPT AUTO: ☐
NAV_MODE: Navigation OFF
Cst Turn ☐ Back Home ☐
Correct Alt ☐
LANDING: ☐ IMMINENT L: ☐

Navigation parameters

#1	Dubins rad	40.00
#2	K_head_e	0.75
#3	max_dpsi (	30.00
#4	dpsi max v	300.00
#5	h max var	20.00
#6		0.00

Waypoint nav

HOME
Lat (°) 46.5233194
Long (°) 6.5694167
Radius (m) 40
Heading (°) 0 ☐
Altitude (m) 40.00 ☑
Speed (m/s) 10.00 ☐
Action homing
turn dir. counter clkw
Activate ☐ Auto NEXT ☑

Dubins' Path

left then right
e1: 27, e2: 287
ltot: 301.09 l: 80.88

Dubins Waypoint

Wpt 0 (0.0m)
46.5231°N -214.7484°E
homing (c-ckw)
ON wpt ☐ REACHED ☐

Current Waypoint

Wpt 1 (69.4m)
46.5237°N 6.5714°E
pass
-100° / 35.0m /
ON wpt ☐ REACHED ☐

Figure 21: Sections added to the original ground-station software, for navigation monitoring

5 Tests organization

5.1 Tests process

As mentioned in the project description, this work is made of 40% of tests, because practice never works as well as theory and adjustments are always needed to be done. The flight tests are done in the following manner : Once all test parameters are set thanks to the monitor, the plane is flown manually by a pilot, who drives it to a reasonable altitude. Then, the pilot switches on the automatic mode. From there, the pilot does not control the plane anymore and the navigation algorithms take the relay. There is always a safety link, with the possibility for the pilot to take back control of the plane at anytime, if he decides that the plane showed an abnormal behavior. During flight, different navigation parameters or solutions can be changed from the ground-station software (of course the pilot has to be warned when the plane will change its behavior, to avoid surprises). When the tests are over, the pilot takes again control of the plane to land it if no automatic landing was programmed.

A maximum of 1 or 2 different parameters/solutions are usually tested during one test session, and most of the time, improvements cannot be realized immediately because the flight has to be analyzed in details. It is important to evaluate subjectively the attitude of the plane by looking at it, and determine if its behavior is good or a bit unstable, etc... But the only efficient way to validate the control is by analyzing the logs saved during flights. In the following section, the tool developed for the evaluation of the plane's navigation is presented.

5.2 Matlab GUI

Matlab is used to analyze the data saved during the flights. First, the file *import_log.m* is used to load the log file in a Matlab structure. A GUI (*waypoint_eval.m*) was realized so that the data in different places in the log can easily be accessed, thanks to the scrollbars that allow to choose the part of the data that has to be plotted. The different informations available in the GUI are presented in fig 22. A main advantage when evaluating the navigation algorithms (and therefore evaluating the trajectory), is the possibility to associate the commands (plotted on the right) with the effect on the trajectory (plotted on the left), or the inverse (effect of plane position on the commands).

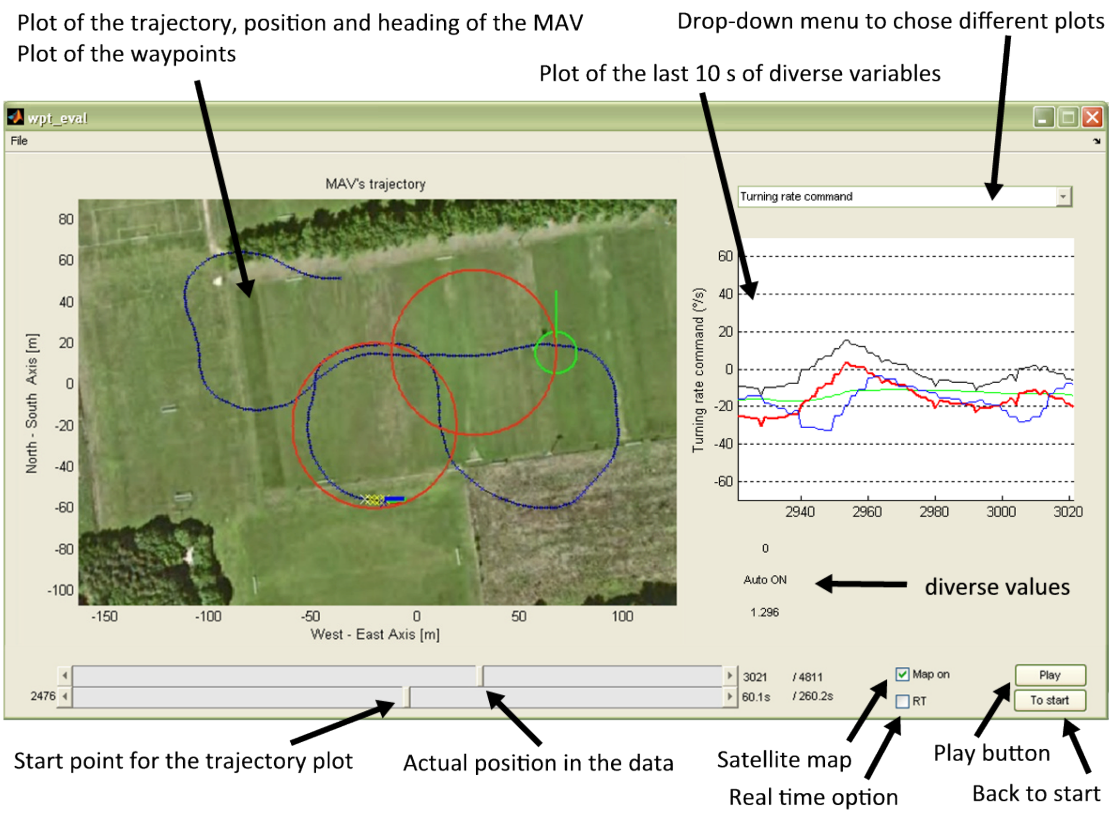


Figure 22: Matlab GUI for flight data analysis, and navigation algorithms evaluation. The plot of the trajectory with the waypoints and the plot of the control commands (among other values) can be played in real time or faster. Thanks to the scroll bar, it is possible to go backward or forward to any position in the log, or make a pause.

