Project Work
Cyber-Physical Systems

Autonomous Aerobatics on Commanded Paths

written by

Youlin Gao
Anthony Blanc
Andreas Bruckmeier

submitted at the
Institute for Real-Time Computer Systems
Technische Universität München,
to
Prof. Marco Caccamo

Date: 7th March 2016

Contents

List of Figures

1. Overview

As a part of the Cyber-Physical Systems course in WS15/16, a controller structure for an aircraft in a simulation environment was realised, performing autonomous aerobatics on commanded paths. In this report, the controller structure is presented in Chapter 2, and performance results are shown in Chapter 3. Before this, an overview of the working principle and the features of the controller are given, and the underlying work is clarified.

1.1. Working Principle and Features

Control target of the presented system is to follow a path as a function of time in 3-dimensional space. The aircraft's aim is to fly along this commanded path by determining the deviation from the given trajectory and thereout calculating acceleration commands which are the inputs of the aircraft's actuator controllers.

Figure 1 depicts the complete control structure of the aerobatic controller. With the position data from the desired trajectory and with actual position signals of the aircraft, the desired acceleration to follow the path is computed. After considering the gravity, the desired acceleration is handed to the control structure where the acceleration in body frame is used as the control variable for the aileron-, elevator-, and rudder-controllers. Also, with the position data, the delay of the aircraft from the desired position is determined, and by considering a *look ahead distance* which forces the aircraft to lag behind the path by a specific distance, the throttle is controlled with this information.

The resulting steering signals, computed by the control structure, are handed to the simulation environment which simulates the physical reaction of the aircraft and then hands back measurement results of its virtual sensors. These signals are processed and provided to the control structure. A feature of the `Signal Processing` is the estimation of the aircraft's position with only accelerometers and gyrometers when there is no available GPS-signal as it is in inverted flight attitude.

1.2. Underlying Work

The control structure, implemented and presented in this project work, is based on the work of Sanghyuk Park[**?**] and modified by establishing the desired path depending on time plus determining the path delay in order to control the throttle.

The estimation of the aircraft's position signal is implemented after a method which is, at the time of handing in this report, not yet published. We got the access ahead of the release by our tutor [**?**].

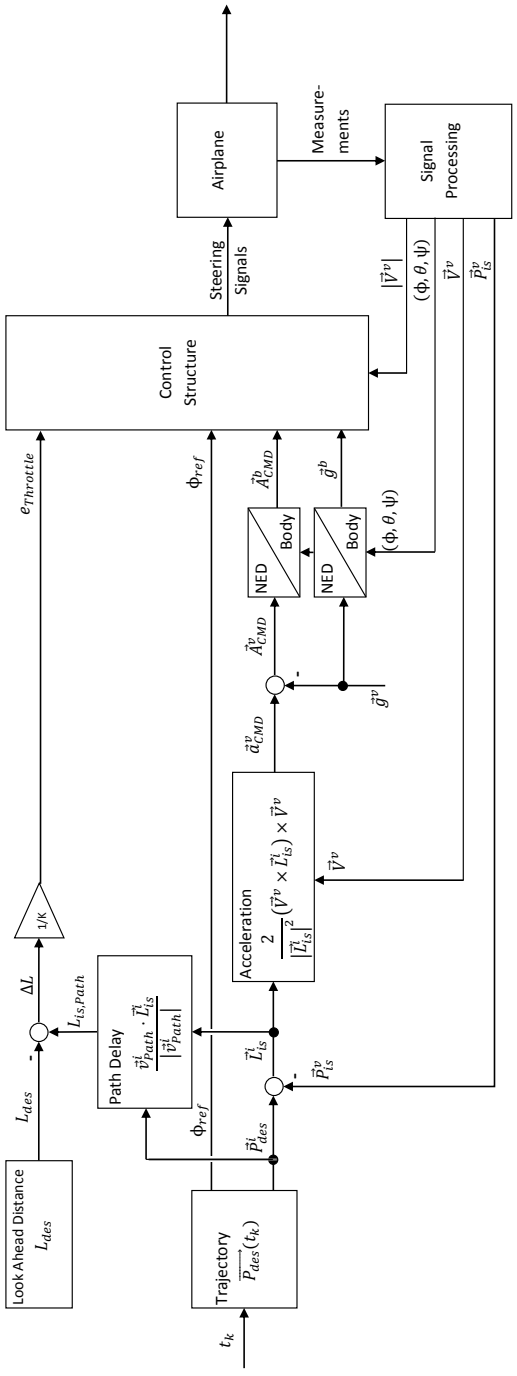

Figure 1: Overview of the control structure of the aerobatic controller.

