ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Bachelor-Thesis

Human-Machine Interface for Operating a Blimb

Spring Term 2012

Supervised by:
Konrad Rudin
Mora Javier Alonso

Authors:
Krebs Matthias
Ledergerber Anton

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Human-Machine Interface for Operating a Blimb

is original work which I alone have authored and which is written in my own words.[1]

Authors

Anton	Ledergerber
Matthias	Krebs

Supervising lecturer

Konrad	Rudin
Javier Alonso	Mora

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`http://www.ethz.ch/students/exams/plagiarism_s_en.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date	Signature
Place and date	Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Abstract

This Bachelors thesis was written within project Skye, which is about a high agile spherical blimp called Skye. Main tasks of this novel system are image capturing and entertainment. The four tetrahedrally arranged motors of the system can be rotated to thrust into any tangential direction to the hull. Combined with the symmetrical design this allows to control all motions in the three dimensional space. A camera system consisting of a high quality camera as well as two additional low size cameras can be used to capture and store images onboard or transmit them to a ground station.
To control the unusual freedom of 6DOF, manual control modes as well as automatic control and a combined manual and automatic control mode have been developed within this thesis. A combination of a tablet PC with a 3D mouse proofed to be suitable to provide intuitive control. A GUI has been developed that easily allows to switch between control modes, to display the system's status and even to control the system via a touch interface. The touch input screen is underlaid by a map for translations and the video stream for rotations respectively. For automatic control, optimal trajectories for Skye have been designed and compared. Three approaches to track them have been implemented and tested in a simulation environment.

Acknowledgements

Without the help of many individuals this thesis would not have been possible. We received the necessary support from all sides throughout the project to realize this HMI. We would like to thank everybody who helped us during its development and therefore also contributed to the successful rollout where Skye flew smoothly piloted with this HMI through the ETH main building.

Special thanks to Prof. Dr. Roland Y. Siegwart, who gave us the opportunity to do this thesis within project Skye at his institute, the ASL.

Also special thanks to Dr. Paul Beardsley, who constantly motivated us and inspired us with great ideas for an intuitive HMI.

We would also like to express our gratitude and thanks to our supervisors, the PhD students Konrad Rudin and Javier Alonso Mora, for the valuable guidance and advice throughout this thesis.

Out of many more individuals, we would like to mention by name Lorenz Meier, Gerhard Röthlin and Alexander Rudyk. They especially helped us with their programming skills when having reached an impasse.

Of course this thesis was only possible because of the whole team Skye. Thank you very much! It was a pleasure to work with all of you.

Finally we would like to express our deepest gratitude to our families and beloved ones, Marina and Eliane. They gave us encouragement and showed great patience when it was most required.

We gained a lot of learning experience in this thesis and it was of real pleasure to see Skye finally fly with the HMI developed in this thesis.

Zürich, June 2012

Matthias Krebs
Anton Ledergerber

Symbols

Symbols

bold	Vector
$\lvert\cdots\rvert$	Absulute value
$\lVert\cdots\rVert$	Euler norm
$\mathbf{q}$	3 dimensional waypoint vector
$\mathbf{r}(t)$	3 dimensional position state vector
$\mathbf{p}(u)$	3 dimensional path vector
$\tilde{\mathbf{p}}(t)$	3 dimensianal trajectory vector
u	Parameter for path
t	Parameter for trajectory; time
L_p	Geometrical length of path or trajectory
T_p	Time length of trajectory
G^h	Geometrical continuity of a curve
C^h	Parametric continuity of a curve
$n+1$	Number of waypoints
$n_{knots}+1$	Number of knots

Indices

cl	Closest point on path or trajectory
k	Index of waypoint $0 \leq k \leq n$

Acronyms and Abbreviations

AC	Assisted Control Mode
API	Application Programming Interface
ASL	Autonomous Systems Lab
DC	Direct Control Mode
DOF	Degree of Freedom
ETH	Eidgenössische Technische Hochschule
FAC	Full Automatic Control Mode
GPS	Global Positioning System
GUI	Graphical User Interface

HAC	Half Automatic Control Mode
HMI	Human-Machine Interface
IMU	Inertial Measurements Unit
NED	North-East-Down
PX4FMU	Pixhawk Flight Management Unit
RC	Remote Control

Chapter 1

Introduction

1.1 Context

Within the bachelor's course in mechanical engineering at the ETH Zurich it is offered to join a focus project instead of focus lectures. A focus project is often a novel or experimental design of a prototype[1] and elaborated over a period of two semesters within a team of mainly students in mechanical engineering, but often in cooperation with students from interdisciplinary courses. The project Skye is settled at the Autonomous Systems Lab (ASL) at ETH Zurich and realized in cooperation with Disney Research Zurich (DRZ).
Project Skye then was a focus project started in autumn 2011 and will be continued even after the "official" end of the project in summer 2012. The *Rollout* event in the historic main building of ETH Zurich was a unique moment when the prototype wafted over the overwhelmed spectators.

Figure 1.1: The highligth of project *Skye*: The *Rollout* event on May 25, 2012 in the main hall of ETH Zurich.

[1] Some former projects can be found on `http://www.asl.ethz.ch/education/bachelor/focus`.

1.2 System Overview

Skye is a high agile unmanned aerial vehicle in form of a spherical blimp. It was developed to achieve a system to capture images for 3D reconstruction as well as to obtain entertainment functions (as airshows, human interaction etc.). As there exist already systems that fullfil these requirements[2] it was claimed to build a system that provides increased flight time and higher operating safety. Skye is the system that accomplishes these improvements.
The two-layered spheric hull is filled with lighter-than-air gas helium. The gas is filled in with an overpressure of 15 mbar that ensures a stiff hull surface without the need of any rigid structure. Four identical motor units are placed tetrahedrally on the hull by Velcro. The *AHM 23-10 EPP 1045* thrusters are pointing tangential to the sphere. Their orientation be changed to any tangential direction using a Maxon *A-Max 16* motor for each unit. The center of gravity is approximately at the same position as the center of buoyancy. Further on, the gravitational force exceeds the buoyancy only for a minimum[3].

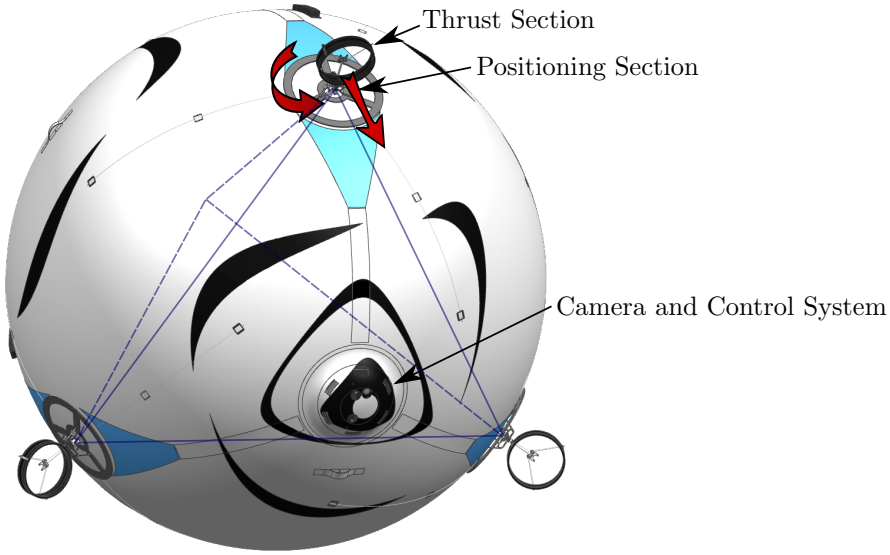


Figure 1.2: CAD sketch of Skye. Red arrows indicate thrust and orientation of the motor units. Blue tetrahedron indicates symmetrical arrangement of them.

Two light weighted *Bluefox* cameras as well as a high end *Prosilica* high resolution camera constitute the the camera system. A solid state disk as well as the cameras are connected to a onboard *Intel Atom* embedded computer. They are all together placed on a electronics platform which is detached to the hull. The computer runs a Linux distribution and can be linked to via a Wi-Fi module. A low level *Cortex M4* processor within a *PX4FMU* Inertial Measurement Unit (IMU) controls the flight system. It is equipped with a barometer, gyroscope, magnetometer, accelerometer and a GPS receiver. The control communication is realized with both a *Lairtech Xbee* module and a *Futaba Rasst 2.4GHz* receiver.
The high symmetric system properties combined with the eight actuations enable to fully control the motion in space. The two redundant DOF are used for optimization (see [10]).
Finally, a system is only as good as the user appreciates it. Therefore a optimal

[2]Mainly quadrotors, but also other UAV.
[3]So the system sinks slowly to ground if turned off.

Human-Machine Interface (HMI) is inevitable to provide a useful system. The aim of this thesis is to realize such a HMI in an optimal way for project Skye.

1.3 Goals

For this bachelor thesis, intuitive interfaces to control and observe the system Skye had to be developed. This includes the implementation of a Graphical User Interface (GUI) for a computer based Ground Station (GS) based on any existing solution. Further on, different manual control modes and suitable Human-Machine Interfaces must have been elaborated and implemented. Finally, automatic manoeuvres (expressed by paths and trajectories) had to be generated considering simplified dynamic system constraints based on the results from a model elaborated in [8]. These goals are summarized in table 1.1.

Goal	Specification
Control Modes	Intuitive Control for Manual for 6DOF Blimp
HMI	Suitable for Control Modes
GUI	Provide System Info, Adapt Properties
Trajectories	Automatic Maneuvers

Table 1.1: Main goals of this thesis

1.4 Similar Systems and their HMI

The HMI (or control device) for a machine mainly depends on its number of control inputs and its level of automation. To control more control inputs, additional sticks or similar have to be provided. Automation of the system can simplify its control, but requests for emergency backups in case of automation fail.
The number of control inputs depends on the system's properties. This is shown in table 1.2[4]. A car can move in a plane. Because it is a nonholonomic system, its states can not be changed individually. Therefore the number of inputs by the driver (two: steering wheel and acceleration pedals) is below the actual degree of motion (x, y, yaw). Indeed, a two dimensional input leads to a very intuitive HMI. For a ballbot or a legged robot each of the 3 degrees of motion can be steered individually. Therefore the HMI device can enable three inputs as it is provided for instance by a Qgo Sphere[5] or a gamepad. A quadrotor is comparable to the system Skye. Its static motion is limited to 4 degrees of freedom (x, y, z, yaw) and therefore a gamepad with two sticks or even a smartphone are suitable control devices. As Skye is (due to its mechanics and actuation symmetry) steerable using all degrees of freedom in the 3 dimensional space it requires a HMI with up to 6 control inputs.

[4] Pictures ©by their owners:
AMZ racing: `http://www.amz.ethz.ch/`
Rezero: `http://rezero.ethz.ch/`
Arac: `http://www.arac.ethz.ch/`
arDrone: `http://ardrone2.parrot.com/`

[5] See [6], section 3.2.1 or `http://www.quasmo.ch/index.php/qgosphere` for more informations about Qgo Sphere.

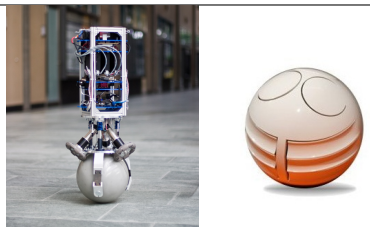
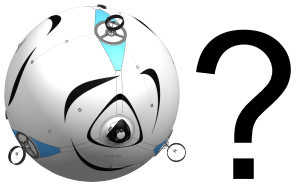
	
Car steered by wheel and pedal 2 control inputs	Ballbot steered by Qgo Sphere 3 control inputs
	
Spiderbot steered by gamepad 3 control inputs	Quadrocopter steered by smartphone 4 control inputs
	
Skye 6 control inputs	

Table 1.2: Examples for systems and their HMI. The number of inputs depends on the number of degree of motion reduced by nonholonomic constraints.

1.5 Structure of the Report

This report is divided into two parts. The first part including chapter 2 and 3 treats the problem described above to find a suitable solution for piloting Skye. First, the different elaborated control modes are shown in chapter 2. The following chapter 3 contains the evaluation of different control devices and describes the realized HMI. Especially the GUI is described in detail in section 3.3.2.
The second part of this thesis is a more technical elaboration of optimal trajectory generation for the system Skye (chapter 4). Trajectory control results are shown in this chapter as well. Further results are found in the appendix.

Chapter 2

The different Control Modes

As mentioned in section 1.2, one of Skye's unique properties are the decoupled six DOF in space. This means, the system has the possibility to turn around an arbitrary axis while moving in any direction. This ability has to be used to fulfill the operation tasks in an optimal way. Although, an intuitive handling for the pilot has to be granted in any way.
The first approach is to provide a manual control mode that allows the pilot to directly control the six DOF. A detailed description will be given in section 2.1.
In contrast to the unrestricted control possibilities for manual control, a more autonomous control would allow the pilot to focus on a specific motion, e.g. the orientation of the camera. Therefore, a part of the six DOF motion of *Skye* is automated[1]. In section 2.2 this is described in more detail. Furthermore, a basic *test phase* control mode lets the operator directly access the eight actuators individually. Table 2.1 gives an overview of the control modes. Those actual realization is presented in chapter 3.

Test Phase	Direct Control	Assisted Control	Half Automatic	Full Automatic
Thrust 1	Thrust x	Velocity x	Velocity	Velocity
Thrust 2	Thrust y	*Velocity* y	Rotation x	Rotation x
Thrust 3	Thrust z	*Velocity* z	Rotation y	Waypoints
Thrust 4	Moment x	Rotation x	Rotation z	Camera target
Direction 1	Moment y	Rotation y	Waypoints	
Direction 2	Moment z	Rotation z		
Direction 3				
Direction 4				

Table 2.1: With the control modes, different inputs can be given by the user. The inputs in *italic* are not available for *Assisted RC Control*.

2.1 Manual Control Modes

Test Phase

As the prototype Skye was built in parallel to this thesis, the overall functionality of the system had to be checked piecewise. The *test phase* control mode was therefore designed to test the propulsion's behaviour. The number of revolutions (which is

[1]The prototype designed for this project does not include any environment detections. Therefore, *autonomous* control with obstacle detection will not yet be possible.

actually defining the thrust) as well as the reference orientation of each of the four actuation units can be set individually[2].

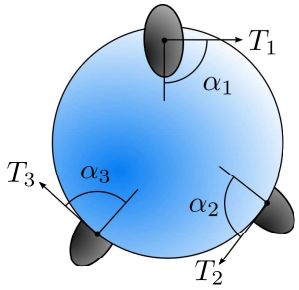
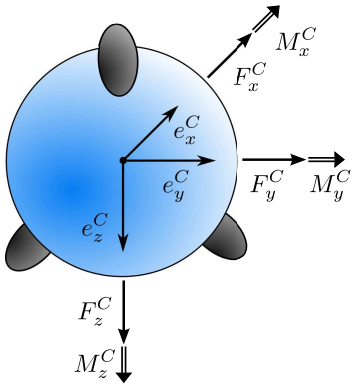


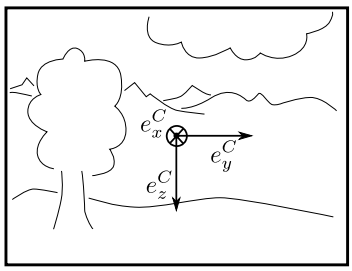
Figure 2.1: Test Phase

Direct Control

A second control mode that mainly intends to verify the system's properties is the *direct control* mode where the operator controls the desired force and moment vectors. This six DOF control therefore needs the transformation of the forces and moments to the eight actuators[3] only. No information about the system's properties and states are needed (motor units excluded). The pilot's input is interpreted as the force and moment components in body fixed coordinates. As the camera system (the eye of Skye) enables the best way to recognize Skye's attitude, the coordinate frame was aligned to the camera's orientation. This is illustrated in figure 2.2, where $\{e_x^C, e_y^C, e_z^C\}$ represents the camera orientated frame (e_x^C collinear with the camera axis, e_y^C pointing to the right, e_z^C pointing downwards) and $\{F_x^C, F_y^C, F_z^C, M_x^C, M_y^C, M_z^C\}$ the user input.



Direct Control



Camera Frame

Figure 2.2: Direct Control (left) and the orientation of the camera frame in relation to the camera's field of view (right).

[2] In [10] referred to as input for the *Thrust Section* and *Positioning Section* respectively.
[3] See [10] for details about the optimization criteria for the motor allocation.

Assisted Control

The *direct control* mode demands well trained skills to properly navigate *Skye*. This is especially due to the low damped rotations [8] and the symmetrical appearance of the system. To cope with the low damped rotations, a stabilizing attitude controller is needed. In order to better deal with the symmetrical appearance, the translational inputs can be interpreted in a earth fixed NED coordinate frame $\{e_x^I, e_y^I, e_z^I\}$. By using a proper state estimation, the pilot's commands can directly be interpreted as velocities and angular velocities respectively[4]. This is actually realized in the *assisted control* mode. Figure 2.3 illustrates the user's input $\{v_x^I, v_y^I, v_z^I, \omega_x^C, \omega_y^C, \omega_z^C\}$. If a user interface for less than six DOF is used, e.g. a remote controller[5] which has only four DOF, the user input is restricted to $\{v_x^C, \omega_x^C, \omega_y^C, \omega_z^C\}$ which is similar to helicopter or airplane control.