6 Results

First, characterization of the low-level control used by navigation is realized. Then, the results of the basic behavior tests are presented, and then the full waypoint navigation. Finally, an attempt to improve the navigation precision is presented.

6.1 Turning rate characterization

Navigation entirely relies on the low-level turning rate control to achieve the desired trajectory. It is therefore important to understand its mechanisms an its limits. It is also important to be sure that the commanded turning rate is really accomplished. No sensor available on the platform can directly return the turning rate of the plane, however, it can be inferred from the trajectory given by the GPS by looking at the radius of the circle the plane is making when flying at a constant turning rate. To go further, the heading of the plane is provided by the GPS as well, therefore it is possible to infer the turning rate by derivating the heading. Since the GPS update is of course discrete (4Hz), the derivate is done by subtracting all values by their preceding values. The derivative is very noisy, and a moving average on 4 values is realized to smooth the curve. This method to measure the turning rate showed good results, and can be used to characterize the turning rate control.

Fig 23 shows the measured turning rate and the commanded turning rate. One can see that the commanded turning rate is quite well achieved, but with a certain delay. This delay is around 1-1.3s, and is probably mainly due to the plane's dynamic : from the time the plane's ailerons are activated for a turn, the MAV takes one second to get to the commanded roll angle and turn because of its inertia. The observed delay may also be, at lower scale, due to the small GPS' delay.

In section 6.5, solutions to take this delay into account are proposed, but this "turning rate response" is good for navigation, since the commanded turning rate is well achieved.

6.2 Homing

6.2.1 Choice of the homing radius

Some open loop turning rate control tests were run (that is to say, without navigation, only with automatic altitude and speed control, and with manual turning rate command) with constant turning rates, and the MAV was capable to achieve circles of quite small radiuses (around $10-15m$). Of course, because no closed-loop corrections on the position are done, the MAV rapidly goes away from its original position and draws spirals, because of perturbations and wind. In closed loop navigation, such small radiuses are not possible to be achieved. This is because the smaller the radius is (and the faster the turning rate is), the faster the corrections have to be done, because the MAV changes

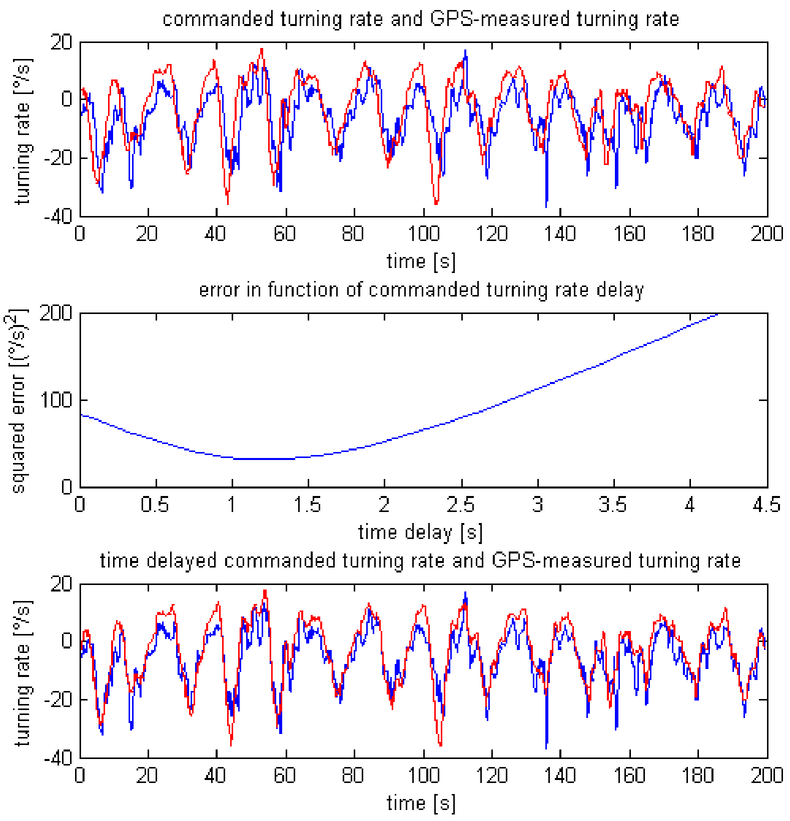
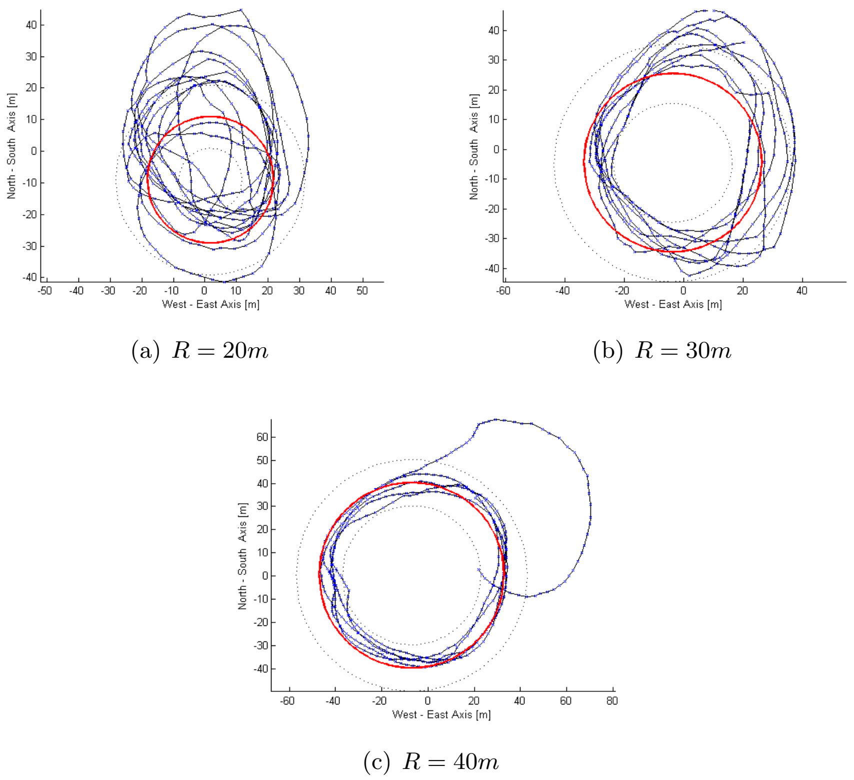