2. Details

In the following, each part of the control structure (compare Figure 1) is presented in detail.

2.1. Interface with the Simulation Environment

The connection of our program with the simulation environment takes place at the beginning of each program run, when sensor data is handed over and at the end of each run, when commands for the actuators are given to the aircraft.

In the code, the function calculateStateVariables reads the sensor data of the aircraft to our part of the software. At the end of each program run, the four steering signals for the actuators throttle, aileron, rudder, and elevator are given to the simulation environment in the function generateOutSignals.

2.2. User Interface

We have implemented an user interface which enables a communication with the aircraft controller via an USB connection with the controller board. We used the program `hterm` for communicating with the controller over a COM-Port.

The interface enables the user to modify controller parameters and flight special pre-established paths at run time. Sending the lette `'h'` print help instruction and explaination how to use the interface.

All this path have been saved and are managed in a state machine in function trajectory.cpp. Every state correspond to a different path. Here are this list of the possible paths:

- 0. Take off
- 1. Circle
- 2. Looping
- 3. Go streight / Glide
- 4. Roll
- 5. Glide back
- 6. Half circle
- 7. Climb up

- 8. Snake

We can sum up this state in the fallowing diagram:

As you can see on the diagram, some states are unstable are limited during the time. The arrow indicates the direct next state. For more detail about the path, report direclty to the concerning section. To go from one state to an other, the user has to tipp the fallowing command:

- ft - Take off
- fc - Circle
- fl - Looping
- fg - Go streight / Glide
- fr - Roll
- fb - Glide back
- fh - Half circle
- fc - Climb up
- fs - Snake

Note: 'f' is for flying and all the second letter are correspond to the state name.

The API allowed us to edit some variables. It includes only the PID and the lookahead distance. Because, we are dealing with numbers, the formalism is very strict.

- et-VAL$_P$$-$$VAL_I$$-$$VAL_D$: $editthrottlePIDvalueea-VAL_P-VAL_I-VAL_D$: $editaileronPIDvalue$

- er-VAL$_P$$-$$VAL_I$$-$$VAL_D$: $editrudderPIDvalueee-VAL_P-VAL_I-VAL_D$: $editelevatorPIDvalue$

- l-VAL$_L$$ookahead$: $editlookaheaddistance$
  with

  - VAL$_P$$souldberemplacebythevalueofthePROPORTIONALgainVAL_I souldberemplacebythevalueofthe$

- VAL$_D$$souldberemplacebythevalueofthe DERIVATEgainVAL_L ookaheadsouldberemplacebythevalueofthewi$

Note: In the code, it is possible to change the separtation charactere. We have chosen '-' but it might be modify by changing the defined value separation$_c$$harin$API_perso.h.

Here are some example to understand how it works:

- et-1.56-2.55-0 : edit throttle PID with the proportional gain equal to 1.56, the integrale gain equal to 1.56 and the derivate gain equal to 0
- ea-7.01-0.55-0.22 : edit aileron PID with the proportional gain equal to 7.01, the integrale gain equal to 0.55 and the derivate gain equal to 0.22

In the code, the function API_interpretate_chain cares for the interpretation of the user commands and the instruction transfer.

It is also to mention, that we introduced two time signals, the *hardware time* and the *relative time*. The hardware time is the time since the start-up of the aircraft controller whilst relative time measures the time passed since the last restart of the aircraft in the simulation environment. A restart brings back the aircraft to the initial position position on the start field, and thus we make sure to reset the desired path plus the relative time, too, in order to bring back the system to a initial state without the need to reset the controller. In the code, this functionality is implemented in lines 164–178 in Autopilot.pde.

2.3. Path Generation

Path generation is connected with the user interface. By typing in certain commands in programs such as `hterm`, we could assign different acrobatic gestures to the plane.

Trajectories are designed in function Trajectory. The idea of designing a trajectory is calculating out the desired postion of the plane 1 second later from the actual position of the aircraft, using the function below:

$$\vec{L}_{is}^{i} = \vec{P}_{des}^{i} - \vec{P}_{is}^{i} \tag{1}$$

And by using the $\vec{L}_{is}^{i}$ vector, we could control the aircraft like dragging the string of a kite.