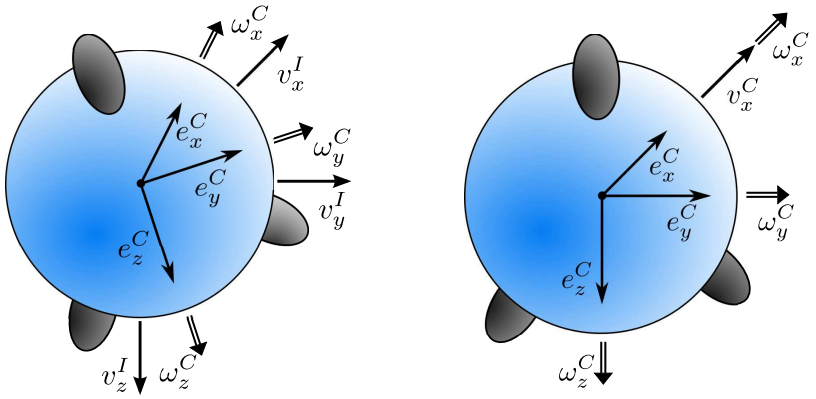


Figure 2.3: Assisted Control with controlling six (left) and four (right) stabilized degrees of freedom.

2.2 Automatic Control Modes

Half Automatic Control

When scanning an object for 3D reconstruction, it is often known in advance where the pictures should be taken from. Therefore, a waypoint based controlling would be desirable. The *half automatic control* mode allows to define a set of $n+1$ waypoints $\{\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_n\}$ for the system's position. The rotational degrees of freedom $\{\omega_x^C, \omega_y^C, \omega_z^C\}$ remain manually controllable. This allows to adjust the camera's orientation while automatically following the path generated through the waypoints. For convenient use, it is important that the user can reduce the translational velocity on the path for instance if an additional view out from the current position is demanded.

[4] See [9] for the state estimation and controller design.
[5] A remote controller (RC) was implemented as redundant control link, see section 3.2.1 or [11].

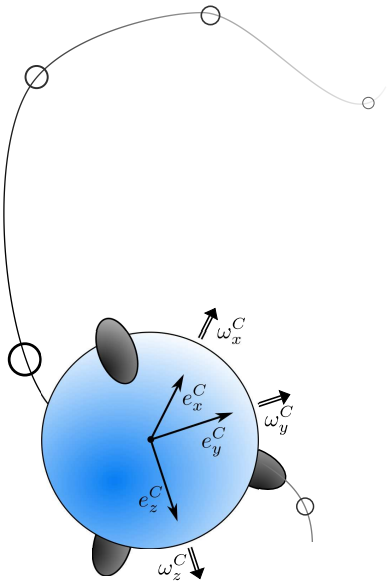


Figure 2.4: Half Automatic Control

Full Automatic Control

An even more advanced control mode would also automatically adjust the system's orientation. Therefore, also the camera target position has to be indicated. For this bachelor thesis, only a solution considering a fix target point has been elaborated. This could easily be extended to multiple target points similar to the waypoints. As it can be seen in figure 2.5, even with this method the orientation around the camera axis e_x^C is not defined explicitly. Therefore, ω_x^C would remain as a manual user input. Indeed, for capturing images it is convenient to adjust the orientation to keep the horizon aligned.

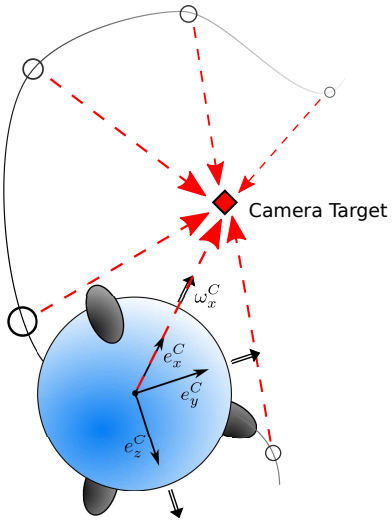


Figure 2.5: Full Automatic Control

Chapter 3

Finding a Hardware and Software Solution

In the following it is described how the previously defined control modes were realized and what else lead to the finally realized HMI which is described in section 3.3. On the one hand there was the hardware which had to be chosen and on the other hand the software to realize the control modes with that hardware had to be written.

3.1 Requirements

One of the goals of this bachelor thesis was to develop a HMI which was tailored to Skye. In the following listing, all requirements demanded for such a HMI are pointed out:

The hardware used for the HMI should...

... offer intuitive control for six DOF, i.e. fit the defined control modes,

... be portable,

... offer wireless connection, i.e. ports for attaching a XBee module and a Wi-Fi router,

... be able to display the video stream of Skye,

... offer the possibility to set waypoints for Skye,

whereas the used operation system should be compatible with the driver of the XBee module and with ROS[1].

3.2 Existing Solutions

Existing HMI solutions were analyzed in order to find out, whether they would be convenient to adopt and whether they would fulfill the requirements on their own.

[1] The decision to use a XBee module and Wi-Fi for communication is explained in [11]. There it is also explained how ROS is used to process imagery. Detailed description about ROS, the Robot Operating System, can be found on `http://www.ros.org/wiki/`.

3.2.1 Hardware

Figure 3.1 shows all devices which were taken into closer consideration. The gamepad option was drapped as it would offer the same possibilities as RC does with the drawback of not being a stand alone device. Smartphones could not be used as they run with operation systems not compatible with ROS[2].

RC Tablet 3D mouse

Qgo Sphere Joystick

Figure 3.1: Different devices taken into consideration

Table 3.1 shows the rating of the considered devices. It is visible that a solution in combination with a tablet had to be used, as only a tablet is able to fulfill the criteria for automatic controls and to run ROS. Although the Qgo Sphere[3] would be most intuitive to use with Skye, a 3D mouse (a space navigator from 3DConnection) was chosen as a supplementary device to the tablet. Because of the fix-installed infrared sensors needed for the detection of the sphere's translational movements, the Qgo Sphere is not portable, which made it inconvenient to use for Skye. A joystick is not designed for six DOF and is therefore much less intuitive in use as a 3D mouse. For the tablet a Lenovo X220 was chosen as it can be transformed to a common notebook. This allows easy postprocessing of any gathered data. Additionally, a Futaba 7C RC serves as a backup in case of any breakdown within the main HMI.

[2]This held true when this thesis was started and might be different nowadays as almost everything could be handled without ROS.

[3]In [6] the usage of the Qgo Sphere as an input device for a ballbot is discussed.

Device		RC	Tablet	3D Mouse	Qgo Sphere	Joystick
Stand alone		✓	✓	-	-	-
Matching	Test Phase	-	✓	-	-	-
	Direct Control	✓	✓	✓	✓	✓
	Assisted Control	✓	✓	✓	✓	✓
	Half Automatic Control	-	✓	-	-	-
	Full Automatic Control	-	✓	-	-	-
Live View		-	✓	-	-	-
Intuitive		quite	quite	very	most	quite
Portable		most	quite	quite	not	quite

Table 3.1: Requirements and Expectations for the Hardware

3.2.2 Software

Google and Apple showed how intuitive a GUI can be in combination with a touchscreen with their operating systems for smartphones. They offer the user unknown flexibility in adopting the GUI to his needs. However, out of Windows, Ubuntu, Mac OS, iOS and Android, Ubuntu was the only operating system which provides complete support for ROS. Therefore Ubuntu 11.10 was chosen as the operating system for the tablet.
As it was clear that there was no GUI ready to use unchanged for the HMI, open source software was analyzed in order to adopt it for Skye. Using QGroundControl (described in section 3.3.2) was the best option. On the one hand it already offered the commonly used tools to control a UAV and on the other hand a good support in adopting it to the needs of Skye was granted as it was developed at the ETH.

3.3 Realization

3.3.1 Compact and Convenient Solution

Using a 3D mouse in combination with a tablet computer has a lot of advantages. Both devices can be put on a board with suspenders. Like this, the pilot has a compact portable control unit, without any external hardware interconnected (compare figure 3.2). While the 3D mouse provides intuitive control of the six DOF, the tablet with its GUI is used to filter and route the signals of the 3D mouse or serves as a highly modular touch input device on its own. With QGroundControl running on it, it also enables the pilot to adjust the controller and to set waypoints, while during the whole flight the live view of Skye is displayed (More on that in section 3.3.2). In case the tablet should crash, the RC can be used to bring down Skye safely to ground.

Figure 3.2: The compact, portable control unit in action

3.3.2 QGroundControl

The GUI QGroundControl for controlling unmanned vehicles was initially developed for Pixhawk[4], a vision based UAV project at the informatics department of ETH. It is a complete open source project based on Qt, which is a cross-platform-framework for GUIs written in `C++`. For this thesis, the complete QGroundControl source code was taken and adopted for the needs of project Skye. This varied from adding single lines to develop complete new additional widgets[5]. These widgets are described below. Inspiration and know-how was taken from [13] and nice tutorials and samples on `http://qt.nokia.com/`. In the following it is explained, how the control modes, defined in chapter 2, were realized with the chosen devices and with the GUI QGroundControl[6].
To allow full control of system skye, a derived subclass to handle system specific procedures had to be programmed. This had to be done within compatibility to the messages defined in the Mavlink protocol (see section 3.3.3). Precompiler commands have been used for cross-platform compatibility and Skye system specifications. With the Qt specific signals/slots interface the Skye subclass has been connected with the communication link as well as the user interfaces. An event handler has been integrated using the 3DxWare API, the official development kit for the used 3D mouse.

The Main Control Widget

With five different control modes (Test Phase, Direct Control (DC), Assisted Control (AC), Half Automatic Control (HAC), Full Automatic Control (FAC)) and three different input option (3D Mouse, Touchscreen, Keyboard), a main control widget was needed to choose between these options (see figure 3.3 for its appearance). The Test Phase mode is not meant to be used in flight and has therefore be activated via the menu bar. The two sliders visible in figure 3.3 allow to adjust the input scaling for translations and rotations respectively. Indoor flights need usually smaller but more accurate inputs, while outdoors more power is needed.

[4]For more information on this project, see `https://pixhawk.ethz.ch/`.
[5]Widgets are kind of subwindows of the application.
[6]The full automatic control mode was only simulated in Matlab and therefore no support of the GUI exists.

The colored arm/disarm button is used to (de-)activate the actuators. At the same time it serves as an emergency button.

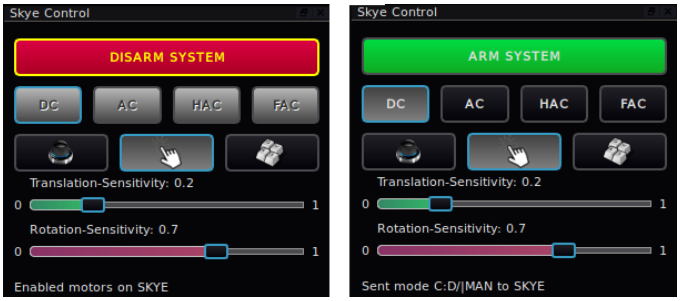


Figure 3.3: The main control widget in which most modes and options can be set. On the right its appearance with activated actuators and and the left its appearance with deactivated actuators.

Test Phase Control

In order to test Skye's four actuators before flight and in the development process, another widget was developed. It allows to adjust each of the eight actuation inputs separately. The input for the four thrusters is set by sliders and the orientation is set by turning the knobs. Again, there is a main button (Activate Engine) to start and stop the motors.

Figure 3.4: The test phase widget

Direct Control

The main device for the direct control mode is the 3D mouse which is connected by USB to the tablet. To use its input signals in QGroundControl, a interface had

to be written. With the two knobs on the device (hardly visible in figure 3.5) the pilot is able to (de-)activate translational or rotational inputs. This allows a very precise control.

Figure 3.5: The 3D mouse used for the direct control

Assisted Control

Feeding velocities instead of a force and a moment makes the usage of the touch-screen beside the 3D mouse more reasonable. For this mode a combination of different widgets is used, shown in figure[7] 3.6. The widgets can be scaled and positioned individually. As a default view for Skye, the shown arrangement of the Head-up-Display (HUD), map and main control widget is provided. If touch input is chosen, the HUD enables rotational touch inputs directly on the displayed live view. The map widget shows the position of Skye and allows to set translational inputs. Note that in a first approach these inputs were always interpreted in the camera frame as the state estimation was not yet running. Intuitively, this has to be transformed to inputs in the North-East-Down (NED) frame which has recently been tested on the system.

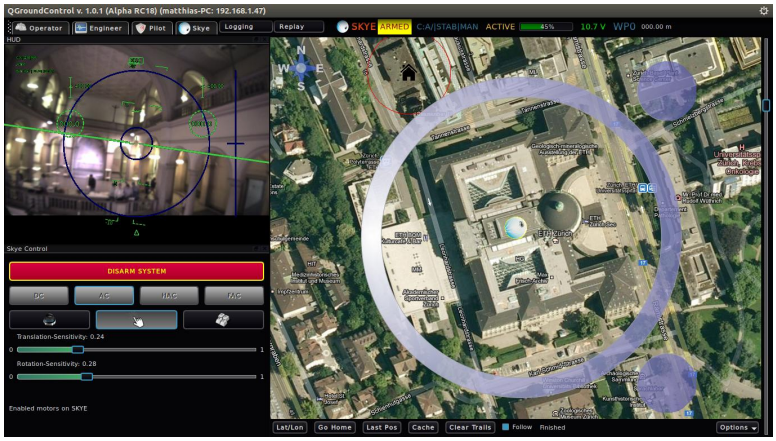


Figure 3.6: The pilot's view on the GUI if assisted control and touch input is activated.

[7] This screenshot is a reproduced situation of the flight in ETH main building. Obviously, no GPS information were available then.

Figure 3.7: The enlarged HUD with touch input in action

Half Automatic Control

For this mode a trajectory is needed. Therefore QGroundControl was adopted to generate it. After setting the waypoints on Google Maps, the user can adjust the their height over ground. In order to properly, the Google Elevation API[8] was used to display the elevation. Once the waypoints are set a interpolating cubic spline is calculated and displayed (compare figure 3.8). Out from the recent position of Skye, a velocity input is calculated using a *Pure Pursuit* approach[9] The part of the path that is already reached is displayed in blue.

Figure 3.8: Generating a trajectory with the GUI

[8] `https://developers.google.com/maps/documentation/elevation/`
[9] In section 4.5.2 *Pure Pursuit* is described how it was implemented in Matlab.

Further on, widgets to adjust camera properties, displaying detailed battery info or to set Direct or Assisted Control to exact values via sliders have been added. Despite of the latter[10] they could not be used in practice. The Direct/Assisted Control widget is shown in figure **??**.

3.3.3 Mavlink

Mavlink (Micro Air Vehicle Communication Protocol) is a very lightway header-only message marshalling library for micro air vehicles[11]. It is a widely used protocol geared for transmission speed and safety. A mavlink packet consists of a six byte prefix including start byte, payload length and sequence number as well as component, system and message ID. A two byte suffix checksum is used. The payload size is maximum 255 bytes. Many autopilots as well as cross-platform software packages are using Mavlink. A set of commonly used messages is provided. The protocol had to be extended for the convenient use with SKYE (an example[12] of such a message definition is shown in listing 3.1). This can be done by defining the additional messages in a xml script. The C-headers will be generated by the Mavlink Generator. The control commands, system telemetry as well as image live stream for SKYE are transmitted using Mavlink protocol. For the transmission of waypoints, an extended waypoint protocol is provided by Mavlink. The waypoints set on the map widget and height profile (see section 3.3.2) as well as the autopilot (*PX4FMU*) are compatible to this protocol.

```
<message id="155" name="SKYE_ASSISTED_CONTROL">
   <description>Control six degrees of freedom. Translational velocity in↩
        Inertial (earth) Frame. Use this manual control mode with the ↩
       requested mode:
"MAV_MODE_ASSISTED_CONTROL_ARMED"</description>
   <field type="uint8_t" name="target_system">System ID</field>
   <field type="float" name="translation_lat">Translation (velocity) in ↩
       Inertial Frame latitude, in m/sec</field>
   <field type="float" name="translation_long">Translation (velocity) in ↩
       Inertial Frame longitude, in m/sec</field>
   <field type="float" name="translation_alt">Translation (velocity) in ↩
       Inertial Frame altitude, in m/sec</field>
   <field type="float" name="rotation_x">Roll (angular velocity), in rad/↩
       sec</field>
   <field type="float" name="rotation_y">Pitch (angular velocity), in rad↩
       /sec</field>
   <field type="float" name="rotation_z">Yaw (angular velocity), in rad/↩
       sec</field>
</message>
```

Listing 3.1: Sequence of skye.xml: Definition of the Assisted Control command Mavlink message. It consists of six 32 bit floating numbers for the input values and a 8 bit integer to indicate the target system's ID (the GUI could actually be connected to more than one UAV at the same time).