Figure 23: Data of a 200s flight with no wind. The first plot shows a delay between the commanded turning rate (red) and the GPS-measured turning rate (blue). The second plot is used to find the delay that exists between both curves, by finding the minimum error. The third plot shows the turning rate curves that fit together when the command is delayed.

its position relatively to the waypoint quickly. Hence, limiting factors are here the GPS' position update rate, and the delay existing between the command and the application of the turning rate. In addition to that, the smaller the radius is, the more critical is the position precision, because the commands vary a lot more in the space for small radiuses. The limiting factor is here the GPS's precision.

As we can see, the use of GPS is probably the most limiting factor for realizing small circles, or any rapidly varying trajectory in general. It is difficult to evaluate analytically the minimum radius that can be achieved, and homing was tested for different radiuses, as shown in fig 24. As we can see, the implemented homing solution gives a good result for a radius of 40m, and the trajectory has very well repeated by the MAV. More important errors can be observed with a 30m radius, and the MAV sometimes goes really far from the waypoint with the 20m radius. A different control would maybe achieve better results for small circles, by relying less on the direct GPS data, that can induce strong changes in the command because of its update rate an imprecisions. However, there is not really a need for the MAV to be able to accomplish small radiuses, and 40m is chosen to be the standard homing radius. In Dubins' waypoint navigation, a small radius implies a smaller path, but the difference is little, and precision is more important.

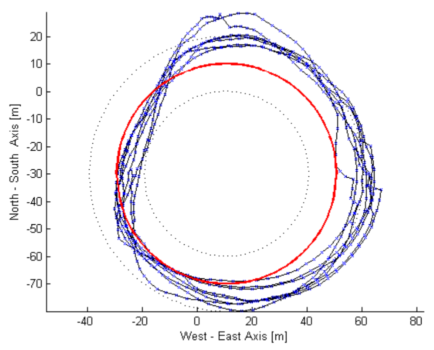


(a) $R = 20m$ (b) $R = 30m$ (c) $R = 40m$

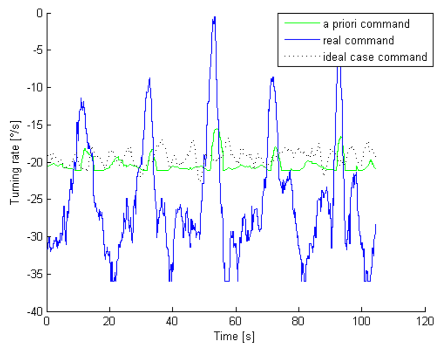
Figure 24: Results of homing, tested with different radiuses R, with almost no wind. The dashed lines are the $\pm 10m$ circles.

6.2.2 Influence of wind and heading gain on homing

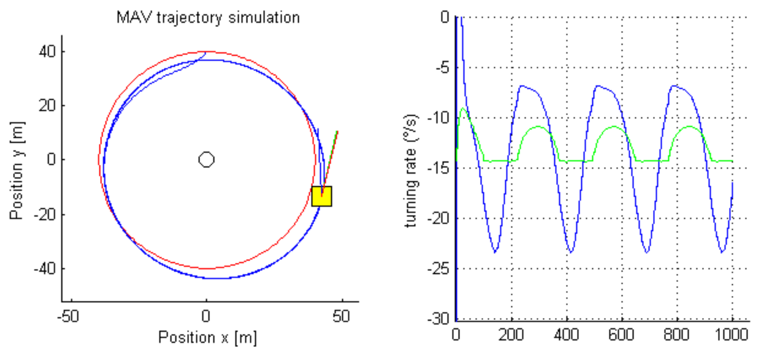
In fig 25, the actions of the different terms of equation (3.8) can be observed. The a priori command ($\dot{\Psi}_d$) gives a relatively constant term, to which the correction term ($K \cdot (\Psi_d - \Psi)$) is added. As we can see, the commanded turning rate is very different from the a priori command, and the correction term is quite significant. It is essential here to correct the disturbances due to the wind, and it is interesting to see that the simulated turning rate command shows this same aspect.



(a) MAV's trajectory, for a 40m radius in light wind conditions (3-5 m/s)



(b) Commanded turning rate (in blue), and a priori term (green)



(c) Left: simulated homing, with 3m/s wind in the East direction. Right: corresponding turning rate command (blue) and a priori term (green)

Figure 25: Real and simulated case, for homing in light wind conditions. Similar turning rate commands are generated (heading correction gain $K = 0.75$).