Currently we have 9 different acrobatic gestures. They are divided into two categories, one stable flight paths and the other temoprary flight gestures.

Stable gestures

Stable flight gestures serve as kind of default flying states, they are maintained when undisturbed, can be also interrupted at any time by other flying modes. To choose from different gestures, we can type in the API interface the following commands:

`'fg'`: glide or go-straight mode. Speed is set to 50 in the forward (or Y) direction.

`'fb'`: back glide mode. Speed is set to 50 in the backward (or -Y) direction.

`'fc'`: circle mode. Testing radius is 70, can be changed in function Trajectory. It has a slight going up speed of 1 to avoid crashing when manoeuvring at a low altitude.

'fs': snake mode. Now tested is a gesture that goes forward (Y direction), and has a left/right snake sinusoidal turns in X direction. Radius is set to 50.

Temporal acrobatic gestures

Thest gestures are temporal ones, which have certain durations and executed from beginning till end. Afterwards the aircraft changes into default flying states -like fly forward or backward. Commands for them are:

'ft': takeoff mode. In this mode the plane has a relatively low climb rate for a safe takeoff.

'fl': looping mode. The plane performs a backward-direction loop by first climbing up and flipping over. A whole barrel loop can be achieved by two backward-direction looping.

'fr': roll mode. The plane rolls over 360 degrees for one time in this mode. The plane finishes rolling in two steps, each turn is 180 degrees.

'fh': half-circle mode. The aircraft flies a half-circle, which would be easier for a downward looping and then flies straight forward.

'fc': climb up mode. Takeoff mode has a too low climb rate, in this mode climb rate is bigger for a quicker climb up.

These acrobatic flight durations are defined and can be changed in function Trajectory. In each gesture's sub-function, parameters such as speed, climb rate or roll speed can also be changed.

2.4. Processing of Control Variables

Processing the Sensor Data

Computing the Desired Acceleration

Introduce look ahead distance, explain, why important -> $L_i s$ is always > 0 and ensures a good acceleration command

Determining the Desired Speed

The desired speed of the aircraft is given implicitly with the desired path and can be gathered by calculating the derivative of $\vec{P}^i_{des}$:

$$\vec{v}^i_{Path}(t_k) = \frac{\vec{P}^i_{des}(t_k) - \vec{P}^i_{des}(t_{k-1})}{\Delta T} \quad . \tag{2}$$

But the desired speed is not the control variable of the throttle controller. As shown in Figure 1, the input for the throttle controller is ΔL which is the distance of the aircraft from the desired

position with respect to the look-ahead distance L_{des} and is computed by

$$\Delta L = \frac{\vec{v}^i_{Path} \cdot \vec{L}^i_{is}}{|\vec{v}^i_{Path}|} - L_{des} = L_{is,Path} - L_{des} \quad . \tag{3}$$

Here, $L_{is,Path}$ is the distance of the aircraft concerning/on the desired flight direction expressed in $\vec{v}^i_{Path}$. This relation is shown graphically in **Figure 2**. The inner structure of the component `Path Delay` is depicted in **Figure 3**. Finally, ΔL is divided by a constant value K and handed to the throttle controller as input $e_{Throttle}$ (see Figure 4).

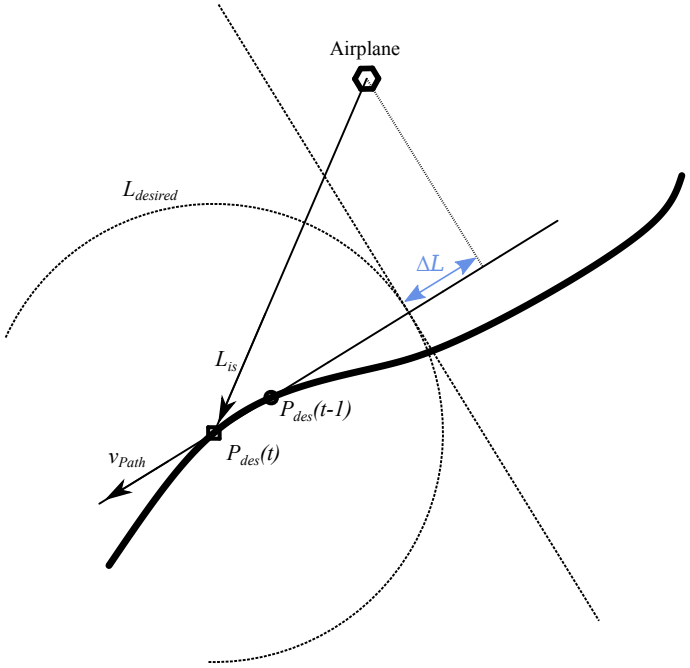

Figure 2: Determination of the input for the throttle-controller ΔL.