[10] The Direct/Assisted Control widget has been used to enable the system verifications in [9] and [8].

[11] Official description from `http://qgroundcontrol.org/mavlink/start`

[12] The full xml file including the message definitions for SKYE is in the appendix (listing A.1).

Chapter 4

Trajectory Planning

For the two most advanced modes, i.e. the Half-Automatic and the Full-Automatic mode described in chapter 2, trajectories had to be generated. In this chapter the best trajectories for SKYE are elaborated and tested with suitable trajectory controllers. Performance results based on a MATLAB simulation are shown.

4.1 Experimental Design

The main application fields of the system SKYE are image capturing and agile performance demonstrations. The waypoints used to test the trajectory algorithms had therefore to be alike these situations. All the results in this chapter belong to the three sample waypoints shown in figure 4.1. The first waypoints e.g. could be used to capture images of the university building. Generally spoken, they represent standard situations for the applications mentioned before. Indeed, to verify the conclusions, sometimes a wider set of waypoints had to be considered.

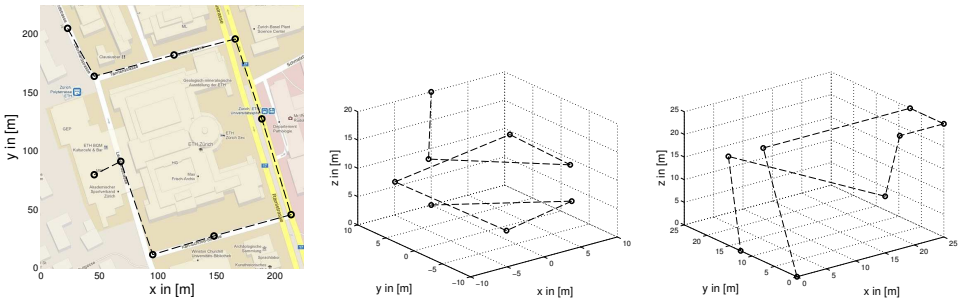


Figure 4.1: The experimental environment based on three samples. **Left:** The *road* waypoints represent the need of low overshoots to not touch obstacles beside the streets. Its long straight ways enable high velocities. **Center:** The *helix* waypoints represents the circumnavigation of any obstacle. It yields to high curvatures of the track. **Right:** The *agile* waypoints include both straight sections as well as high curvatures.

Furthermore, to score and optimize the generated trajectories $\tilde{p}(t)$ and the resulting trace of the system $r(t)$ we set up the following criteria. Criteria (i) to (iii) are referred to as *static criteria* as they do not depend any simulation. The remaining ones (*dynamic criteria*) then mainly depend on the used controller[1].

[1]For detailed description of the used notation consider section 4.2.

Being L_p the length of the path $\mathbf{p}(u)$ and T_p the time of the trajectory $\tilde{\mathbf{p}}(t)$

$$L_p = \int_{u_{min}}^{u_{max}} d\mathbf{p} \qquad T_p = \int_{t_{min}}^{t_{max}} dt \tag{4.1}$$

the criteria are:

i) Average deviation between actual path and chord (straight) connection between waypoints

$$J_1 = \int_{u_{min}}^{u_{max}} \|\mathbf{p}(u) - \mathbf{p}_2(u)\| du \cdot L_p^{-1} \tag{4.2}$$

ii) Average curvature of the path[2]

$$J_2 = \int_{u_{min}}^{u_{max}} \frac{\| \frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2} \|}{\| \frac{d\mathbf{p}}{du} \|^3} du \cdot L_p^{-1} \tag{4.3}$$

iii) Average acceleration of the trajectory

$$J_3 = \int_{t_{min}}^{t_{max}} \|\ddot{\tilde{\mathbf{p}}}(t)\| dt \cdot T_p^{-1} \tag{4.4}$$

iv) Deviation between trajectory and trace of the system[3]

$$J_4 = \int_{t_{min}}^{t_{max}} \|\tilde{\mathbf{p}}(t_{cl}) - \mathbf{r}(t)\| dt \cdot T_p^{-1} \tag{4.5}$$

v) Average acceleration of the system

$$J_5 = \int_{t_{min}}^{t_{max}} \|\ddot{\mathbf{r}}(t)\| dt \cdot T_p^{-1} \tag{4.6}$$

vi) Time synchronity[4]

$$J_6 = \|\tilde{\mathbf{p}}(t_{max}) - \mathbf{r}(t_{max})\| \cdot L_p^{-1} \tag{4.7}$$

4.2 Definition of Trajectories

4.2.1 Paths and Trajectories

The main difference between a path $\mathbf{p}(u)$ and a trajectory $\tilde{\mathbf{p}}(t)$ is that only the latter includes time, i.e. considers the dynamics. A path is only defined as the way to go from point a to point b. Therefore it only has geometrical properties. In order to generate a trajectory, a time needs to be assigned to each point on the path. This is done with a function $u = u(t)$ that connects the parameter u of the geometrical path with the time. In the following this function $u = u(t)$ will also be referred to as the motion law as in [3]. The composition of the $u = u(t)$ and the geometrical path $\mathbf{p}(u)$ finally forms the trajectory $\tilde{\mathbf{p}}(t)$. This concept is shown in figure 4.2.

[2]For a derivation of curvature see any vetor analysis book, e.g. [15] chapter II, page 71.

[3]The deviation vector between trace and its closest point on the trajectory is always normal to the latter. Compare with figure 4.16.

[4]Time synchronity should be warranted for accurate *trajectory* following. In SKYE's task of caputuring time independent imagery, it was only considered as a secondary aspect.

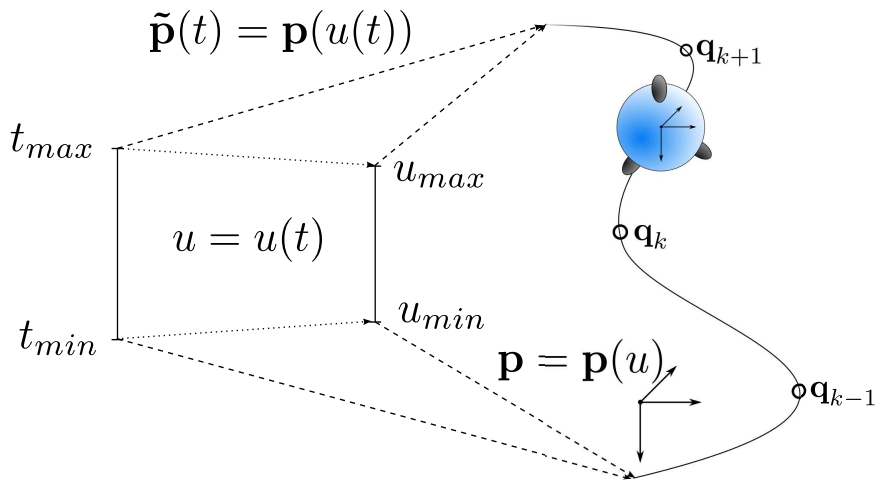


Figure 4.2: Composition of a path $\mathbf{p}(u)$ and the motion law $u(t)$ forming the trajectory $\tilde{\mathbf{p}}(t)$

In order to get the velocity and acceleration of the trajectory, the chain rule has to be applied to $\tilde{\mathbf{p}}(t) = (\mathbf{p} \circ u)(t)$:

$$\dot{\tilde{\mathbf{p}}}(t) = \frac{d\mathbf{p}}{du}\dot{u}(t) \tag{4.8}$$

$$\ddot{\tilde{\mathbf{p}}}(t) = \frac{d\mathbf{p}}{du}\ddot{u}(t) + \frac{d^2\mathbf{p}}{du^2}\dot{u}^2(t) \tag{4.9}$$

4.2.2 Interpolation and Approximation

If one wants to draw a path through a set of waypoints, we can distinguish between two cases. Firstly, the path can pass through all waypoints no matter how many bends it will have. Secondly, the path tries to best fit the waypoint set, i.e. a function of a certain order is adopted to best fit the waypoints. This can be done with different methods, e.g with least-squares. The first approach is called interpolation whereas the second approach is called approximation. Depending on the choice, different curves with different properties are formed (see figure 4.3).

Figure 4.3: Interpolation and approximation of waypoints

For the the trajectory planning for Skye, it was necessary to interpolate the waypoints. This is due to the GUI used (see section 3.3). There, only a small amount

of waypoints is set and the pilot expects SKYE to pass through all of them and not to take any shortcut. I.e.:

$$\mathbf{p}(u_k) \stackrel{!}{=} \mathbf{q}_k = \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} \tag{4.10}$$

The approximation of the waypoints could result in a collision with buildings as illustrated in 4.3.
There mainly exist two solution to interpolate a given set of $n+1$ waypoints. A polynomial (of order $\geq n+1$) from $\Pi_{>n}$ or else a set of polynomials of lower order, defined over a certain interval, can be used. This set of low-order polynomials then forms a spline. As impressively shown in [16], splines are the better option for more than a few waypoints has high order polynomials tend to high oscillations.

4.3 Splines

4.3.1 Parameterization

Before actually a path can be drawn through a set of waypoints respectively a spline can be calculated, a value of the parameter u needs to be assigned to all $n+1$ waypoints. I.e. a so called knot vector $\mathbf{u}$ composed of a nondecreasing sequence of $u_0 < u_1 < ... < u_n$ must be found in order to calculate a spline for each dimension.

$$\mathbf{p}(u) = \begin{bmatrix} p_x(u) \\ p_y(u) \\ p_z(u) \end{bmatrix} \tag{4.11}$$

In this section, the most common used techniques are analyzed[5]. The choice of parameterization affects the geometrical properties of the path on the one hand and the velocity and acceleration of the trajectory on the other hand (compare equation (4.8)). Latter is especially of big impact if a proportional relation is used to describe the motion law, i.e. $u = \lambda t$ (compare section 4.4).

Uniform

This is the simplest way to define the knot vector $\mathbf{u}$. Here, basically the index number of the waypoints are assigned to u if $c = 1$. Therefore no meaningful interpretation can be made.

$$u_k - u_{k-1} = c$$

Chord length

In this method the chord length between the waypoints is calculated and then summed up and assigned to u. With $u = \lambda t$ this method can be interpreted to aim for a more or less constant velocity over the whole trajectory.

$$u_k - u_{k-1} = \left\| \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} - \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ z_{k-1} \end{bmatrix} \right\|$$

[5] A vaster evaluation of different parameterizations can be found in [17]

Arc length

The arc length method is an improved version of the chord length distribution in order to reach a constant velocity. Here instead of taking the chord length between two waypoints, the arc length is estimated and summed up.

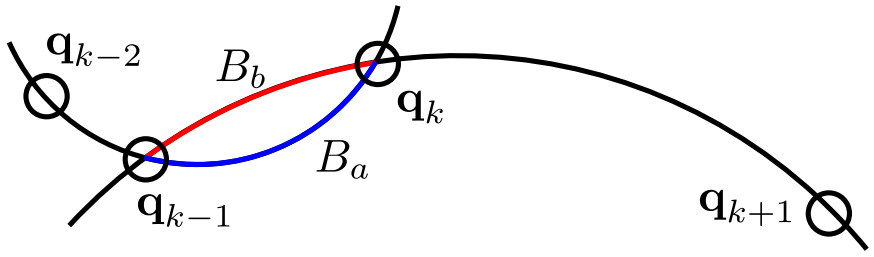


Figure 4.4: The arc length is estimated using the segments of two circles.

$$u_k - u_{k-1} = \frac{B_a + B_b}{2}$$

Centripetal

This method was first proposed in [14]. Here the root of the chord length is used. The motivation for this method is that the larger the angular change from waypoint_{n-1} to waypoint_n, the more centripetal force is accepted. A justification for this statement can be found in [7] and [14].

$$u_k - u_{k-1} = \sqrt{\left\| \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} - \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ z_{k-1} \end{bmatrix} \right\|}$$

Discussion

All those four methods were tested on our three sample waypoints defined in 4.1. Since the helix waypoints all have the same distance between each other, all four methods result in the same path, respectively trajectory. As the chord and arc length distribution try to maintain a constant velocity, high curvature will be omitted. This however leads to overshoots as seen in figure 4.5. Choosing cubic splines reduces this effect whereas quintic splines amplify it (compare figure A.1).

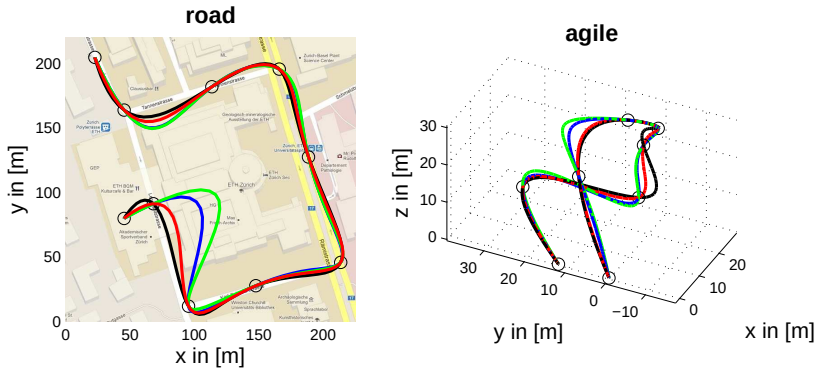


Figure 4.5: The different parameterizations shown with quartic splines; Uniform: black, Chord length: blue, Arc length: green, Centripetal: red

Since those overshoots are intolerable, a uniform or centripetal distribution has to be chosen. Although in our sample waypoints the two variants seem to be quite similar, the uniform distribution sometimes leads to loops or cusps as shown in [14] and [17]. This occurs if the chord distance between the waypoints suddenly changes a lot as the uniform distribution is completely independent of the waypoint set. Therefore the centripetal distribution was chosen to fit best our needs.
Beside the geometrical appearance due to the different distributions the dynamic properties are also of big interest, especially if a constant time scaling is used as described in section 4.4.2. Here, as the centripetal distribution is already chosen due to its geometrical properties, this is only shown for reasons of completeness. However, if the waypoints were set in a closer succession the geometrical properties would not be that important anymore and the decision should be rethought.

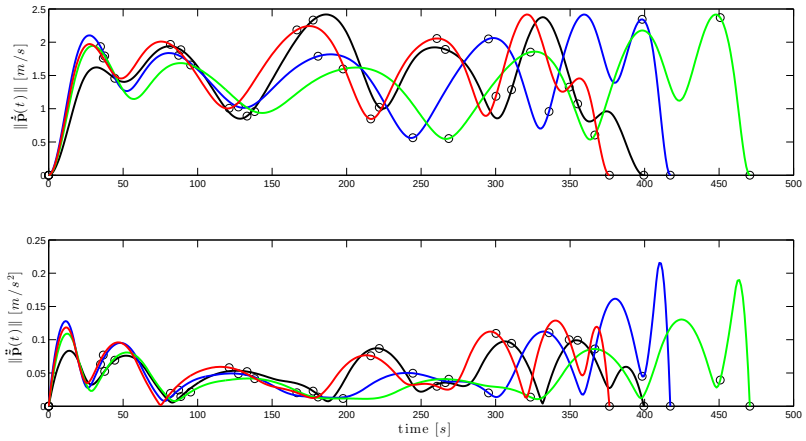


Figure 4.6: Velocity and acceleration of the *road* trajectory generated with quartic splines. In this case the motion law limits the maximum velocity whereas the acceleration remains below its limit. Parametrizations: Uniform (black), chord length (blue), arc length (green) and centripetal (red).

In figure 4.6 the effect of the different distributions on the velocity and acceleration resulting from a constant time scaling can clearly be seen with quartic splines. The chord length as well as the arc length distribution keep the velocity more or less constant but start oscillating at the end of the track. For cubic splines the situation is slightly different (compare table 4.1 and **Appendix will soon include the quartic and quintic table**) but over all criteria centripetal yields the best result.

Cubic *Road*		unit	Uniform	Chord length	Arc length	Centripetal
Av. Deviation	J_1	[m]	3.859	4.477	4.903	3.650
Av. Curvature	J_2	$[\mathrm{m}^{-1}]$	0.025	0.022	0.020	0.024
Av. Acceleration	J_3	$[\mathrm{m\,s}^{-2}]$	0.042	0.164	0.176	0.071
Time		[s]	388.5	322.2	334.4	340.3

Table 4.1: Comparison of the parametrizations. Static criteria are listed for cubic trajectories using the *road* waypoints.

4.3.2 Spline Degree

Continuity

As soon as we talk of splines the question about the spline order arises, i.e. of what degree are the polynomials which make up the spline curve. If polynomials of degree one are used then already the first derivative of the curve will not be continuous anymore. We say that this curve has a geometrical continuity of G^0. In the case of a trajectory the geometrical continuity of the path will directly have an influence on the continuity of the motion as can be seen from the equations (4.8) or in figure 4.7. E. g. with a motion law of $u = \lambda t$, cubic splines are discontinuous in the jerk whereas quartic are still continuous and quintic splines would be continuous up to the snap, i.e. $\tilde{\mathbf{p}}(t)$ would have a parametric continuity of C^4 .