To illustrate the effect of mis tuning the gain K, figure 26 shows the result of a test in windy conditions with a gain set too high ($K = 1$). The disturbances due to the wind are causing strong deviations of trajectory and the corrections, amplified by the gain K, are to high and oscillations are created. With no wind, such a gain induces almost no oscillations, but a gain that can be used in most situations was chosen. Finally, a gain of $K = 0.75$ was found after many tests to fit most situations (windy or not). The oscillations are very limited, and the heading corrections are strong enough to achieve a quite precise homing.

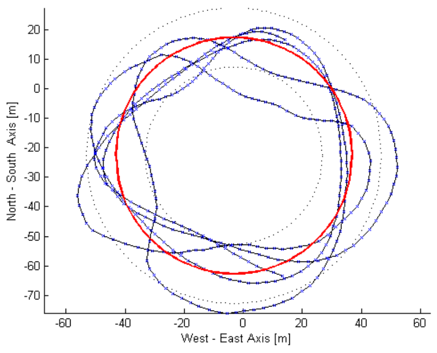


Figure 26: Homing with a 40m radius, windy conditions ($> 5m/s$). Oscillations are present in the trajectory, because of a too high gain K equal to 1.

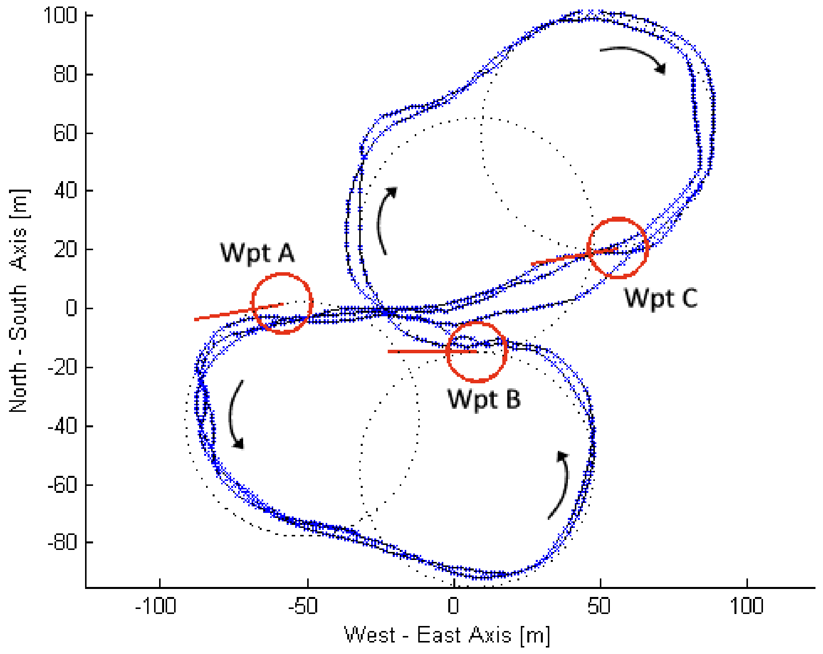
6.3 Waypoint navigation

From the moment that homing is well tuned, Dubins' waypoint navigation is achieved relatively precisely, since it only relies on homing waypoints (see section 4.3). Figure 27 shows the result of such a waypoint navigation, with all intermediate homing waypoints set to make the MAV follow Dubins'path. The precision is always better than $10m$ when passing over the waypoint, and the trajectory is followed with a good repeatability. If a waypoint had been missed because of disturbances or wind, the MAV would have realized more turns on the last homing circle till the waypoint would be reached. To avoid a blocking situation where the waypoint is never reached (because of strong wind for example), one could imagine a precision tolerance increasing with the number of turns.

6.4 Landing

For landing, line following is used to align the MAV and then a gradual descent is commanded (see sec 4.2). One can see in fig 28 that the line following gives good results. The altitude control would make the plane land close to the waypoint if the landing had not been smoothed at the end, to have a low-speed landing. The plane lands always 10 to 20m after the waypoint, but since this is a repetitive behavior, navigation can easily be adapted for a quite precise landing.

Since line following showed to be relatively precise, it could also be used for the "pass" waypoints : first Dubins' path can be planned to make the MAV arrive 50m before the waypoint, and then a line following can be realized during the last 50m for example. If a good precision is needed when passing over the waypoint, this could be a solution, because otherwise the waypoints are always reached when the plane is turning, which is maybe less precise.



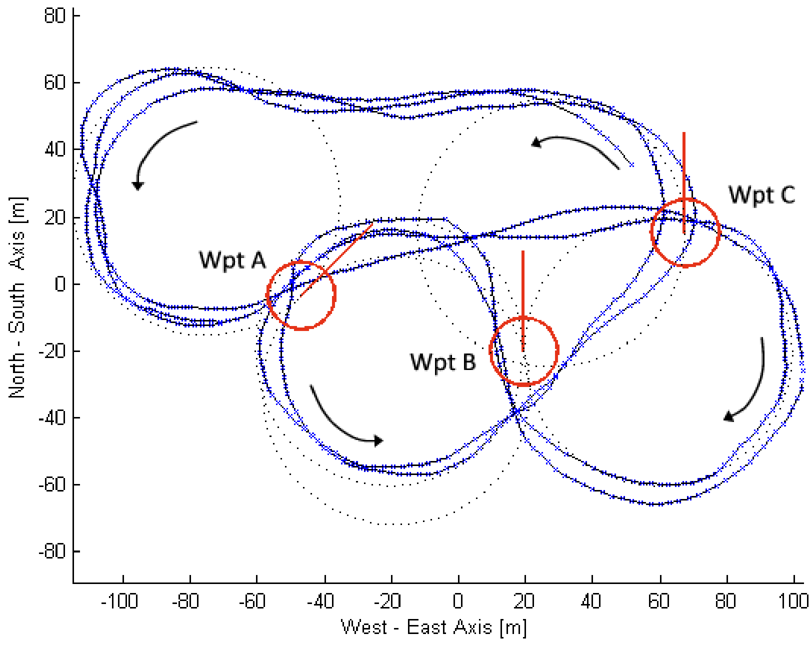


Figure 27: Two examples of tests with a sequence of 3 ”pass” waypoints, with the intermediate Dubins waypoints (dashed). The circles around ”the pass” waypoints represent a $10m$ precision. No wind during test.