With this control mechanism, the aircraft will adjust the throttle in order to always stay behind the desired position by the length of the look ahead distance L_{des} which is the intersection of the desired flight direction $\vec{v}^i_{Path}$ and the circle of L_{des} as shown in Figure 2. In this point, ΔL will become zero, which is the control target of the control structure (of course, the whole tangent of the intersection point will meet the control target of $\Delta L = 0$ but the rest of the control structure will force the aircraft onto the desired path and thus to this one intersection point).

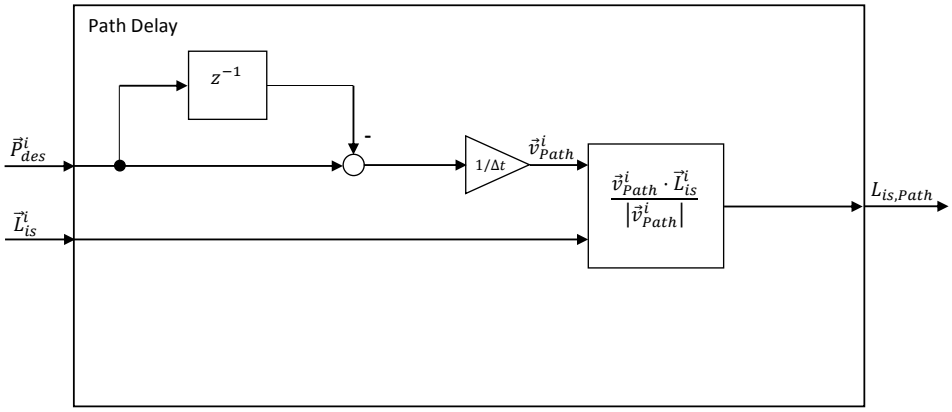

Figure 3: Inner structure of the component `Path Delay`.

2.5. Control Structure

In the `Control Structure`, the four actuators of the aircraft, namely throttle, aileron, rudder, and elevator, are regulated by PID-controllers with an error E as input value:

$$output(t) = K_p \cdot E(t) + K_i \cdot \int_{t_0}^{t} E(\tau)\,\mathrm{d}\tau + K_d \cdot \frac{\mathrm{d}}{\mathrm{d}t}E(t) \quad . \tag{4}$$

Throttle

The input error of the throttle is the sum of current aircraft velocity $|\vec{V}^v|$ and the distance of the aircraft $e_{Throttle}$, lagging behind its desired position, as described in Section 2.4:

$$E_{throttle} = |\vec{V}^v| + e_{Throttle} \quad . \tag{5}$$

The aircraft velocity term is for keeping the engine speed at the desired level, only the error term can increase or decrease the engine speed.

Aileron

The aileron controller is implemented as described in the work of S. Park[**?** , p. 71]. A reference vector $\hat{e}_{roll}$, always pointing out of the roof of the aircraft, is compared with the acceleration command $\vec{A}^b_{CMD}$. The control target is to let both vectors point into the same direction (this is shown graphically in the paper of S. Park on page 70 [**?**]. Mathematically, this is achieved with following

equation:

$$E_{aileron} = \arcsin\left(\hat{e}_{roll} \times \frac{\vec{A}^b_{CMD}}{|\vec{A}^b_{CMD}|}\right)_x \tag{6}$$

The reference vector $\hat{e}_{roll}$ can be modified by the angle ϕ_{ref}. At the value of $\phi_{ref} = 90°$, $\hat{e}_{roll}$ is pointing out of the right wing of the aircraft; at $\phi_{ref} = 180°$, the reference vector is pointing out of the fuselage. Altering the angle of the reference vector can be used for performing a roll manoeuvre.

Rudder and Elevator

Rudder and elevator are controlled with the acceleration command $\vec{a}^b_{CMD}$, which does not contain the gravity term $\vec{g}^b$. The rudder control takes the y-component as input, and the elevator control uses the z-component:

$$E_{rudder} = (\vec{a}^b_{CMD})_y \quad , \quad E_{elevator} = (\vec{a}^b_{CMD})_z \tag{7}$$

The `Control Structure` is depicted in symbolic form in **Figure 4**.