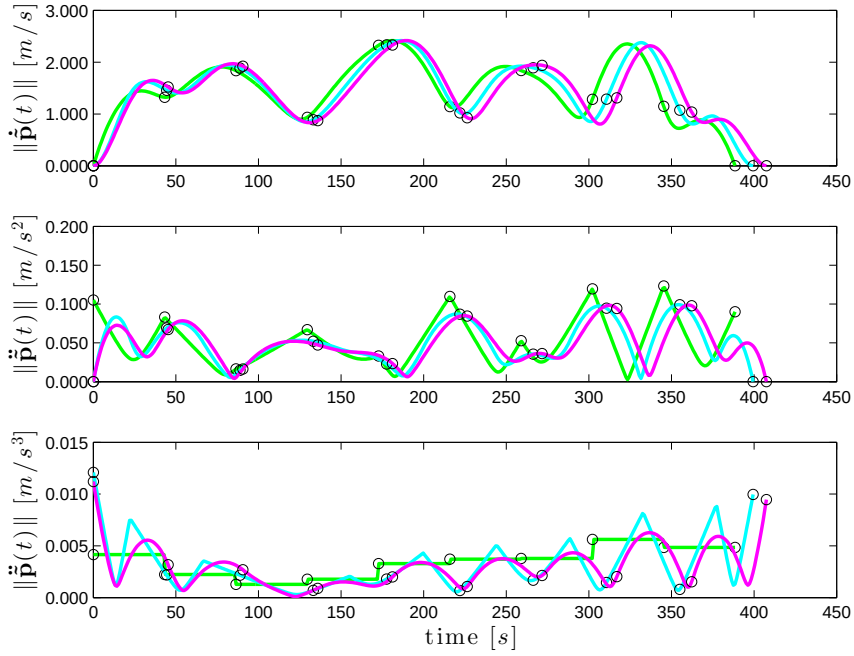


Figure 4.7: The different degrees of continuity for cubic (green), quartic (cyan) and quintic(magenta) splines.

For Skye it was intuitively clear that at least cubic splines had to be used, i.e. a parametric continuity of C^2 was needed. This is due to its propulsion system, which is briefly described in section 1.2 and in detail in [10]. The orientation of the thrusters cannot be changed immediately, i.e. the positioning motor needs time to turn the thruster in the requested direction. Therefore the trajectory should not ask for a step input for the orientation of thruster which is respected by using cubic splines or splines of higher degree. However, in order to find out whether even the dynamics of the thrusters or further dynamics of the positioning motor had to be considered, the whole propulsion system was fed with forces calculated from the accelerations $F_x = m_{tot}a_x$. As can be seen from figure 4.8 the propulsion system is able to handle all the resulting inputs of cubic, quartic and quintic splines.

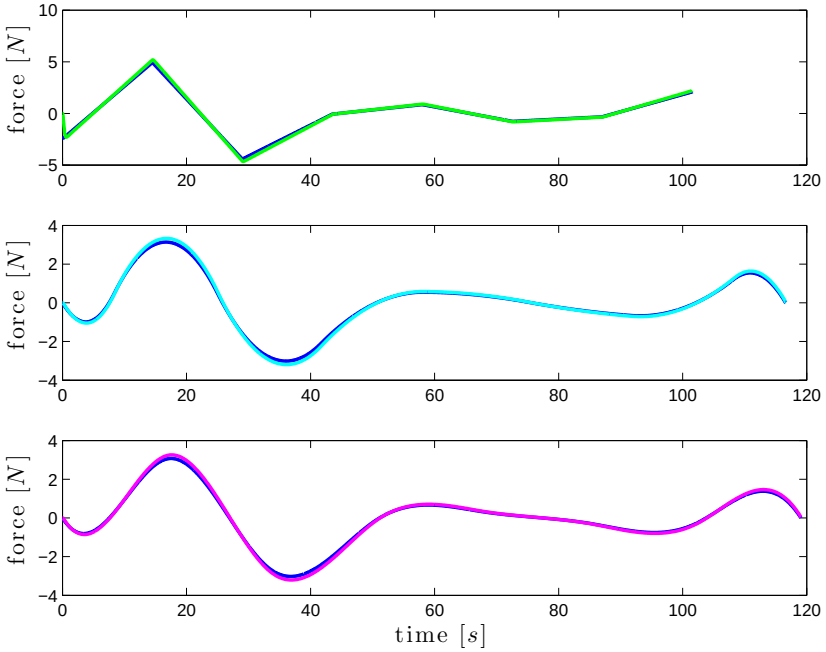


Figure 4.8: The comparison of the forces output of the propulsion dynamics with the forces input resulting from the different degrees of splines; input (blue), output: cubic (green), quartic (cyan) and quintic (magenta).

Geometrical appearance

Beside the continuity and the dynamic properties again the geometrical appearance needs to be considered in order to choose the best degree of splines for Skye. As shown in figure 4.9, the cubic splines are most suitable to use with a not so dense set of waypoints. Quartic and qubic splines have higher overshoots which would need to be omitted by setting waypoints more densely. As this would be cumbersome with the GUI described in 3, cubic splines are most suitable.

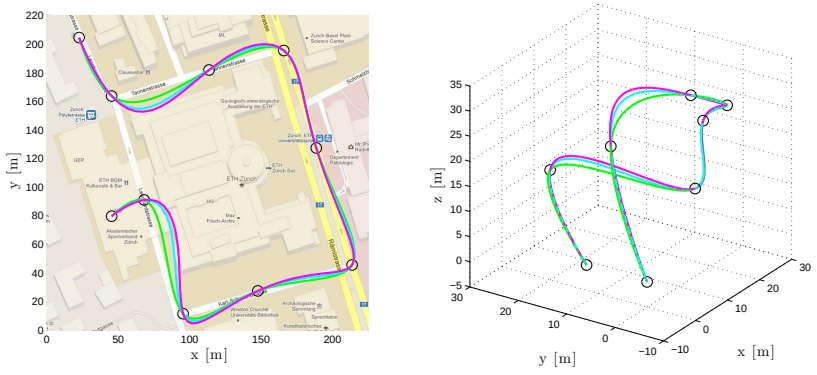


Figure 4.9: The comparison of the cubic:green, quartic:cyan and quintic:magenta splines with a centripetal parameterization of the waypoints

4.3.3 Boundary Conditions

If $p_x(u)$ is a spline of degree p and consists of $n_{knots+1}$ knots[6], it has $p + n_{knots}$ degrees of freedom. The polynomial of the first interval a has $p + 1$ degrees of freedom. The polynomial for the second interval b is given by

$$a^j(u_1) = b^j(u_1),\ j = 0, ..., p - 1$$

Therefore b will only provide one additional degree of freedom. This is also true for all the following intervals. The spline $p_x(u)$ has finally $p + n_{knots}$ degrees of freedom[7]. Using piecewise polynomials the number of the waypoints $n + 1$ is also the number of knots $n_{knots} + 1$. Given the parameter u for all the $n + 1$ waypoints yields $p + n_{knots} - (n_{knots} + 1) = p - 1$ constraints that still have to be imposed. As a constant time scaling is used[8] it is necessary to define:

$$\frac{dp_x(u)}{du}\Big|_{u=u_0} = \frac{dp_x(u)}{du}\Big|_{u=u_n} = 0$$

in order to have the velocity at the start and at the end of the trajectory to equal zero

$$\dot{\tilde{p}}_x(t_{start}) = \dot{\tilde{p}}_x(t_{end}) = 0$$

For quintic splines two additional constraints can be set

$$\frac{d^2p_x(u)}{du^2}\Big|_{u=u_0} = \frac{d^2p_x(u)}{du^2}\Big|_{u=u_n} = 0$$

such that also the acceleration at the start and at the end of the trajectory is zero (compare with equation(4.8)). For quartic splines also two additional constraints can be set if B-splines are used and the knots are shifted as described in section 4.3.4.
Those boundary conditions are set for all dimensions.

4.3.4 Implementation

As the implementation of the spline generation was not the goal of this Bsc. Thesis, it was decided to use MATLAB's Curve Fitting Toolbox. Yet it soon became evident that in order to correctly set boundary conditions for quartic and quintic splines and to get a thorough understanding, it was best to program the algorithms and only use the toolbox to make calculations with the generated splines.

While for the cubic splines the piecewise polynomial approach from [1] was chosen, B-splines were chosen as a basis for the quartic and quintic splines and the proposed C-Code in [2] was implemented in MATLAB to get the B-splines.

If $\mathbf{u}_B = [u_0, ...u_{n_{knots}}]$ is the knot vector for the B-splines, the $m = n_{knots} - p - 1$[9] B-splines of degree p for this knot vector are defined recursively using the Cox-de Boor recursion formula:

[6] Knots are the places where the polynomial pieces are connected.
[7] For a more mathematical proof refer to [16].
[8] See section 4.4.2. Also a non linear solution is described.
[9] For a derivation of this relationship, see [2] or [17].

$$B_j^0(u) = \begin{cases} 1, & \text{if } u_j \leq u < u_{j+1} \\ 0, & \text{otherwise} \end{cases} \tag{4.12}$$

$$B_j^p(u) = \frac{u - u_j}{u_{j+p} - u_j} B_j^{p-1}(u) + \frac{u_{j+p+1} - u}{u_{j+p+1} - u_{j+1}} B_{j+1}^{p-1}(u),\ p > 0 \tag{4.13}$$

A linear combination of B-splines then finally forms the path, whereas $\mathbf{c_j}$ is also referred to as control polygon:

$$\mathbf{p}(u) = \sum_{j=0}^{m} \mathbf{c_j} B_j^p(u),\ u_{min} \leq u \leq u_{max} \tag{4.14}$$

First the knot vector $\mathbf{u}$ calculated in 4.3.1 needs to be extended for the use with B-splines. For the quintic splines the following extension is used:

$$\mathbf{u}_B = [\underbrace{u_0, \ldots, u_0}_{6}, u_1, \ldots, u_{n-1}, \underbrace{u_n, \ldots, u_n}_{6}] \tag{4.15}$$

So the total number of knots is $n_{knots} + 1 = n + 11$ and therefore the number of the control polygon's points to be defined is $m + 1 = n_{knots} + 1 - 5 - 1 = n + 5$ Given $n + 1$ waypoints the 4 additional constraint on velocity and acceleration, stated in 4.3.3 are left

For the quartic splines the knot vector is shifted in order to get a better result as proposed in [1]:

$$\mathbf{u}_B = [\underbrace{u_0, \ldots, u_0}_{5}, (u_0 + u_1)/2, \ldots, (u_{k-1} - u_k)/2, \ldots, (u_{n-1} + u_n)/2, \underbrace{u_n, \ldots, u_n}_{5}] \tag{4.16}$$

Doing the same calculation as for the quintic splines here again results in $m + 1 = n + 5$ as well. This is why the mentioned four boundary conditions in 4.3.3 hold true.

Once the knot vector $\mathbf{u}_B$ is defined and the boundary conditions are set, the only thing left is to solve the following linear system of equation for all dimensions in order to get the control polygon[10], here shown for x:

$$\mathbf{A}_{[m+1 \times m+1]} \mathbf{c}_{x[m+1 \times 1]} = \mathbf{b}_{x[m+1 \times 1]} \tag{4.17}$$

where

$$\mathbf{c}_x = [c_{x0}, c_{x1}, \ldots, c_{xm-1}, c_{xm}]$$

and

$$\mathbf{A} = \begin{bmatrix} B_0^p(u_0) & B_1^p(u_0) & \ldots & B_m^p(u_0) \\ B_0^{p(1)}(u_0) & B_1^{p(1)}(u_0) & \ldots & B_m^{p(1)}(u_0) \\ B_0^{p(2)}(u_0) & B_1^{p(2)}(u_0) & \ldots & B_m^{p(2)}(u_0) \\ B_0^p(u_1) & B_1^p(u_1) & \ldots & B_m^p(u_1) \\ \vdots & \vdots & & \vdots \\ B_0^p(u_{n-1}) & B_1^p(u_{n-1}) & \ldots & B_m^p(u_{n-1}) \\ B_0^{p(2)}(u_n) & B_1^{p(2)}(u_n) & \ldots & B_m^{p(2)}(u_n) \\ B_0^{p(1)}(u_n) & B_1^{p(1)}(u_n) & \ldots & B_m^{p(1)}(u_n) \\ B_0^p(u_n) & B_1^p(u_n) & \ldots & B_m^p(u_n) \end{bmatrix}, \quad \mathbf{c_x} = \begin{bmatrix} c_{x0} \\ c_{x1} \\ \vdots \\ c_{xm-1} \\ c_{xm} \end{bmatrix}, \quad \mathbf{b_x} = \begin{bmatrix} x_0 \\ v_{x0} \\ a_{x0} \\ x_1 \\ \vdots \\ x_{n-1} \\ a_{xn} \\ v_{xn} \\ x_n \end{bmatrix}$$

[10] In fact, choosing the polygon spanned by the waypoints as a control polygon, an approximation like the one displayed in 4.3 results.

4.4 Motion Law

4.4.1 System Constraints

In order to plan a feasible trajectory one has to know the capabilities of the system. Here, just a basic derivation for maximum velocity and the maximum accelerations is given, for more details refer to [8], [10].

The maximum feasible acceleration in any direction is calculated to be:

$$|a_{max}| = \frac{\|\mathbf{F_{res,w}}\|}{m_{tot}} = 0.96\,\mathrm{m\,s^{-2}} \tag{4.18}$$

Whereas the $F_{res,w}$ is the force resulting from all four thrusters operated under full load in the worst direction and m_{tot} is the sum of the masses of the helium, the virtual mass and the mass of the system itself.

The maximum feasible velocity in any direction is calculated to be:

$$|v_{max}| = \sqrt{\frac{\|\mathbf{F_{res,w}}\|}{\frac{1}{2}c_d\rho\pi r^2}} = 2.9\,\mathrm{m\,s^{-1}} \tag{4.19}$$

which is nothing but $\|\mathbf{F}_{res,min}\| = \|\mathbf{F}_{dray}\|$.

For trajectories for position and orientation the maximal feasible angular acceleration is also important. It is calculated to be:

$$|\Psi_{max}| = \frac{\|\mathbf{M_{res,w}}\|}{|\lambda_{max,J_B}|} = 2.06\,\mathrm{rad\,s^{-2}} \tag{4.20}$$

which is quite conservative because it is assumed that the worst axis for turning is also the principle axis of the inertia tensor with the highest inertia.

Since the system is almost undamped for rotations, the rotational velocities will never be the limiting factor.

4.4.2 Motion Law

Knowing the system constraints the motion law $u = u(t)$ can be defined. There are three constraints which have to be met:

- $u(t)$ must be monotonically increasing.
- $u(t)$ must have a continuity such that the composition $\tilde{\mathbf{p}}(t) = (\mathbf{p} \circ u)(t)$ has the requested parametric continuity.
- $u(t)$ must be designed such that the trajectory $\tilde{\mathbf{p}}(t)$ meets the system constraints.

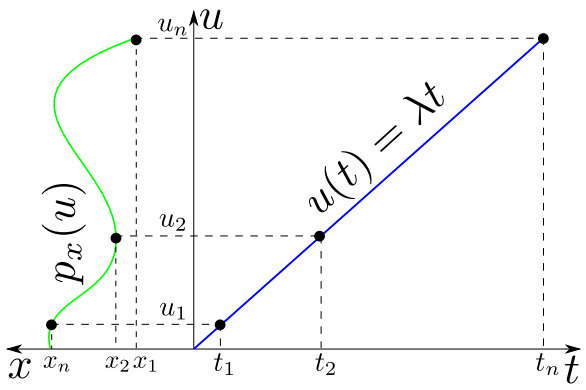


Figure 4.10: Visualization of the constant time scaling

Constant Time Scaling

The most obvious solution is the one of a constant time scaling in which a constant λ is used to scale time[11].

$$u(t) = \lambda t \tag{4.21}$$

In this case, $u(t)$ is monotonically increasing. And since $u(t)$ is continuous in the first derivative and for all higher derivative equal to zero, the continuity of $\tilde{\mathbf{p}}(t)$ only depends on on geometrical continuity of $\mathbf{p}(u)$ for the acceleration and higher derivatives of time.

$$\dot{\tilde{\mathbf{p}}}(t) = \frac{d\mathbf{p}}{du}\lambda \tag{4.22}$$

$$\ddot{\tilde{\mathbf{p}}}(t) = \frac{d^2\mathbf{p}}{du^2}\lambda^2 \tag{4.23}$$

$$\dddot{\tilde{\mathbf{p}}}(t) = \frac{d^3\mathbf{p}}{du^3}\lambda^3 \tag{4.24}$$

$$\vdots$$

The last constraint on $u(t)$ is met by choosing λ as follows:

$$\lambda = min\left\{ \frac{|v_{max}|}{\|\mathbf{p}^{(1)}(u)\|_{max}}, \quad \sqrt{\frac{|a_{max}|}{\|\mathbf{p}^{(2)}(u)\|_{max}}} \right\} / S \tag{4.25}$$

where S is a security factor, e.g. for model uncertainties.[12]
This approach of the constant time scaling is shown in figure 4.10, using an adopted visualization from [7].