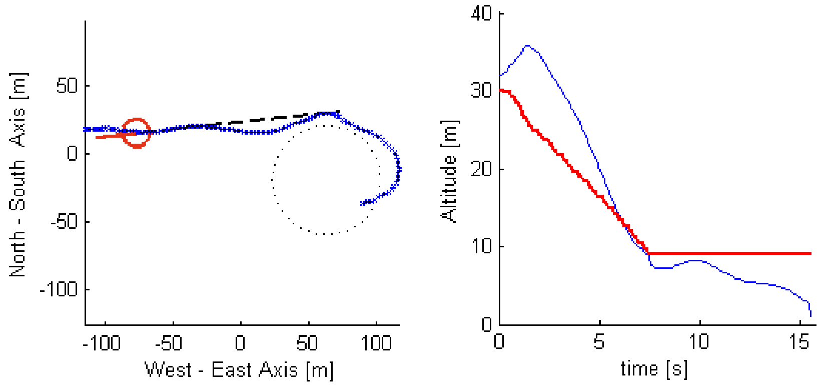


Figure 28: Test result of a landing phase. Left: landing trajectory (line following during 150m). Right: commanded altitude (red) and measured altitude (blue). At the end of the landing phase, the command is not decreased any more, to let the plane slow down and realize a smooth landing.

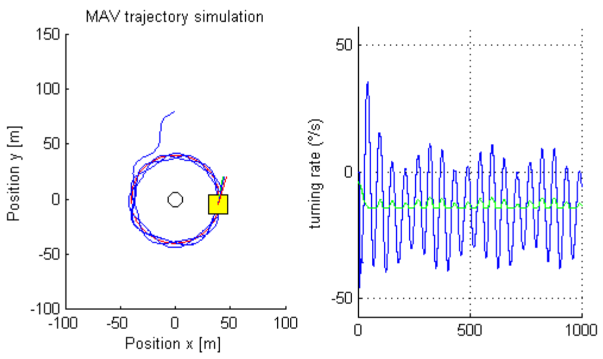


Figure 29: Result of a simulation with a 1s delay before the turning rate command is effective (K=1, wind = 1m/s)

6.5 Position prediction

The delay demonstrated in sec 6.1, between the commanded and the measured turning rate, induces a delay in the MAV's reactions. It is suspected to cause some oscillations, that are also present in simulations when delay is introduced (see fig 29).

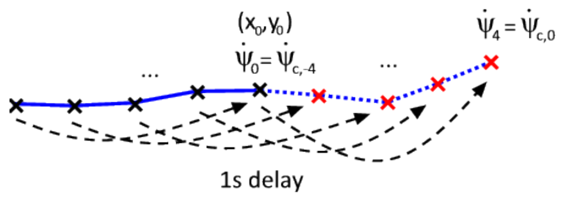
A solution to cancel this delay is proposed. Since it was calculated to be equal to about 1s, the idea is to estimate the MAV's position 1 second later, and to apply the turning rate command in function of the estimated position to cancel the delay. One second represents four GPS position updates, and to estimate the position on second later, the last four turning rate values are used to estimate the four following positions (see fig 30).

The time is discretized with a period of 0.25s, hence the 1s delay between the command and the effective turning rate can be written as :

$$\dot{\Psi}_k = \dot{\Psi}_{c,k-4} \tag{6.1}$$

Let x_0, y_0 and Ψ_0 be the actual GPS position and heading, and x_k, y_k, Ψ_k the k^{th} position estimations. Every increment, for $0 < k \leqslant 4$ tries to estimate the next GPS position 0.25s later.

$$x_k = x_{k-1} + 0.25 \cdot V_0 \cdot \sin(\Psi_{k-1}) \quad (6.2)$$
$$y_k = x_{k-1} + 0.25 \cdot V_0 \cdot \cos(\Psi_{k-1}) \quad (6.3)$$
$$\Psi_k = \Psi_{k-1} + 0.25 \cdot \dot{\Psi}_{k-1} = \Psi_{k-1} + 0.25 \cdot \dot{\Psi}_{c,k-5} \quad (6.4)$$

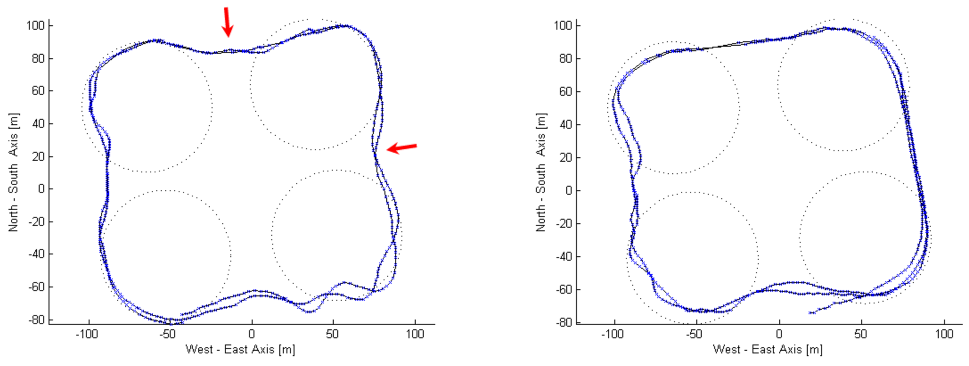


(x_0, y_0)
$\dot{\Psi}_0 = \dot{\Psi}_{c,-4}$
$\dot{\Psi}_4 = \dot{\Psi}_{c,0}$
...
...
1s delay