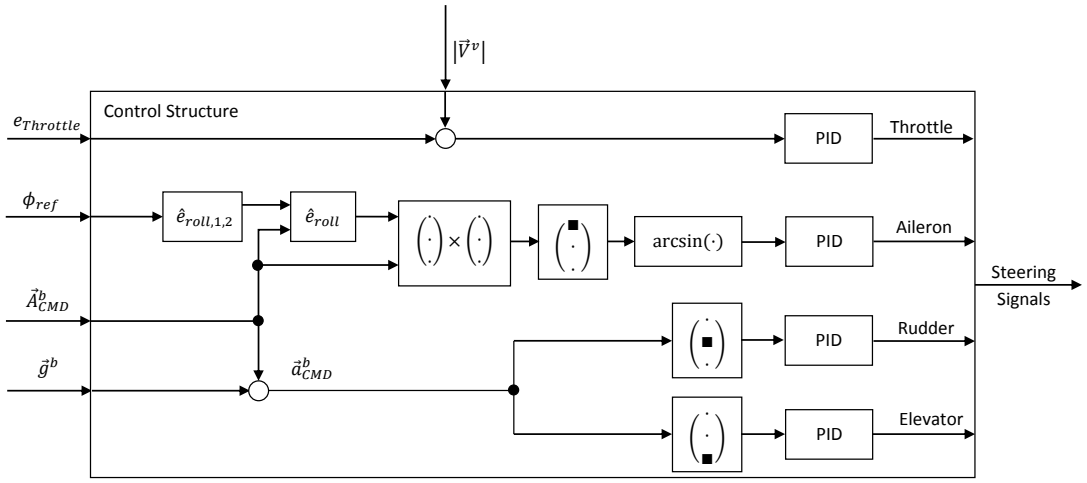

Figure 4: Inner structure of the component `Control Structure`.

2.6. GPS Signal Estimation

In the case of an inverted flight attitude, a GPS-signal cannot be received due to GPS receivers not pointing to the sky. Therefore, the position of the aircraft has to be estimated with signals from measuring instruments, still available, which is the data from accelerometers and gyroscopes.

However, in the simulation, position data is always provided perfectly, so an inverted flight attitude has first to be detected and position signals have then to be ignored in order to realise a *real* GPS-signal.

So, the function NoSignalAvailableGPS is implemented, observing the euler angles of the aircraft, wherewith the flight attitude of the aircraft is depicted. As soon as the roll angle phi or pitch angle θ is according to amount greater than $60°$, this function activates the *position estimation* which overwrites the ideal position signals.

The *position estimation*-principle, implemented in the function estimateStateVars, is a method of integration [**?**]. First, the derivatives of the Euler angles are computed with the available rotation rates by changing the coordinate frame. Then, the Euler-derivatives are time-step integrated with the previous values to obtain the new Euler angles, e.g.:

$$\phi_{t_k} = \frac{1}{2}(\dot{\phi}_{t_k} + \dot{\phi}_{t_{k-1}}) \cdot \Delta T + \phi_{t_k} \quad . \tag{8}$$

In the next step, the acceleration of the aircraft is depicted with the data of the accelerometer A plus data of the rotation rates pqr, where $\mathbf{C_r}$ offsets the location of the IMU and $\mathcal{R}_b^i$ is the transformation from body frame to inertial frame:

$$\ddot{P}^i(t_k) = \mathcal{R}_b^i(\phi, \theta, \psi)(t_k) \cdot A(t_k) + \mathbf{C_r} \cdot [pqr(t_k) \times uvw(t_{k-1})] \quad . \tag{9}$$

Here, uvw is the velocity of the aircraft in body frame:

$$uvw(t_k) = \mathcal{R}_i^b(\phi, \theta, \psi)(t_k) \cdot \dot{P}^i(t_k) \quad . \tag{10}$$

With the same time-step integration method as shown in Equation 8, the aircraft-velocity, and with a further integration step, the position of the aircraft can be obtained:

$$\ddot{P}^i \rightarrow \dot{P}^i \rightarrow P^i \quad . \tag{11}$$

As soon as the aircraft is able to receive a GPS-signal again, the estimation is switched off and the ideal position data is used.

3. Results

In summary, the presented and implemented control structure showed very good results for various aerobatic manoeuvres which are shown in the following.