A Simple Non Linear Approach

As described in 4.3.3, polynomials of at least quartic degree would be necessary to set the boundary conditions of the second derivative $\ddot{\tilde{\mathbf{p}}}(t)$ when using constant time scaling. As an alternative (not only for cubic splines), non linear time parametrization can be used. A simple approach is to replace the beginning and ending of the linear time parametrization $u(t) = \lambda t$ by a polynomial of higher degree[13].

[11] It must be noticed that this is nothing but a reparametrization of the path with parameter t.
[12] In our case S would also to be used to consider additional manual angular velocities that are superpositioned.
[13] See [7] and [2] for more advanced approaches.

We use cubic time parametrizations for the intervals u_0 to u_1 and u_{n-1} to u_n as illustrated in figure 4.11.

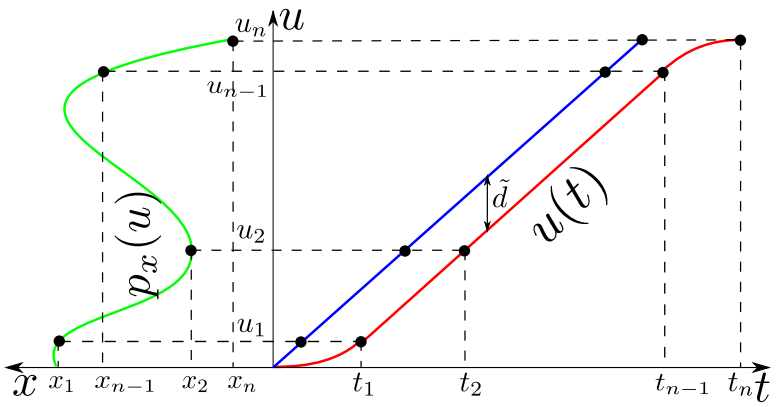


Figure 4.11: Visualization of the constant time scaling with adopted boundaries.

Like that $\ddot{\tilde{\mathbf{p}}}(t_0)$ and $\ddot{\tilde{\mathbf{p}}}(t_n)$ can be set to zero if

$$\dot{u}(t_0) = \dot{u}(t_n) = \frac{d\mathbf{p}}{du}\bigg|_{u=u_0} = \frac{d\mathbf{p}}{du}\bigg|_{u=u_n} = 0$$

With these conditions, $u(t_0) = u_0$, $u(t_n) = u_n$ and the equations resulting from continuous boundary conditions, the following motion law is obtained by solving the nonlinear set of equations[14]

$$u(t) = \begin{cases} a_0 t^3 + b_0 t^2 + c_0 t + d_0 & 0 < t \leq t_1 \\ \lambda t + \tilde{d} & t_1 < t \leq t_{n-1} \\ a_{n-1} t^3 + b_{n-1} t^2 + c_{n-1} t + d_{n-1} & t_{n-1} < t \leq t_n \end{cases} \tag{4.26}$$

whereas

$$\begin{aligned}
a_0 &= \frac{\lambda}{9t_1^2}, \quad b_0 = \frac{\lambda}{3t_1}, \quad c_0 = 0, \quad d_0 = 0, \quad \tilde{d} = -\frac{5}{9}\lambda t_1, \\
a_{n-1} &= -\frac{4\lambda^3}{27(\tilde{d} + \lambda t_{n-1} - u_n)^2}, \quad b_{n-1} = \frac{4\lambda^3 t_{n-1}}{9(\tilde{d} + \lambda t_{n-1} - u_n)^2}, \\
c_{n-1} &= \frac{\lambda(9\tilde{d}^2 + 18\lambda\tilde{d}t_{n-1} + 5\lambda^2 t_{n-1}^2 - 18\tilde{d}u_n - 18\lambda t_{n-1}u_n + 9u_n^2)}{9(\tilde{d} + \lambda t_{n-1} - u_n)^2}, \\
d_{n-1} &= \frac{27\tilde{d}^3 + 54\lambda\tilde{d}^2 t_{n-1} + 27\lambda^2\tilde{d}t_{n-1}^2 + 4\lambda^3 t_{n-1}^3 - 54\tilde{d}^2 u_n - 54\lambda\tilde{d}t_{n-1}u_n + 27\tilde{d}u_n^2}{27(\tilde{d} + \lambda t_{n-1} - u_n)^2}, \\
t_n &= \frac{-3\tilde{d} - \lambda t_{n-1} + 3u_n}{2\lambda}
\end{aligned}$$

[14] E.g. using Wolfram Mathematica.

4.5 Controller Implementation

A trajectory controller and two commonly used path tracking controllers[15] are tested to follow the defined trajectories. The *trajectory following* controller supplies the system's position controller, described in [9], with a feed forward reference signal. Although it delivers good results for ideal case, the tracking gets worse for the non perfect model case. The *pure pursuit* controller, which is based on a lookahead point as well as the *cross track error* controller dynamically react on model uncertainties and yield therefore more robust and accurate path tracking results.

4.5.1 Trajectory following

Assuming a perfect model and a trajectory considering all system constraints[16] the position $r\,(t)$ of the system can be assumed to be equal to the trajectory $\tilde{p}(t)$ at any time. Therefore, a straight forward way of a trajectory controller is to follow the trajectory $\tilde{p}(t)$ for every time t. This yields accurate tracking in a safe environment [7].
A position PI controller with feedforward terms for velocity and acceleration can therefore be used with the reference input

$$[r_{ref}(t),\ \dot{r}_{ref}(t),\ \ddot{r}_{ref}(t)]^T = [\tilde{p}(t),\ \dot{\tilde{p}}(t),\ \ddot{\tilde{p}}(t)]^T \tag{4.27}$$

The controller scheme is shown in figure 4.13.

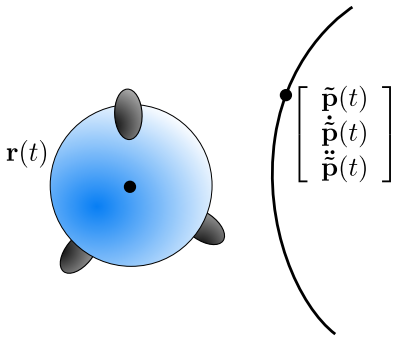


Figure 4.12: Trajectory Following. The reference state is predefined by the trajectory for every time t.

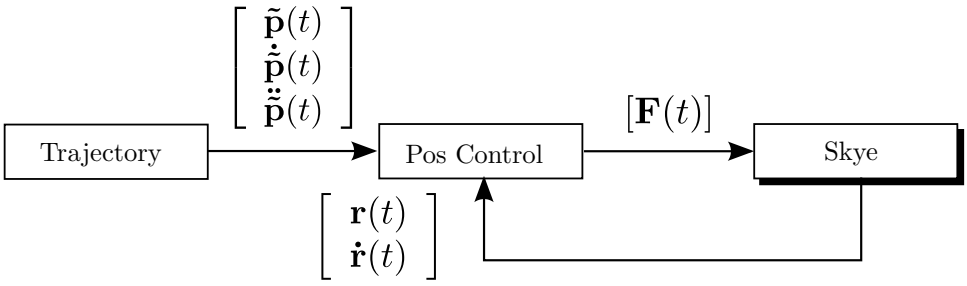


Figure 4.13: Trajectory following controller. The value of the parameter t of the trajectory is equal to the current time.

[15] [3] provides a good overview over various approaches.
[16] I.e. saturations of $\dot{r}(t)$ and its derivatives, see section 4.4.1.

4.5.2 Pure Pursuit Controller

Another commonly used controller to track paths is *Pure Pursuit* [3]. This technique is mainly used for path tracking while having a constant tangential velocity. The idea of using a lookahead point was adapted for trajectory control. To do that, the lookahead point $\tilde{\mathbf{p}}(t_{cl}+\Delta T)$ is set on the trajectory in front[17] of the current closest position $\tilde{\mathbf{p}}(t_{cl})$.

$$\tilde{\mathbf{p}}(t_{cl}+\Delta T) = \tilde{\mathbf{p}}(t_{cl}) + \Delta T \cdot \dot{\tilde{\mathbf{p}}}(t_{cl}) + \frac{1}{2}\Delta T^2 \cdot \ddot{\tilde{\mathbf{p}}}(t_{cl}) + \mathcal{O}(\Delta T^3) \tag{4.28}$$

Therefore, the controller considers the full dynamics of the trajectory, i.e. $\tilde{\mathbf{p}}(t)$ and all its derivatives[18]. Figure 4.14 represents the controllers functionality. The control scheme can be seen in figure 4.15. Actually, the inner control block is a velocity controller only. The reference velocity is given by

$$\dot{\mathbf{r}}_{ref}(t) = \frac{\tilde{\mathbf{p}}(t_{cl}+\Delta T) - \mathbf{r}(t)}{\Delta T} \tag{4.29}$$

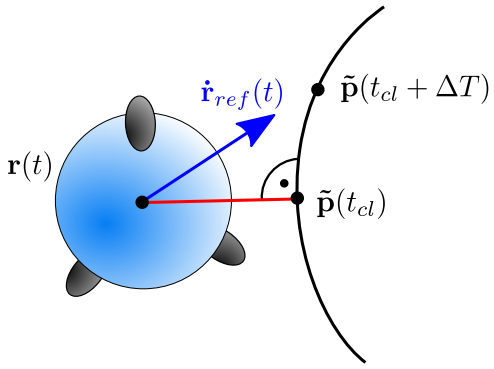


Figure 4.14: Pure pursuit. The reference velocity is given by the difference between the actual position and a lookahead point defined by the trajectory's closest point shifted by ΔT ahead.

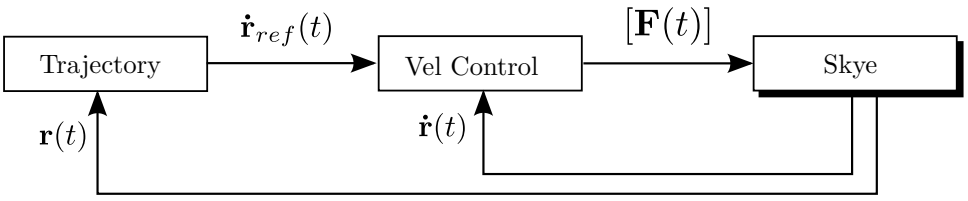


Figure 4.15: Pure pursuit controller. The position feedback is closed by defining the lookahead point depending on the closest point of the trajectory.

4.5.3 Cross Track Error Controller

The idea of the *cross track error* controller is to directly reduce the cross track distance. This practise is also used in navy [5] or for wheeled robots [4]. The actual operation is very similar to the one in *trajectory following*, but instead of considering the trajectory's current time, here, the reference input time is defined by the closest point of the trajectory. Afterwards, the same position controller as

[17] $\Delta T = 1$ second proofed to be reasonable for all considered conditions.
[18] This is a simple and robust alternative to any derivative controller.

mentioned above is used to track the reference state. Figure 4.16 illustrates the choise of the reference input for the controller. The control scheme is shown in figure 4.17.

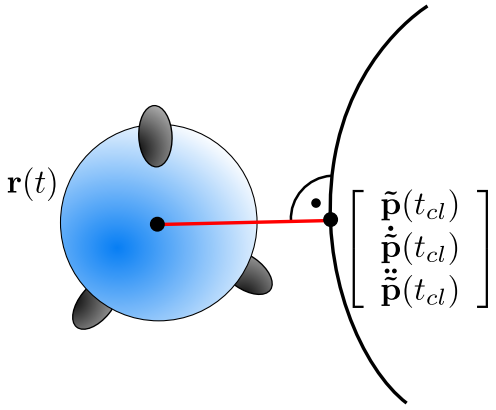


Figure 4.16: Cross track error. The reference point is the nearest point on the trajectory.

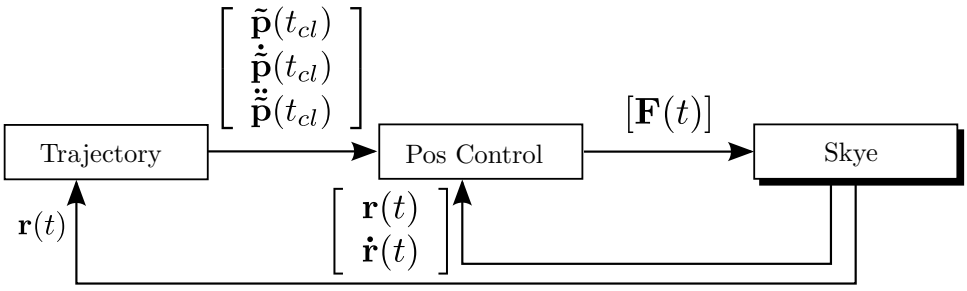


Figure 4.17: Cross track error controller. The value of the trajectories time stamp t_{cl} of the closest point is generally not equal the current time t.

4.6 Results

A model of system Skye has been elaborated in [8] and was available as Matlab Simulink model to test the controller performance. The controllers described above were tested within the experimental design notified in section 4.1. If not mentioned explicitly, the shown trajectories are designed using cubic splines and centripetal parametrization. The results are presented for perfect model estimation as well as wind disturbance and model uncertainties. Furthermore the correlation between different parameters is exposed. Further results can be found in the appendix (A.2). The occurring accelerations as well as the percentage of reached trajectory the remain nearly the same for all controllers.

4.6.1 Perfect Model Estimation

The three trajectory controllers were tested with the given waypoint samples. Figure A.2 shows their tracking. The performance difference between them is minimal as it can be seen from the results listed in table 4.2. In comparison to the systems characteristic size (diameter 2.7 m) the deviations between trajectory and trace are

extremely small, i.e. less than 1% of the characteristic size. The average acceleration is

Controller		unit	*Road*	*Helix*	*Agile*
Trajectory Following	Av. Dev.	[m]	0.007	0.017	0.014
Pure Pursuit	Av. Dev.	[m]	0.007	0.009	0.017
Cross Track	Av. Dev.	[m]	0.007	0.007	0.006
Trajectory Following	Av. Acc.	$[\mathrm{m\,s^{-2}}]$	0.029	0.174	0.137
Pure Pursuit	Av. Acc.	$[\mathrm{m\,s^{-2}}]$	0.031	0.159	0.128
Cross Track	Av. Acc.	$[\mathrm{m\,s^{-2}}]$	0.029	0.159	0.126
Trajectory Following	Reached	[%]	100	100	100
Pure Pursuit	Reached	[%]	99	97	98
Cross Track	Reached	[%]	97	97	97

Table 4.2: Results of the three controllers using perfect model estimation

4.6.2 Wind Disturbance

The same simulations have been conducted with wind disturbance. Wind forces based on a constant wind velocity of $1\,\mathrm{m\,s^{-1}}$ in positive x-direction have been added to the scenes. The disturbance is added 10 seconds after the simulation's start.
Figure A.3 shows the tracking reached by the three controllers. The resulting performance values are listed in table 4.3. While *pure pursuit* and *cross track error* control still result in small deviation[19] the deviation has grown to a considerable big scale for *trajectory following*. *Cross track error* control slightly scores better than *pure pursuit*. Again, the accelerations are similar for all controllers. While *trajectory following* reaches the end of the trajectory within the given time T_p by definition, the path tracking based trajectory controllers will remain at 86 % to 96 % of the trajectories length L_p at the time T_p.
Compared to the results of perfect modelling, the deviations of the accurate controllers get worse 5 to 10 times and for *trajectory following* by factor 10 to 200. The accelerations are comparable to the ideal case.

Controller		unit	*Road*	*Helix*	*Agile*
Trajectory Following	Av. Dev.	[m]	1.783	0.606	0.111
Pure Pursuit	Av. Dev.	[m]	0.079	0.098	0.037
Cross Track	Av. Dev.	[m]	0.034	0.061	0.023
Trajectory Following	Av. Acc.	$[\mathrm{m\,s^{-2}}]$	0.031	0.191	0.138
Pure Pursuit	Av. Acc.	$[\mathrm{m\,s^{-2}}]$	0.031	0.136	0.121
Cross Track	Av. Acc.	$[\mathrm{m\,s^{-2}}]$	0.028	0.140	0.120
Trajectory Following	Reached	[%]	100	100	100
Pure Pursuit	Reached	[%]	96	86	96
Cross Track	Reached	[%]	94	89	95

Table 4.3: Results of the three controllers with wind disturbance

[19] They are still about factor 20 smaller than the characteristic size of Skye.