Figure 30: Estimation of the turning rate, heading and position of next four positions,

This estimation showed good results for tests without wind, and it was used for the control, to set one second in advance the turning rate. The effect on the trajectory (see fig 31) are interesting, even if it does not change radically the trajectory. The MAV anticipated well two turns with the position estimation, and the corrections of trajectory showed by the red arrows in fig 31(a) are not present in fig 31(b). This is because the MAV anticipated that it would have to go straight, and reduced its turning rate in advance. But both other turns were not as well anticipated, and corrections are needed to be done in both cases.

This predictive control needs probably some more tuning, but it looks like it can improve a bit the trajectory and remove partially the effect of the delay.



(a) no delay anticipation

(b) with 1s delay anticipation

Figure 31: Comparison of trajectories for a 4 homing waypoints route, once without delay anticipation, once with.

7 Conclusion

A complete and fully customizable waypoint navigation was developed in this project, and showed relatively precise results for trajectories implying limited turning rates. Indeed, as seen in section 6.2.1, the GPS limitations do not permit too high turning rates in navigation. A solution to take into account the delay existing between the command and the plane's behavior is proposed, and could be a good way to deal with the MAV's inertia.

Landing at a pre-defined position was also demonstrated, and therefore the MAV can be programmed to be fully autonomous from the initialization of navigation till the end of the flight. In addition to that, an emergency mode uses navigation to make the MAV go back home if the radio controller's signal is lost.

A future work could study in more details the effect of wind, and an estimation of the wind's amplitude and direction could be imagined. The a priori control could then take the wind into account to achieve a more precise trajectory in windy conditions. Improvements could also be done to increase the maximum turning rate acceptable in navigation, by trying to improve the GPS position precision thanks to interpolation or thanks to fusion with other sensors.

This project was a good example of the challenges one has to face when programming an autonomous robot: First, debugging is difficult, because only limited information is available to help to solve problems occurring in a microcontroller. This is why the navigation algorithms were implemented first in Matlab, and then transferred on the flight controller. It is also important to have an efficient way to understand what happens during the experiences, this is why a Matlab GUI was realized for the logs analysis. Secondly, the tests can never be realized in the same conditions (especially with outdoor tests), and different problems can happen almost randomly. For example, it was discovered after many weeks that in function of the sun's exposition of the MAV, the pressure sensor is altered, which resulted in very strange flight behaviors.

Finally, the project goal is achieved, and one can program any desired sequence of waypoints, so that the MAV autonomously flies to the desired places and lands at the desired point.

Lausanne, 6th of June 2008

Adrien Briod

References

[1] Derek R. Nelson, D. Blake Barber, Timothy W. McLain, Randal W. Beard, *Vector Field Path Following for Small Unmanned Air Vehicles*

[2] Sanjay P. Bhat, Pradeep Kumar, *A Feedback Guidance Strategy for an Autonomous Mini-Air-Vehicle*

[3] R. Zhu, D. Sun, Z. Zhou, *Integrated design of trajectory planning and control for micro air vehicles*

[4] R. L. McNeely, *Trajectory planning for Micro Air Vehicles in the presence of wind*

[5] L. E. Dubins, *On curves of minimal length with a constraint on average curva-ture and with prescribed initial and terminal positions and tangents*, in *AmericanJournal of Mathematics*, vol. 79, 1954, pp 497-516

[6] Sabine Hauert and Severin Leven, *Swarming MAVs for communication relay*, http://lis.epfl.ch/research/projects/SwarmingMAVs/, EPFL-STI-I2S-LIS, Switzer-land, 2008

8 Annexes

8.1 Trajectory control calculations

8.1.1 Tangent direction calculation for circle following

The following equations are used to calculate Ψ_d so that the desired direction is along the tangent of the homing circle. See figure 6 for symbols' signification :

$$\begin{aligned} D &= \sqrt{Dx^2 + Dy^2} && (8.1) \\ \alpha &= \operatorname{atan2}(Dx, Dy) && (8.2) \\ \beta &= \operatorname{asin}(\frac{R}{D}) && (8.3) \\ \Psi_d &= \begin{cases} \alpha - \beta & \text{for clockwise turn} \\ \alpha + \beta & \text{for counter-clockwise turn} \end{cases} && (8.4) \end{aligned}$$

8.1.2 Vector field calculation for circle following

The following equations, found in [1], are used to calculate Ψ_d to generate the vector field shown in fig 7, and the a priori desired turning rate $|\dot{\Psi}_d|$ plotted in fig 8 :

For $D > 2 \cdot R$:

$$\begin{aligned} \Psi_d &= \alpha + t \cdot \operatorname{atan2}(\frac{R}{D}) && (8.5) \\ |\dot{\Psi}_d| &= V \cdot R \cdot \frac{R^2 + 2 \cdot D^2}{D \cdot (R^2 + D^2)^{3/2}} && (8.6) \end{aligned}$$