Restraints

But beforehand, it is to mention that the rudder was not activated as the results were recorded. The reason was a faulty adapter software which caused a disconnection with the controller after a few seconds of flight. Instead, we used a stable working adapter software which did not support the control of the rudder.

Also, the GPS signal estimation could not be activated due to imprecise estimation results. The reasons were the lack of information about the used units of the measurement data and the position of the accelerometers in the aircraft which has to be considered in a compensation term.

Analysis of flown Trajectories

In this project, we also adopted Matlab to help us analyse our flown trajectories. What we wanted is a 3D recording of the real flown trajectory, with this it is easier to find out possible errors in control or path parameter designs.

Matlab code and results are stored in analysis part. Basically we compare 3 sets of data recorded from program `hterm`: desired path, real trajectory and $\vec{L}_{is}^{i}$ vector. By dragging around the 3D fig format plot in Matlab we can have a direct notion of how the aircraft had flown and also changes in $\vec{L}_{is}^{i}$ vector.

Here shown in **Figure 5** are plots with the circle mode. The desired path is shown above and the real trajectory is underneath. We can see that the aircraft adjusted itself to the sharp turn quickly and followed the desired path quite accurate. The takeoff part and glide flight are also flown desirably according to the recorded plot.

For the snake flight also, here we tested the left/right sinusoidal turns in **Figure 6**. In coordinates this corresponds to sinusoidal changes in X directions. Similarily, the aircraft had adjusted itself for the first few seconds and then followed the previously designed path.

Rolling are shown in **Figure 7**. We first designed a straight path and then performed the acrobatic rolling. During rolling, the aircraft would inevitably fly off-course a bit, especially when the rolling is performed at a lower rate. It is also proven from the plot, the plane did the full roll in two steps and then adjusted itself to the desired straight path again.

A very important parameter to reassure that the plane would follow the desired path is the look ahead distance L_{des}. If L_{des} is too large, and the designed path is not a straight line, the plane would probably fly randomly or even finally crash. A reasonable short L_{des} can provide good responses of the plane when changing courses.

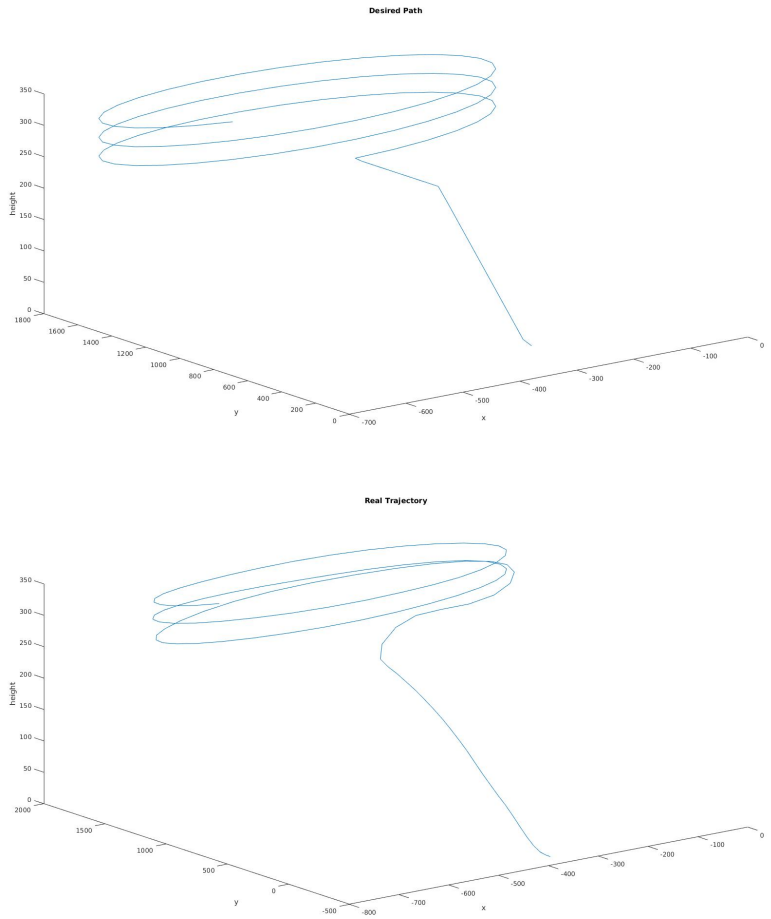

Figure 5: Comparison of desired and real trajectories of circle flights.