4.6.3 Model Uncertainties

The simulations were repeated by designing the trajectories by system constraints with a safety factor reduced to less than one, i.e. to 0.8. Therefore, the trajectory overestimates the system's actuation possibilities.
The tracking of all controllers is shown in figure A.4. The numerical results are listed in table 4.4. The deviation of *trajectory following* control is considerable larger than of the two other ones. *Pure pursuit* even tops the *cross track error* results in this case. The average accelerations are slightly higher when using *trajectory following*. Contrary to the *trajectory following* which heads to the actual end of the trajectory by T_p, *pure pursuit* and *cross track error* will lead to 86 % to 93 % of the circuit's length.
In comparison to the ideal case, the deviations of the accurate controllers get worse 2 to 10 times. *Trajectory following* leads to deviations 50 to 400 times larger than for perfect modelling. Average accelerations grew by factors less than 2.

Controller		unit	*Road*	*Helix*	*Agile*
Trajectory Following	Av. Dev.	[m]	3.228	1.628	0.838
Pure Pursuit	Av. Dev.	[m]	0.049	0.026	0.036
Cross Track	Av. Dev	[m]	0.092	0.044	0.058
Trajectory Following	Av. Acc.	$[\mathrm{m\,s^{-2}}]$	0.066	0.234	0.234
Pure Pursuit	Av. Acc.	$[\mathrm{m\,s^{-2}}]$	0.045	0.210	0.182
Cross Track	Av. Acc.	$[\mathrm{m\,s^{-2}}]$	0.047	0.220	0.192
Trajectory Following	Reached	[%]	100	100	100
Pure Pursuit	Reached	[%]	86	90	86
Cross Track	Reached	[%]	87	93	89

Table 4.4: Results of the three controllers with underestimated trajectory constraints

4.6.4 Data Correlation

Within the gathered data of the tracking with ideal conditions (section 4.6.1) and considering all introduced parametrizations and spline degrees from section 4.3 a data set of 108 items was analyzed for correlation[20]. Some correlating results could be observed.
Figure 4.18 shows the relation between the given time T_p for the trajectory and the average acceleration J_5 occurring on the system. The result is trivial, as the less time is given to follow the waypoints, the higher gets the acceleration. The dependence on the used parametrization is different for the different waypoints, whereas higher spline degrees (quartic and quintic) yield to slightly smaller time T_p and acceleration J_5 than lower degree (cubic) consequently.
In figure 4.19 the correlation between the average deviation J_4 and the average acceleration J_5 occurring on the system are shown for the *road* waypoints[21]. As expected, higher accelerations correlates with higher deviations. Cubic splines yield to worse performance than quartic and quintic that are nearly the same.
Note that cubic splines are restricted to 4 boundary conditions and therefore start and end with accelerations unequal zero.

[20] The additional cases with non perfect modelling ($3 \cdot 108$ items with wind and/or model uncertainties) will not be shown. Indeed, the correlations cannot be seen there explicitly.

[21] Figure A.5 includes all waypoint samples. The correlations are similar but much more disturbed.

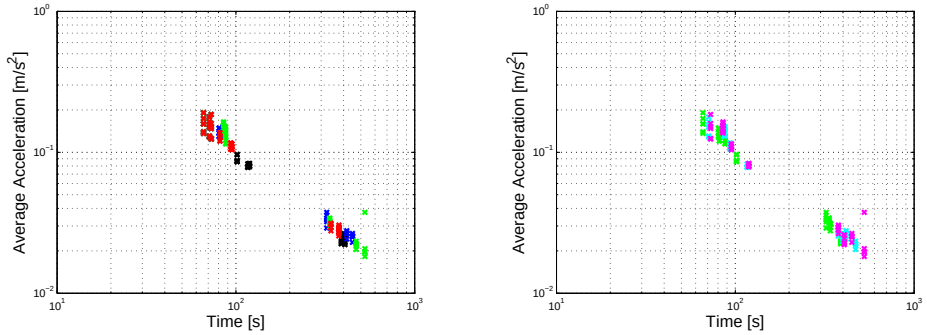


Figure 4.18: Correlation between trajectory time length T_p and average accelerations on trace. On the double logarithmic plot the correlation becomes clear. **Left**: Different waypoint parametrizations uniform (black), chord length (blue), arc length (green) and centripetal (red). **Right**: Different spline degrees cubic (green), quartic (cyan) and quintic (magenta).

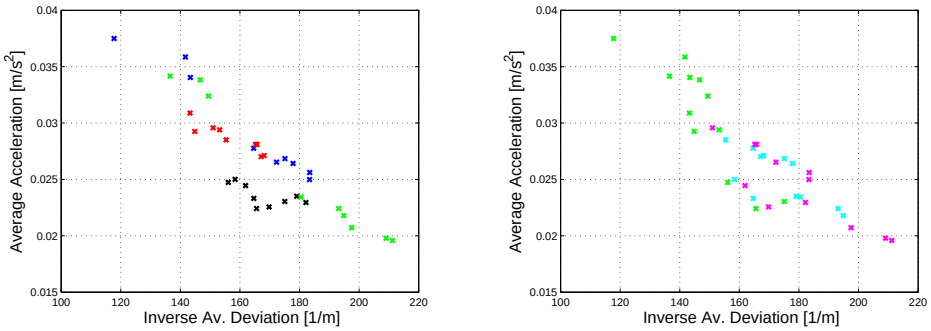


Figure 4.19: Correlation between deviation between trajectory and trace. Data points from *road* waypoints. **Left**: Different waypoint parametrizations uniform (black), chord length (blue), arc length (green) and centripetal (red). **Right**: Different spline degrees cubic (green), quartic (cyan) and quintic (magenta).

4.7 Discussion

The almost endless field of trajectory control represents the manifold needs for trajectories in different application fields. Even to control UAVs no throughout good or bad trajectory algorithms can be named. Theirs quality rather has to be rethought for any specific use. Therefore, as for every optimization problem, it is fundamental to properly define rating criteria before thinking of any trajectories.
As for our case tight waypoint tracking was considered more than for instance time minimization or smoothness, cubic splines based on centripetal parametrized waypoints result in good performance for most considerable waypoint samples. For sake of simplicity, a linear time parametrization considering the systems actuation saturations yield sufficient results.
Dependent on the choice how the generate the trajectories, proper controllers influence the actual performance of the system in case of non perfect modelling. As the presented trajectory algorithms cannot consider e.g. current environmental conditions as wind it is recommended to use time-variant trajectory controllers[22] to maintain best possible trajectory tracking.

[22] We do not use the name *path tracking* as using time parametrized trajectories allow to strictly obtain system constraints. Path tracking is mainly restricted to non agile vehicles cruising at constant speed as boats [5], planes or wheeled systems [4], [3].

Chapter 5

Conclusion

In this Thesis, a HMI, tailored to the needs of Skye has been developed. This ranges from defining control modes for six DOF, planing feasible trajectories to implementing these ideas to form a complete HMI. As for the trajectory planning existing solutions were adopted, the combination of touchscreen with a 3D mouse to control Skye is a new, so far unseen approach. However, in order to realize all ideas for an intuitive HMI, a lot of challenges had to be met.

Challenges Working on a real project as a team means also finding interfaces for the different domains of the project as well as finding interfaces for the workload the different team members are taking. For the HMI which is almost connected to everything of the project, this meant that the options for choosing the hardware and software were quite limited. Nevertheless a compact and convenient solution was found in discussions with the control and communication section of team Skye. Taking existing solution brings with it, that these solutions also have to be understood thoroughly. Otherwise it is impossible to adopt it reasonably. E.g. the source Code of QGroundControl had to be analyzed and this proved to be not that easy as it was someone else's code. However, as time passes by, you get used to it. Developing a HMI for a system which itself is not yet fully developed, was also a challenge. Numerous days were invested in other domains of the project, in order the make the whole system run and especially to make the HMI working properly with it. Especially the integration of an alpa version FMU required thousand of tests till it was running.

Maturity of the Solutions Due to a not completely mature prototype and due to lack of time, not all ideas could be tested on the real system. While the finished test phase, direct and assisted control modes proved their functionality on the real system, the half and full automatic control modes still need to be tested. However, there is already a simulation of the half automatic control mode running on the GUI and it is assumed that only the controller and the state estimation would have to be adopted to make it work. This is not like the situation of the full automatic control mode. It is still far from maturity as only a Matlab simulation of it exists.

Open Ideas As mentioned above is the full automatic control not yet elaborated in detail. However, it is supposed that a complete automatic mode would boost the application field of Skye and therefore it is suggested to realize this mode in a further step of project Skye.
Right now the generated trajectories are only optimized to not exceed the system abilities. In a further step they could be optimized for time or energy consumption.

Although a widget for the control of the onboard cameras was developed, it was never put in action. The same applies to a battery widget which would constantly display the battery level of all three accumulators. Having these widgets work would extend the pilot's control over Skye.

Looking back on the developed HMI and the source code of the software, a lot of things would be done in a more effective and easier way. Yet this is not surprising as Lorenz Meier, a skilled programmer who developed QGroundControl, told us: "You know what I did once I had finished the source code of QGroundControl? - I wrote it all again." However, the general concept of the HMI with tablet, 3D mouse and trajectory planning proved to be the right choice.

Appendix A

Trajectories

A.1 Parameterizations

A.1.1 Arc Length Distribution

TO DO

A.1.2 Effect of different Parameterizations with cubic, quartic and quintic splines

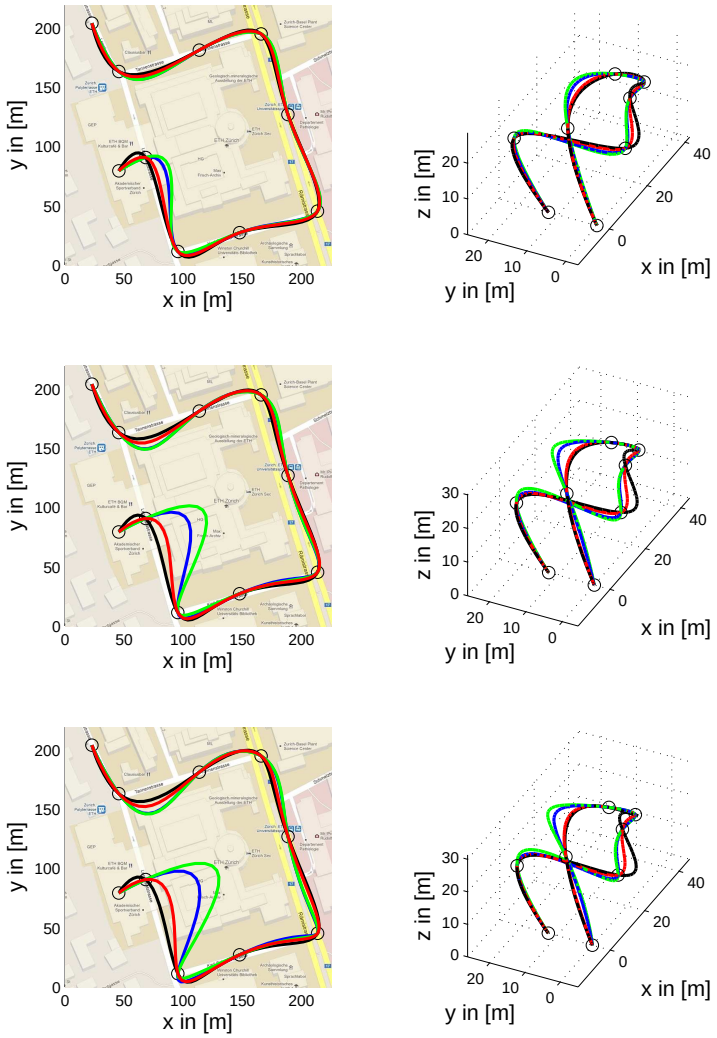


Figure A.1: The different parameterizations shown with cubic, quartic and quintic splines

A.2 Control Results

A.2.1 Varying trajectory constraints

Repetitively simulations with different scaled time parametrization (i.e. varying system constraint's safety factor S from 0.8 to 1.6) were conducted[1]. Figure A.2.1

[1] *Agile* waypoints, centripetal parametrization, cubic splines, cross track error controller, no wind disturbance.

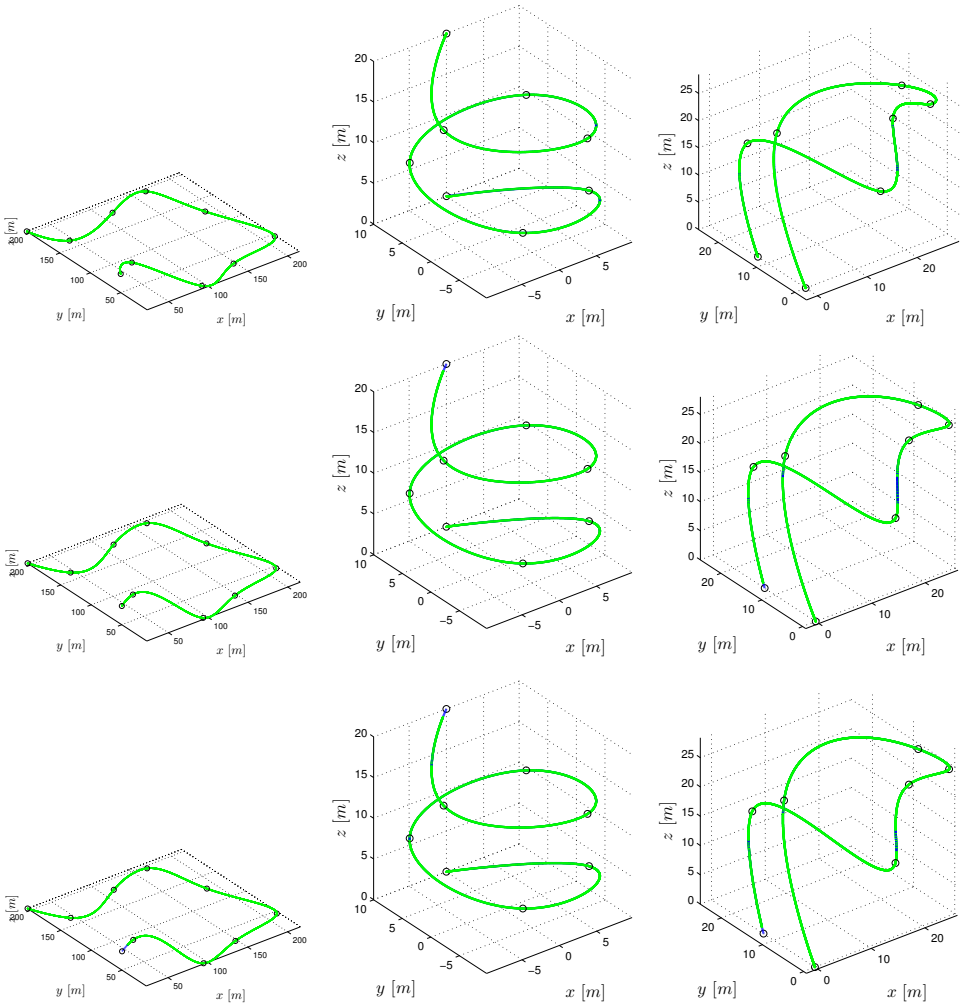


Figure A.2: Trajectory tracking with perfect model estimation. **Top**: *Trajectory following* control **Center**: *Pure pursuit* control **Bottom**: *Cross track error* control

illustrates the different trajectories and the corresponding traces.