For $R < D < 2 \cdot R$:

$$\begin{aligned} \Psi_d &= \alpha + t \cdot (\frac{\pi}{6} + \frac{\pi}{3} \cdot \overline{r}) && (8.7) \\ |\dot{\Psi}_d| &= \frac{V}{D} \cdot sin(\frac{\pi}{6} + \frac{\pi}{3} \cdot \overline{r}) + \overline{r} \cdot (\frac{\frac{\pi}{3} \cdot V \cdot cos(\frac{\pi}{6} + \frac{\pi}{3} \cdot \overline{r})}{R}) && (8.8) \end{aligned}$$

With :

$$\overline{r} = (1 - \frac{D - R}{R}) \qquad (8.9)$$

For $D < R$:

$$\begin{aligned} \Psi_d &= \alpha - t \cdot (\frac{\pi}{6} + \pi + \frac{\pi}{3} \cdot \frac{D}{R}) && (8.10) \\ |\dot{\Psi}_d| &= \frac{V}{D} \cdot \sin(\frac{\pi}{6} + \frac{\pi}{3} \cdot \frac{D}{R}) - \frac{D}{R} \cdot \frac{\pi}{3} \cdot cos(\frac{\pi}{6} + \frac{\pi}{3} \cdot \frac{D}{R}) && (8.11) \end{aligned}$$

8.1.3 Vector field calculation for line following

The following equations, found in [1], refer to fig 32 and calculate the vector field for a line following shown in fig 9. The line is fully described by a point (Waypoint A) and an angle (heading of Waypoint Ψ_A):

$$\epsilon = \cos\Psi_A \cdot Dx - \sin\Psi_A \cdot Dy \tag{8.12}$$

$$\Psi_d = \begin{cases} \Psi_A + \text{sign}(\epsilon)\Psi_e & \text{for } |\epsilon| > \tau \\ \Psi_A + \frac{\epsilon}{\tau}\Psi_e & \text{for } |\epsilon| > \tau \end{cases} \tag{8.13}$$

Ψ_e and τ are parameters. Ψ_e is the entry heading angle and τ is the transition region boundary distance. $\Psi_e = \pi/4$ and $\tau = 20m$ were chosen for implementation.

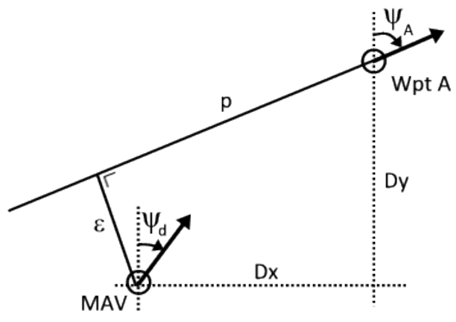


Figure 32: Ψ_d is the desired heading for the MAV, calculated at every point of the space to form a vector field for line following. The line is described by Waypoint A and heading Ψ_A

8.1.4 Coordinate change for the waypoint navigation problem

The general case where two waypoints having fixed orientations are disposed in the space (fig 10(a)) can be transformed with different coordinates, so that all situations can be described by 3 parameters (fig 10(b)). The coordinate change is described by the following equations :

$$L = \sqrt{Dx^2 + Dy^2} \tag{8.14}$$

$$\alpha = \text{atan2}(Dx, Dy) \tag{8.15}$$

$$\Psi_0 = \alpha - \Psi_A \tag{8.16}$$

$$\Psi_F = \alpha - \Psi_B \tag{8.17}$$

8.1.5 Shortpath and Lowturning waypoint navigation

This method, to create a trajectory linking two waypoints with heading constraints, looks for the circle tangent to both lines formed by both waypoints, as shown in fig 33.

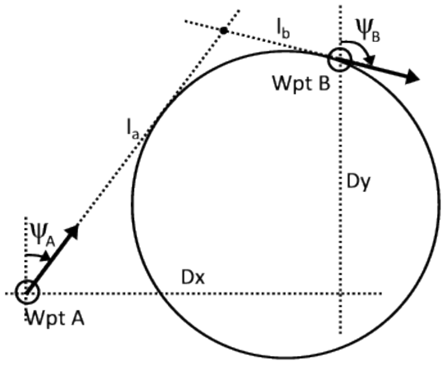


Figure 33: Circle tangent to both lines and having one of the waypoints on it. An arc-line or line-arc trajectory is then created in function of the waypoints disposition.

The following equations, partially found in [3], are used to determine if the trajectory is an arc-line or a line-arc configuration, and the radius of the arc is calculated.

$$l_a = \frac{Dx\cos\Psi_B - Dy\sin\Psi_B}{\sin(\Psi_A - \Psi_B)} \tag{8.18}$$

$$l_b = \frac{Dy\sin\Psi_A - Dx\cos\Psi_A}{\sin(\Psi_A - \Psi_B)} \tag{8.19}$$

if $l_a > l_b$, it's a line-arc configuration with :

$$R = \frac{Dy\sin\Psi_A - Dx\cos\Psi_A}{1-\cos(\Psi_A - \Psi_B)} \tag{8.20}$$

if $l_a < l_b$, it's an arc-line configuration with :

$$R = \frac{Dx\cos\Psi_B - Dy\sin\Psi_B}{1-\cos(\Psi_A - \Psi_B)} \tag{8.21}$$

8.1.6 Dubins' optimal path

To find the optimal path of Dubins' solution (see §3.6.3), the centers of Z_0, Y_0, Z_F and Y_F are first computed. Then, the length of the four possible paths are calculated. To find the length of the path, 3 values are calculated : the angle ϕ_0 during which the MAV flies along circle 0, the length l of the tangent line to both circles, and the angle ϕ_F during which the MAV flies along circle F. Angles ϕ_0 and ϕ_F are in the range $[0, 2\pi[$. The total length is then easily calculated thanks to this formula :

$$l_{tot} = \phi_0 R + l + \phi_F R \tag{8.22}$$

The following formulas refer to fig 34, and calculate the length of a Y_0LZ_F path, knowing the distance C between both circles and the angle β.