A. Appendix

A.1. References

A.2. Notation

t_k	Point in time (time-discrete)
ΔT	Time period of one time step of the system

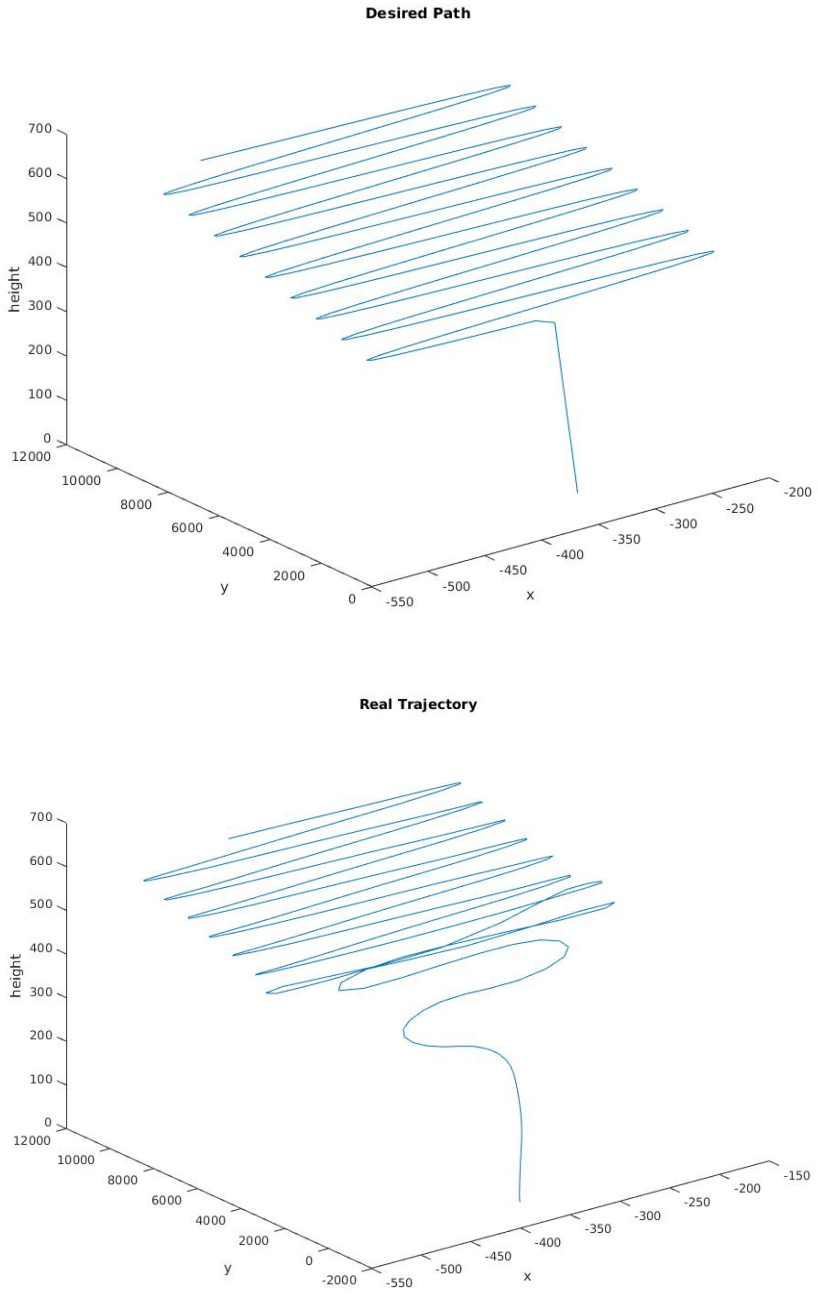

Figure 6: Comparison of desired and real trajectories of snake flights.