A.2.2 Increasing the paths curvature

And here follows a subsection with constraint comparisons of a helix with smaller radius.

```
<?xml version='1.0'?>
<mavlink>
    <include>common.xml</include>
    <enums>
        <enum name="MAV_SKYE_MODE">
            <description>These defines are predefined OR-combined mode↩
                 flags used for project skye. There is no need to use ↩
                values from this enum, but it simplifies the use of ↩
                the mode flags. Note that manual input is enabled in ↩
                all modes as a safety override.</description>
            <entry value="67" name="MAV_MODE_TESTPHASE_DISARMED">
                <description>System is ready to test the motors.</↩
                    description>
            </entry>
```

```
            <entry value="195" name="MAV_MODE_TESTPHASE_ARMED">
                <description>System is ready to test the motors.</↩
                    description>
            </entry>
            <entry value="65" name="MAV_MODE_DIRECT_CONTROL_DISARMED">
                <description>System is under Direct Control, no ↩
                    stabilization.</description>
            </entry>
            <entry value="193" name="MAV_MODE_DIRECT_CONTROL_ARMED">
                <description>System is under Direct Control, no ↩
                    stabilization.</description>
            </entry>
            <entry value="81" name="MAV_MODE_ASSISTED_CONTROL_DISARMED↩
                ">
                <description>System is under Assisted Control, ↩
                    stabalized.</description>
            </entry>
            <entry value="209" name="MAV_MODE_ASSISTED_CONTROL_ARMED">
                <description>System is under Assisted Control, ↩
                    stabalized.</description>
            </entry>
            <entry value="89" name="MAV_MODE_HALF_AUTOMATIC_DISARMED">
                <description>System is under Half Automatic Control, ↩
                    translation by waypoints, rotation by manual ↩
                    input. Stabalized.</description>
            </entry>
            <entry value="217" name="MAV_MODE_HALF_AUTOMATIC_ARMED">
                <description>System is under Half Automatic Control, ↩
                    translation by waypoints, rotation by manual ↩
                    input. Stabalized.</description>
            </entry>
            <entry value="93" name="MAV_MODE_FULL_AUTOMATIC_DISARMED">
                <description>System is under Full Automatic Control, ↩
                    steering by waypoints only. Stabalized.</↩
                    description>
            </entry>
            <entry value="221" name="MAV_MODE_FULL_AUTOMATIC_ARMED">
                <description>System is under Full Automatic Control, ↩
                    steering by waypoints only. Stabalized.</↩
                    description>
            </entry>
        </enum>
        <enum name="MAV_CAM_RECONFIG_PIXEL_CLOCK">
            <description>Camera Reconfigure parameter "color_coding"</↩
                description>
            <entry value="6" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_6M">
                <description/>
            </entry>
            <entry value="8" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_8M">
                <description/>
            </entry>
            <entry value="10" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_10M">
                <description/>
            </entry>
            <entry value="13" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_13M5"↩
                >
                <description/>
            </entry>
            <entry value="20" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_20M">
                <description/>
            </entry>
            <entry value="24" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_24M">
                <description/>
            </entry>
            <entry value="27" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_27M6"↩
                >
                <description/>
            </entry>
            <entry value="32" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_32M">
                <description/>
            </entry>
            <entry value="37" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_37M">
                <description/>
            </entry>
            <entry value="40" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_40M">
                <description/>
            </entry>
            <entry value="50" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_50M">
                <description/>
            </entry>
```

```
            <entry value="57" name="MAV_CAM_RECONFIG_PIXEL_CLOCK_57M6">
                <description/>
            </entry>
        </enum>
        <enum name="MAV_CAM_RECONFIG_COLOR_CODING">
            <description>Camera Reconfigure parameter "color_coding"</description>
            <entry value="0" name="MAV_CAM_RECONFIG_COLOR_CODING_AUTO">
                <description/>
            </entry>
            <entry value="1" name="MAV_CAM_RECONFIG_COLOR_CODING_MONO8">
                <description/>
            </entry>
            <entry value="2" name="MAV_CAM_RECONFIG_COLOR_CODING_MONO16">
                <description/>
            </entry>
            <entry value="3" name="MAV_CAM_RECONFIG_COLOR_CODING_RAW8">
                <description/>
            </entry>
            <entry value="4" name="MAV_CAM_RECONFIG_COLOR_CODING_BGR8">
                <description/>
            </entry>
            <entry value="5" name="MAV_CAM_RECONFIG_COLOR_CODING_BGRA8">
                <description/>
            </entry>
            <entry value="6" name="MAV_CAM_RECONFIG_COLOR_CODING_BGR16">
                <description/>
            </entry>
        </enum>
        <enum name="MAV_CAM_RECONFIG_BAYER_METHOD">
            <description>Camera Reconfigure parameter "bayer_method"</description>
            <entry value="0" name="MAV_CAM_RECONFIG_BAYER_METHOD_IMAGE_PROC">
                <description/>Decode via ROS image_proc</entry>
            <entry value="1" name="MAV_CAM_RECONFIG_BAYER_METHOD_DOWNSAMPLE">
                <description/>DownSample</entry>
            <entry value="2" name="MAV_CAM_RECONFIG_BAYER_METHOD_SIMPLE">
                <description/>Simple</entry>
            <entry value="3" name="MAV_CAM_RECONFIG_BAYER_METHOD_BILINEAR">
                <description/>Bilinear</entry>
            <entry value="4" name="MAV_CAM_RECONFIG_BAYER_METHOD_HQ">
                <description/>HQ</entry>
            <entry value="5" name="MAV_CAM_RECONFIG_BAYER_METHOD_VNG">
                <description/>VNG</entry>
            <entry value="6" name="MAV_CAM_RECONFIG_BAYER_METHOD_AHD">
                <description/>AHD</entry>
        </enum>
        <enum name="MAV_CAM_RECONFIG_AUTO_CONTROL_SPEED">
            <description>Camera Reconfigure parameter "auto_control_speed"</description>
            <entry value="0" name="MAV_CAM_RECONFIG_AUTO_CONTROL_SPEED_ACS_MEDIUM">
                <description>Medium</description>
            </entry>
            <entry value="1" name="MAV_CAM_RECONFIG_AUTO_CONTROL_SPEED_ACS_SLOW">
                <description>Slow</description>
            </entry>
            <entry value="2" name="MAV_CAM_RECONFIG_AUTO_CONTROL_SPEED_ACS_FAST">
                <description>Fast</description>
            </entry>
        </enum>
        <enum name="MAV_CAM_RECONFIG_HDR_MODE">
            <description>Camera Reconfigure parameter "hdr_mode"</description>
            <entry value="0" name="MAV_CAM_RECONFIG_AUTO_CONTROL_HDR_MODE_HDR_OFF">
```

```
                    <description>Off</description>
                </entry>
                <entry value="1" name="↩
                    MAV_CAM_RECONFIG_AUTO_CONTROL_HDR_MODE_HDR_FIXED0">
                    <description>Fixed0</description>
                </entry>
                <entry value="2" name="↩
                    MAV_CAM_RECONFIG_AUTO_CONTROL_HDR_MODE_HDR_FIXED1">
                    <description>Fixed1</description>
                </entry>
                <entry value="3" name="↩
                    MAV_CAM_RECONFIG_AUTO_CONTROL_HDR_MODE_HDR_FIXED2">
                    <description>Fixed2</description>
                </entry>
                <entry value="4" name="↩
                    MAV_CAM_RECONFIG_AUTO_CONTROL_HDR_MODE_HDR_FIXED3">
                    <description>Fixed3</description>
                </entry>
                <entry value="5" name="↩
                    MAV_CAM_RECONFIG_AUTO_CONTROL_HDR_MODE_HDR_FIXED4">
                    <description>Fixed4</description>
                </entry>
                <entry value="6" name="↩
                    MAV_CAM_RECONFIG_AUTO_CONTROL_HDR_MODE_HDR_FIXED5">
                    <description>Fixed5</description>
                </entry>
                <entry value="7" name="↩
                    MAV_CAM_RECONFIG_AUTO_CONTROL_HDR_MODE_HDR_FIXED6">
                    <description>Fixed6</description>
                </entry>
                <entry value="8" name="↩
                    MAV_CAM_RECONFIG_AUTO_CONTROL_HDR_MODE_HDR_USER">
                    <description>User</description>
                </entry>
            </enum>
            <enum name="MAV_CAM_ID">
                <description>Camera ID</description>
                <entry value="0" name="MAV_CAM_ID_ALL">
                    <description>Off</description>
                </entry>
                <entry value="1" name="MAV_CAM_ID_PROSILICA">
                    <description>High resolution AVT prosilica camera</↩
                        description>
                </entry>
                <entry value="2" name="MAV_CAM_ID_BLUEFOX_LEFT">
                    <description>Low resolution matrix-vision bluefox ↩
                        camera, position left</description>
                </entry>
                <entry value="3" name="MAV_CAM_ID_BLUEFOX_RIGHT">
                    <description>Low resolution matrix-vision bluefox ↩
                        camera, position right</description>
                </entry>
            </enum>
            <enum name="MAV_CAM_IMAGE_FORMAT">
                <description>Image Format</description>
                <entry value="0" name="MAV_CAM_IMAGE_FORMAT_RAW">
                    <description>RAW format</description>
                </entry>
                <entry value="1" name="MAV_CAM_IMAGE_FORMAT_JPEG">
                    <description>JPEG format</description>
                </entry>
                <entry value="2" name="MAV_CAM_IMAGE_FORMAT_PNG">
                    <description>PNG format</description>
                </entry>
              </enum>
<!-- Copied DATA_TYPES from pixhawk.xml -->
            <enum name="DATA_TYPES">
                <description>Content Types for data transmission handshake↩
                    </description>
                <entry value="1" name="DATA_TYPE_JPEG_IMAGE" />
                <entry value="2" name="DATA_TYPE_RAW_IMAGE" />
                <entry value="3" name="DATA_TYPE_KINECT" />
            </enum>
            <enum name="MAV_SKYE_BATTERY_PACK_ID">
                <description>ID for each accu pack for detailed battery ↩
                    information</description>
                <entry value="0" name="MAV_SKYE_BATTERY_PACK_ID_NONE">
                    <description>no accu</description>
                </entry>
                <entry value="1" name="MAV_SKYE_BATTERY_PACK_ID_1">
```

```
                <description>Accu pack 1</description>
            </entry>
            <entry value="2" name="MAV_SKYE_BATTERY_PACK_ID_2">
                <description>Accu pack 1</description>
            </entry>
            <entry value="3" name="MAV_SKYE_BATTERY_PACK_ID_3">
                <description>Accu pack 1</description>
            </entry>
        </enum>
    </enums>
    <messages>
        <message id="152" name="SKYE_BATTERY_STATUS">
            <description>Transmit battery informations for a accu pack.</description>
            <field type="uint8_t" name="accu_id">Accupack ID, see ENUM MAV_SKYE_BATTERY_PACK_ID</field>
            <field type="uint16_t" name="voltage_cell_1">Battery voltage of cell 1, in millivolts (1 = 1 millivolt)</field>
            <field type="uint16_t" name="voltage_cell_2">Battery voltage of cell 2, in millivolts (1 = 1 millivolt)</field>
            <field type="uint16_t" name="voltage_cell_3">Battery voltage of cell 3, in millivolts (1 = 1 millivolt)</field>
            <field type="uint16_t" name="voltage_cell_4">Battery voltage of cell 4, in millivolts (1 = 1 millivolt)</field>
            <field type="int16_t" name="current_battery">Battery current, in 10*milliamperes (1 = 10 milliampere), -1: autopilot does not measure the current</field>
            <field type="int8_t" name="battery_remaining">Remaining battery energy: (0%: 0, 100%: 100), -1: autopilot estimate the remaining battery</field>
        </message>
        <message id="153" name="SKYE_TEST_MOTORS">
            <description>Message type for project SKYE configuration with four thrusters and four direction actuations. Requested mode: "MAV_MODE_TESTPHASE_ARMED"</description>
            <field type="uint8_t" name="target_system">System ID</field>
            <field type="uint8_t" name="thrust_1">Thrust of motor 1, range [0,200]</field>
            <field type="uint8_t" name="thrust_2">Thrust of motor 2, range [0,200]</field>
            <field type="uint8_t" name="thrust_3">Thrust of motor 3, range [0,200]</field>
            <field type="uint8_t" name="thrust_4">Thrust of motor 4, range [0,200]</field>
            <field type="int16_t" name="direct_1">Direction of direction motor 1, in 0.1 degrees [-360deg: -3600, 360deg: 3600] </field>
            <field type="int16_t" name="direct_2">Direction of direction motor 2, in 0.1 degrees [-360deg: -3600, 360deg: 3600] </field>
            <field type="int16_t" name="direct_3">Direction of direction motor 3, in 0.1 degrees [-360deg: -3600, 360deg: 3600] </field>
            <field type="int16_t" name="direct_4">Direction of direction motor 4, in 0.1 degrees [-360deg: -3600, 360deg: 3600] </field>
        </message>
        <message id="154" name="SKYE_DIRECT_CONTROL">
            <description>Control six degrees of freedom in Body Frame. This type of control is intended to use if there is no attitude feedback in the control. Requested mode: "MAV_MODE_DIRECT_CONTROL_ARMED"</description>
            <field type="uint8_t" name="target_system">System ID</field>
            <field type="float" name="thrust_x">Resulting thrust in Body Frame x, in Newton</field>
            <field type="float" name="thrust_y">Resulting thrust in Body Frame y, in Newton</field>
            <field type="float" name="thrust_z">Resulting thrust in Body Frame z, in Newton</field>
            <field type="float" name="moment_x">Resulting moment in Body Frame x (roll), in Newtonmeter</field>
            <field type="float" name="moment_y">Resulting moment in Body Frame y (pitch), in Newtonmeter</field>
```

```
            <field type="float" name="moment_z">Resulting moment in ↩
                Body Frame z (yaw), in Newtonmeter</field>
        </message>
        <message id="155" name="SKYE_ASSISTED_CONTROL">
            <description>Control six degrees of freedom. Translational↩
                 velocity in Inertial (earth) Frame. Use this manual ↩
                control mode with the requested mode: "↩
                MAV_MODE_ASSISTED_CONTROL_ARMED"</description>
            <field type="uint8_t" name="target_system">System ID</↩
                field>
            <field type="float" name="translation_lat">Translation (↩
                velocity) in Inertial Frame latitude, in m/sec</field>
            <field type="float" name="translation_long">Translation (↩
                velocity) in Inertial Frame longitude, in m/sec</field↩
                >
            <field type="float" name="translation_alt">Translation (↩
                velocity) in Inertial Frame altitude, in m/sec</field>
            <field type="float" name="rotation_x">Roll (angular ↩
                velocity), in rad/sec</field>
            <field type="float" name="rotation_y">Pitch (angular ↩
                velocity), in rad/sec</field>
            <field type="float" name="rotation_z">Yaw (angular ↩
                velocity), in rad/sec</field>
        </message>
        <message id="157" name="SKYE_SCALED_PRESSURE">
            <description>The pressure readings for the typical setup ↩
                of one absolute and differential pressure sensor. The ↩
                units are as specified in each field. For accurate ↩
                analysis needed by at least 6Hz.</description>
            <field type="uint32_t" name="time_boot_ms">Timestamp (↩
                microseconds since UNIX epoch or microseconds since ↩
                system boot)</field>
            <field type="float" name="press_abs1">Absolute pressure (↩
                hectopascal)</field>
            <field type="float" name="press_diff11">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_diff12">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_diff13">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_abs2">Absolute pressure (↩
                hectopascal)</field>
            <field type="float" name="press_diff21">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_diff22">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_diff23">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_abs3">Absolute pressure (↩
                hectopascal)</field>
            <field type="float" name="press_diff31">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_diff32">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_diff33">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_abs4">Absolute pressure (↩
                hectopascal)</field>
            <field type="float" name="press_diff41">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_diff42">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="float" name="press_diff43">Differential ↩
                pressure 1 (hectopascal)</field>
            <field type="int16_t" name="temperature">Temperature ↩
                measurement (0.01 degrees celsius)</field>
        </message>
        <message id="158" name="SKYE_MOTOR_SIGNAL">
            <description>The values transmitted to the motor nodes.</↩
                description>
            <field type="uint32_t" name="time_usec">Timestamp (since ↩
                UNIX epoch or microseconds since system boot)</field>
            <field type="uint8_t" name="thrust1_raw">Thrust output ↩
                thruster 1, range [0,200]</field>
            <field type="uint8_t" name="thrust2_raw">Thrust output ↩
                thruster 2, range [0,200]</field>
            <field type="uint8_t" name="thrust3_raw">Thrust output ↩
                thruster 3, range [0,200]</field>
            <field type="uint8_t" name="thrust4_raw">Thrust output ↩
                thruster 4, range [0,200]</field>
```

```
            <field type="int16_t" name="position1_raw">Orientation ↩
                output position motor 1, in 0.1 degrees [-360deg: ↩
                -3600, 360deg: 3600]</field>
            <field type="int16_t" name="position2_raw">Orientation ↩
                output position motor 2, in 0.1 degrees [-360deg: ↩
                -3600, 360deg: 3600]</field>
            <field type="int16_t" name="position3_raw">Orientation ↩
                output position motor 3, in 0.1 degrees [-360deg: ↩
                -3600, 360deg: 3600]]</field>
            <field type="int16_t" name="position4_raw">Orientation ↩
                output position motor 4, in 0.1 degrees [-360deg: ↩
                -3600, 360deg: 3600]</field>
        </message>
        <message id="159" name="SKYE_MOTOR_MEASSURED_POSITION">
            <description>The meassured orientation of the motors</↩
                description>
            <field type="uint32_t" name="time_usec">Timestamp (since ↩
                UNIX epoch or microseconds since system boot)</field>
            <field type="int16_t" name="pos1_raw">Meassureed ↩
                orientation motor 1, in 10qc [360deg: 17614]</field>
            <field type="int16_t" name="pos2_raw">Meassureed ↩
                orientation motor 1, in 10qc [360deg: 17614]</field>
            <field type="int16_t" name="pos3_raw">Meassureed ↩
                orientation motor 1, in 10qc [360deg: 17614]</field>
            <field type="int16_t" name="pos4_raw">Meassureed ↩
                orientation motor 1, in 10qc [360deg: 17614]</field>
        </message>
        <message id="160" name="SKYE_CONTROLLER_OUTPUT">
            <description>Output of controller (Input for actuation ↩
                chain)</description>
            <field type="uint32_t" name="time_usec">Timestamp (since ↩
                UNIX epoch or microseconds since system boot)</field>
            <field type="float" name="force_x">Signal given from ↩
                controller to actuation chain, 1 Newton</field>
            <field type="float" name="force_y">Signal given from ↩
                controller to actuation chain, 1 Newton</field>
            <field type="float" name="force_z">Signal given from ↩
                controller to actuation chain, 1 Newton</field>
            <field type="float" name="moment_x">Signal given from ↩
                controller to actuation chain, 1 Newtonmeter</field>
            <field type="float" name="moment_y">Signal given from ↩
                controller to actuation chain, 1 Newtonmeter</field>
            <field type="float" name="moment_z">Signal given from ↩
                controller to actuation chain, 1 Newtonmeter</field>
        </message>
        <message id="161" name="SKYE_CAM_RECONFIGURE_BLUEFOX_SETTINGS">
            <description>Message to change all the parameters present ↩
                in ROS Reconfigure GUI bluefox node</description>
            <field type="uint8_t" name="target_system">System ID</↩
                field>
            <!--            <field type="char[32]" name="guid">↩
                Serial number of camera, suffix BLUEFOX_ + 8 dec ↩
                digits (use first camera if null)</field> -->
            <field type="uint8_t" name="cam_id">ID of camera, see ENUM↩
                 MAV_CAM_ID</field>
            <field type="char[32]" name="frame_id">ROS tf frame of ↩
                reference, resolved with tf_prefix unless absolute.</↩
                field>
            <field type="uint8_t" name="pixel_clock">Pixel clock of ↩
                image sensor [MHz]. See enum ↩
                MAV_CAM_RECONFIG_PIXEL_CLOCK</field>
            <field type="float" name="frame_rate">Camera speed (frames↩
                 per second). Range: 1.0 to 240.0</field>
            <field type="char[32]" name="camera_info_url">Camera ↩
                calibration URL for this video_mode (uncalibrated if ↩
                null). </field>
            <field type="uint8_t" name="binning_x">Number of pixels ↩
                combined for horizontal binning, use device hints if ↩
                zero. Range: 0 to 4</field>
            <field type="uint8_t" name="binning_y">Number of pixels ↩
                combined for vertical binning, use device hints if ↩
                zero. Range: 0 to 4</field>
            <field type="uint16_t" name="roi_width">Width of Region of↩
                 Interest in unbinned pixels, full width if zero. ↩
                Range: 0 to 5000</field>
            <field type="uint16_t" name="roi_height">Height of Region ↩
                of Interest in unbinned pixels, full height if zero. ↩
                Range: 0 to 5000</field>
            <field type="uint16_t" name="x_offset">Horizontal offset ↩
                for left side of ROI in unbinned pixels. Range: 0 to ↩
```

```
                5000</field>
            <field type="uint16_t" name="y_offset">Vertical offset for↩
                 top of ROI in unbinned pixels. Range: 0 to 5000</↩
                field>
            <field type="uint8_t" name="color_coding">Color coding. ↩
                See enum MAV_CAM_RECONFIG_COLOR_CODING</field>
            <field type="uint8_t" name="bayer_method">Bayer decoding ↩
                method (default: ROS image_proc). See enum ↩
                MAV_CAM_RECONFIG_BAYER_METHOD</field>
            <field type="uint8_t" name="exposure">Auto exposure value ↩
                . Range: 0 to 255</field>
            <field type="uint8_t" name="shutter_auto">boolean. Shutter↩
                 control state. </field>
            <field type="float" name="shutter_auto_min">Min shutter ↩
                time [s] in auto mode. Range: 0.0 to 0.1</field>
            <field type="float" name="shutter_auto_max">Max shutter ↩
                time [s] in auto mode Range: 0.0 to 0.1</field>
            <field type="float" name="shutter">Shutter time [s]. ↩
                Range: 0.0 to 0.1</field>
            <field type="uint8_t" name="gain_auto">boolean. Gain ↩
                control state. </field>
            <field type="uint8_t" name="gain_auto_min">Min relative ↩
                circuit gain [dB] in auto mode. Range: 0.0 to 12.0</↩
                field>
            <field type="uint8_t" name="gain_auto_max">Max relative ↩
                circuit gain [dB] in auto mode. Range: 0.0 to 12.0</↩
                field>
            <field type="uint8_t" name="gain">Relative circuit gain [↩
                dB]  Range: 0.0 to 12.0</field>
            <field type="uint8_t" name="auto_control_speed">Control ↩
                speed for automatic features. See enum ↩
                MAV_CAM_RECONFIG_AUTO_CONTROL_SPEED</field>
            <field type="uint8_t" name="auto_query_values">boolean. ↩
                Queries settings that are auto controlled. </field>
            <field type="uint8_t" name="hdr_mode">HDR mode. See enum ↩
                MAV_CAM_RECONFIG_HDR_MODE</field>
            <field type="uint8_t" name="use_ros_time">boolean. ↩
                Timestamp Image and CameraInfo using ros::Time::now() ↩
                </field>
        </message>
        <message id="162" name="SKYE_CAM_RECONFIGURE_PROSILICA_SETTINGS↩
            ">
            <description>Message to change all the parameters present ↩
                in ROS Reconfigure GUI prosilica node. FIXME: MESSAGE ↩
                NOT DEFINED YET</description>
            <field type="uint8_t" name="target_system">System ID</↩
                field>
            <field type="uint8_t" name="cam_id">ID of camera, see ENUM↩
                 MAV_CAM_ID (MAV_CAM_ID_PROSILICA)</field>
        </message>
        <message id="163" name="SKYE_CAM_RECONFIGURE_IMAGE_HANDLER">
            <description>Set parameters of image handler which stores ↩
                images on HDD and steams them via wifi</description>
            <field type="uint8_t" name="target_system">System ID</↩
                field>
            <field type="uint8_t" name="cam_id">ID of camera, see ENUM↩
                 MAV_CAM_ID</field>
            <field type="uint8_t" name="save_image">Boolean, true: ↩
                image stream is saved as specified in fields below</↩
                field>
            <field type="uint8_t" name="save_percent">Percentage of ↩
                shot images that are stored</field>
            <field type="uint8_t" name="format">Format of saved image,↩
                 see ENUM MAV_CAM_IMAGE_FORMAT</field>
            <field type="uint8_t" name="png_level">Compression level ↩
                for png save format (no compr:1, full compr:9)</field>
            <field type="uint8_t" name="jpeg_quality">Compression ↩
                quality for jpeg save format (lowest quality: 0, ↩
                highest quality: 100)</field>
            <field type="char[8]" name="frame_name">Frame name, max. 8↩
                 characters</field>
            <field type="char[32]" name="path">Path, where images on ↩
                SKYE should saved to, max. 32 characters</field>
            <field type="uint8_t" name="send_image">Boolean, true: ↩
                image stream is sent to GS as specified in fields ↩
                below</field>
            <field type="uint8_t" name="send_percent">Percentage of ↩
                shot images that are sent</field>
            <field type="uint8_t" name="format2">Format of sent image,↩
                 see ENUM MAV_CAM_IMAGE_FORMAT</field>
```

```
            <field type="uint8_t" name="png_level2">Compression level ↩
                for png send format (no compr:1, full compr:9)</field>
            <field type="uint8_t" name="jpeg_quality2">Compression ↩
                quality for jpeg send format (lowest quality: 0, ↩
                highest quality: 100)</field>
            <field type="uint8_t" name="keep_old_modus">Boolean, true:↩
                 new image handle modus is added to previous (image is↩
                 saved/sent more than once), false: only
save/send with new configuration</field>
        </message>
        <message id="164" name="SKYE_CAM_TAKE_SHOT">
            <description>Take a single shot and store the image via ↩
                ROS on groundstation</description>
            <field type="uint8_t" name="target_system">System ID</↩
                field>
            <field type="uint8_t" name="cam_id">ID of camera, see ENUM↩
                 MAV_CAM_ID</field>
            <field type="uint8_t" name="take_shot">Boolean, true: send↩
                 recent image (save settings) to GS</field>
        </message>
        <message id="165" name="SKYE_CAM_IMAGE_TRIGGERED">
            <description>This message indicates that there is a new ↩
                image in default local path that can be displayed in ↩
                QGC. Use this message only if you do NOT transmit the ↩
                image via the mavlink protocol (ENCAPSULED_DATA)</↩
                description>
            <field type="uint64_t" name="timestamp">Timestamp</field>
            <field type="uint8_t" name="cam_id">ID of camera, see ENUM↩
                 MAV_CAM_ID</field>
            <!--                <field type="uint32_t" name="seq">IMU ↩
                seq</field>      -->
            <field type="float" name="roll">Roll angle in rad</field>
            <field type="float" name="pitch">Pitch angle in rad</field↩
                >
            <field type="float" name="yaw">Yaw angle in rad</field>
            <!--                <field type="float" name="local_z">↩
                Local frame Z coordinate (height over ground)</field> ↩
                  -->
            <field type="int32_t" name="lat">Latitude, expressed as * ↩
                1E7</field>
            <field type="int32_t" name="lon">Longitude, expressed as *↩
                 1E7</field>
            <field type="int32_t" name="alt">Altitude in meters, ↩
                expressed as * 1000 (millimeters), above MSL</field>
            <!--                <field type="float" name="ground_x">↩
                Ground truth X</field>
            <field type="float" name="ground_y">Ground truth Y</field>
            <field type="float" name="ground_z">Ground truth Z</field>↩
                    -->
        </message>
        <message id="166" name="SKYE_HOME_MAXON">
            <description/>
            <field type="uint8_t" name="target_system">System ID</field↩
                >
            <field type="uint8_t" name="homing">Boolean, true: home ↩
                maxon motors</field>
        </message>
        <message id="167" name="SKYE_THREAD_COUNTS">
            <description>Info about running threads on autopilot</↩
                description>
            <field type="uint32_t" name="time_usec">Timestamp (since ↩
                UNIX epoch or microseconds since system boot)</field>
            <field type="uint8_t" name="running_threads">Number of ↩
                running threads</field>
            <field type="uint64_t" name="heartbeatloop_count">Counter ↩
                of heartbeatloop</field>
            <field type="uint64_t" name="receiveloop_count">Counter of↩
                 receiveloop</field>
            <field type="uint64_t" name="telemetryloop_count">Counter ↩
                of telemetryloop</field>
            <field type="uint64_t" name="gyroaccloop_count">Counter of↩
                 gyroaccloop</field>
            <field type="uint64_t" name="controlloop_count">Counter of↩
                 controlloop</field>
            <field type="uint64_t" name="stateestimation_count">↩
                Counter of stateestimation</field>
        </message>
        <message id="168" name="SKYE_THREAD_USLEEP">
            <description>Info about running threads on autopilot</↩
                description>
```

```
            <field type="uint32_t" name="time_usec">Timestamp (since ↩
                UNIX epoch or microseconds since system boot)</field>
            <field type="int64_t" name="heartbeatloop_usleep">↩
                Microseconds heartbeatloop is sleeping</field>
            <field type="int64_t" name="receiveloop_timestamp">↩
                Microseconds receiveloop is sleeping</field>
            <field type="int64_t" name="telemetryloop_usleep">↩
                Microseconds telemetryloop is sleeping</field>
            <field type="int64_t" name="gyroaccloop_usleep">↩
                Microseconds gyroaccloop is sleeping</field>
            <field type="int64_t" name="controlloop_usleep">↩
                Microseconds controlloop is sleeping</field>
            <field type="int64_t" name="stateestimation_usleep">↩
                Microseconds stateestimation loop is sleeping</field>
        </message>
        <message id="169" name="SKYE_REQUEST_CAM_RECONFIGURE_SETTINGS">
            <description>Request to transmit the current camera ↩
                reconfigure settings of camera specified by cam_id</↩
                description>
            <field type="uint8_t" name="target_system">System ID</↩
                field>
            <field type="uint8_t" name="cam_id">ID of camera, see ENUM↩
                 MAV_CAM_ID</field>
        </message>
        <message id="170" name="SKYE_REQUEST_CAM_RECONFIGURE_HANDLER">
            <description>Request to transmit the current image handler↩
                 settings of camera specified by cam_id</description>
            <field type="uint8_t" name="target_system">System ID</↩
                field>
            <field type="uint8_t" name="cam_id">ID of camera, see ENUM↩
                 MAV_CAM_ID</field>
        </message>
<!-- DATA_TRANSMISSION_HANDSHAKE and ENCAPSULATED_DATA copied from ↩
    pixhawk.xml -->
        <message id="193" name="DATA_TRANSMISSION_HANDSHAKE">
            <description>Tell QGC that a new image will be transmitted↩
                 as ENCAPSULATED_DATA</description>
            <field type="uint8_t" name="type">type of requested/↩
                acknowledged data (as defined in ENUM DATA_TYPES in ↩
                mavlink/include/mavlink_types.h)</field>
            <field type="uint32_t" name="size">total data size in ↩
                bytes (set on ACK only)</field>
            <field type="uint16_t" name="width">Width of a matrix or ↩
                image</field>
            <field type="uint16_t" name="height">Height of a matrix or↩
                 image</field>
            <field type="uint8_t" name="packets">number of packets ↩
                beeing sent (set on ACK only) </field>
            <field type="uint8_t" name="payload">payload size per ↩
                packet (normally 253 byte, see DATA field size in ↩
                message ENCAPSULATED_DATA) (set on ACK only) </field>
            <field type="uint8_t" name="jpg_quality">JPEG quality out ↩
                of [1,100]</field>
        </message>
        <message id="194" name="ENCAPSULATED_DATA">
            <description>Split image into mavlink packages to send it ↩
                in the data array</description>
            <field type="uint16_t" name="seqnr">sequence number (↩
                starting with 0 on every transmission)</field>
            <field type="uint8_t[253]" name="data">image data bytes</↩
                field>
        </message>
    </messages>
</mavlink>
```