$$\alpha = \arccos(\frac{2R}{C}) \tag{8.23}$$

$$l = \sqrt{C^2 - 4R^2} \tag{8.24}$$

$$\phi_0 = \pi - \beta - \alpha - \Psi_0; \tag{8.25}$$

$$\phi_F = \pi - \beta - \alpha - \Psi_F; \tag{8.26}$$

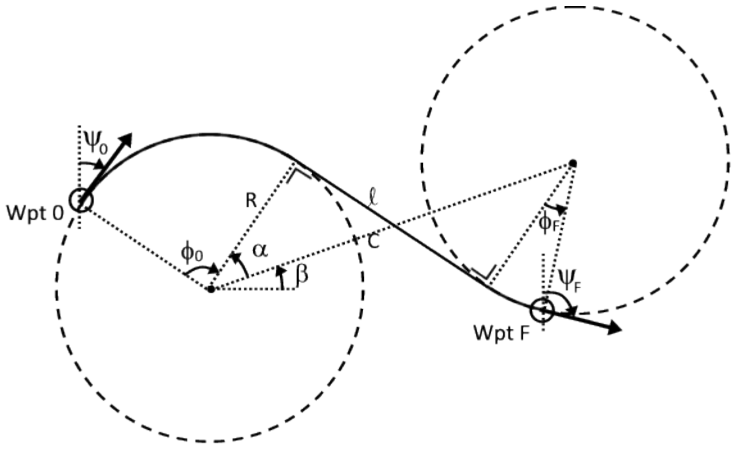


Figure 34: Values needed to calculate the length of a Y_0LZ_F path (clockwise, then counter-clockwise)

The other cases are similar and the formulas are the following : Z_0LY_F (counter-clockwise, then clockwise) :

$$\alpha = \arccos(\frac{2R}{C}) \tag{8.27}$$
$$l = \sqrt{C^2 - 4R^2} \tag{8.28}$$
$$\phi_0 = \beta - \alpha + \Psi_0 \tag{8.29}$$
$$\phi_F = \beta - \alpha + \Psi_F \tag{8.30}$$

Z_0LZ_F (twice counter-clockwise) :

$$l = C \tag{8.31}$$
$$\phi_0 = \beta - \frac{\pi}{2} + \Psi_0 \tag{8.32}$$
$$\phi_F = \frac{\pi}{2} - \Psi_F - \beta \tag{8.33}$$

Y_0LY_F (twice clockwise) :

$$l = C \tag{8.34}$$
$$\phi_0 = -\beta + \frac{\pi}{2} - \Psi_0 \tag{8.35}$$
$$\phi_F = -\frac{\pi}{2} + \Psi_F + \beta \tag{8.36}$$

Note : the Y_0LZ_F and Z_0LY_F paths exist only if $C \geqslant 2R$.

8.2 Embedded code modifications

Diverse modifications not related to navigation were realized in the flight controller's code for a proper operation of the new modules.

8.2.1 Data transmission

Data transmission with the ground-station software had to be made more modular, since the sending of only one specified data structure was originally possible, and message reception was not very modular. Data reception happens in *usart1.c*, and data transmission in *dma.c*.

For data reception, each byte coming from the ground station is stored in a buffer, and the function *USART1_msg_reception()* checks when one complete message has arrived (header + id + length + content + checksum). The received messages can be addressed to diverse modules, and a function was created in each module that can potentially receive a message. These functions are : *FC_msg_in()* in *flightcontrol.c* to deal with the configuration messages of the flight controller (flight parameters), *Nav_msg_in()* in *navigation.c* for navigation configuration and *Wpt_msg_in()* in *waypoint.c* for waypoint configuration.

For data transmission, DMA is used, so that the sending of data is non-blocking. One just has to write the data to send in the DMA RAM and configure the right length. After many trials, it was discovered that problems were *sometimes* happening when more than 128 bytes are sent in a row. Indeed, once over 20 to 50 times, checksum errors are repeatedly occurring at reception on ground station. To avoid this, a ring buffer is used to store everything that has to be sent, and then the DMA RAM is filled with 100 bytes from this buffer every 25ms (sending these 100 bytes takes about 15ms). The data to send is therefore cut in 100 bytes packets, which does not produce the checksum errors observed earlier (But some transmission errors are always present when the Xbees are far from each other during flights for example). The function *dma_send_structure()* copies in the transmission buffer any sequence of bytes that have to be sent to the ground station, and adds automatically the header and the checksum. This function allows to send easily any data structure of any length, and is used to send the structures presented in sec 4.5.

8.2.2 Trims management

When the plane is flying straight with no altitude change, the values of the motor and servo commands are equal to the *trims*. The MAV's low-level control returns correction values for the motor and servos commands, and these values are added to the trims to give the final commands. The problem here is to know when the flight controller has to save these trims in its memory. Originally, they were saved at the moment when the pilot switches on the automatic mode. This caused problems because the pilot sometimes is still actuating a command when it switches on the automatic mode, which induces then a false offset when the commands to the servos are computed by the flight controller. This even caused once the MAV to crash, and a more reliable solution had to be found.

The new idea is to save the trims at the start of the flight, at the same time as when the sensor bias are done. A quick manual flight before saving the trims can be achieved to ensure that good values for the trims will be saved. Saving the trims at the beginning of the flight is also obligatory for the emergency mode. Indeed, when the radio signal is lost by the plane, it is important that the trims were saved *before* the RC signal loose. The trims are saved by calling the function *RC_save_fail_safe()*.