ϕ_{ref}	Angle of reference vector for aileron control, forces the roll-angle
$\vec{P}^i_{des}$	Desired position of the aircraft (inertial frame)

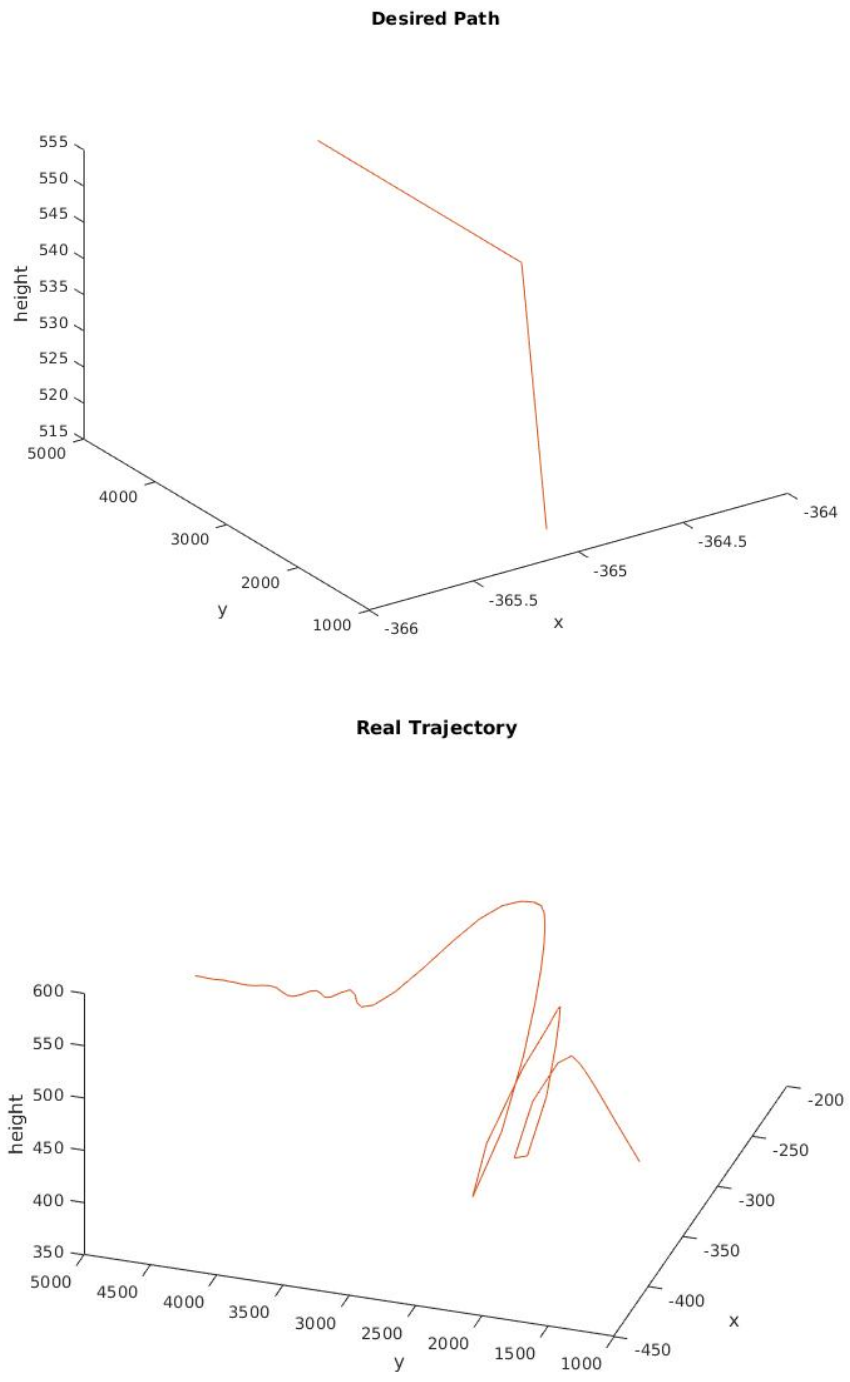

Figure 7: Comparison of desired and real trajectories of rolling flights.

$\vec{P}_{is}^{i}$	Actual position of the aircraft (inertial frame)
$\vec{L}_{is}^{i}$	Difference between desired and actual position (inertial frame)
$\vec{a}_{CMD}^{v}$	Desired Acceleration of the aircraft (vehicle frame)
$\vec{g}^{v,b}$	Acceleration by gravity (vehicle,body frame)
$\vec{A}_{CMD}^{v,b}$	Desired Acceleration with gravity considered (vehicle,body frame)
L_{des}	Look-ahead distance
$L_{is,Path}$	Distance concerning the desired path, the aircraft lags behind
ΔL	Desired distance the aircraft needs to catch up
$e_{Throttle}$	Input error for the throttle controller
(ϕ, θ, ψ)	Euler angles of the aircraft
$\vec{V}^{v}$	Velocity vector of the aircraft (vehicle frame)