Listing A.1: "Definition of Skye specific Mavlink messages"

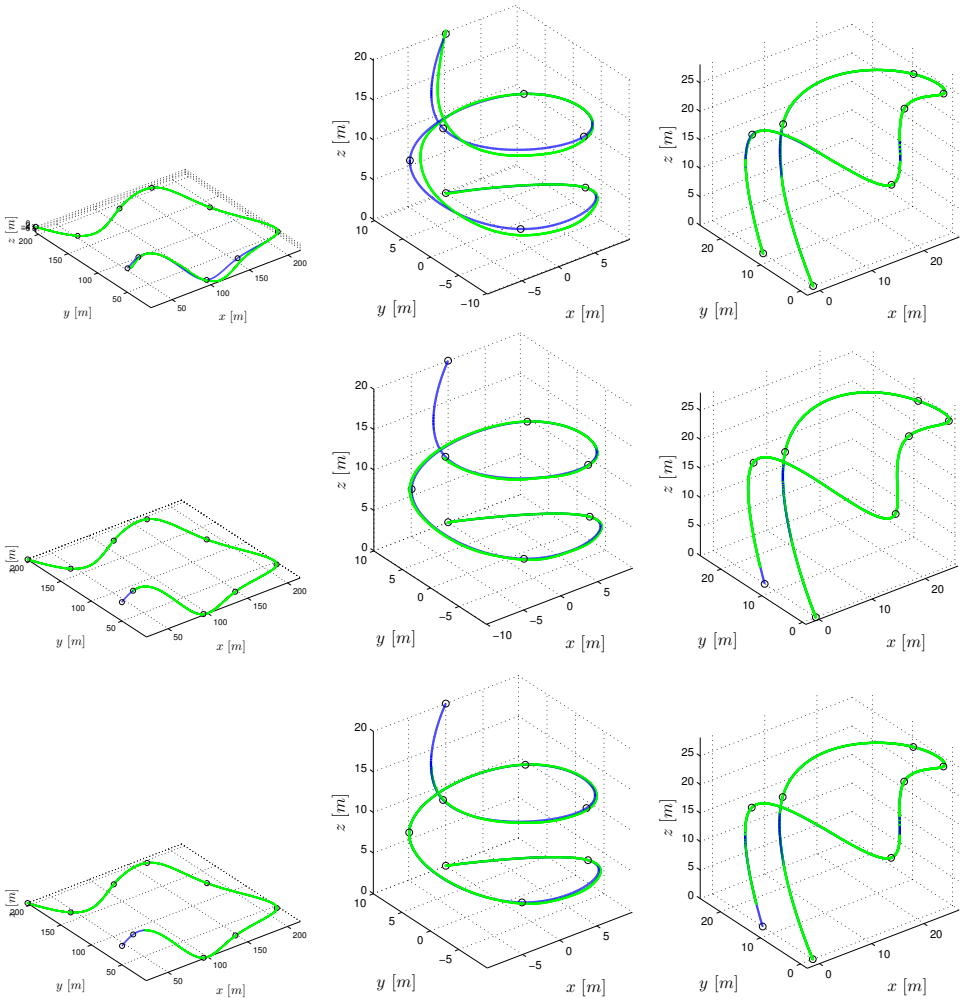


Figure A.3: Trajectory tracking with wind disturbance (1 m/s x-direction). **Top**: *Trajectory following* control forces the system to reach the end within the given constraints. As trajectory constraints do not consider wind, tracking errors are high. **Center**: *Pure pursuit* control is more robust to wind and the tracking still accurate. In comparison to the ideal case it only makes only 85-95% of the track within T_p. **Bottom**: *Cross track error* control is accurate and makes 89-95% of the track within T_p.

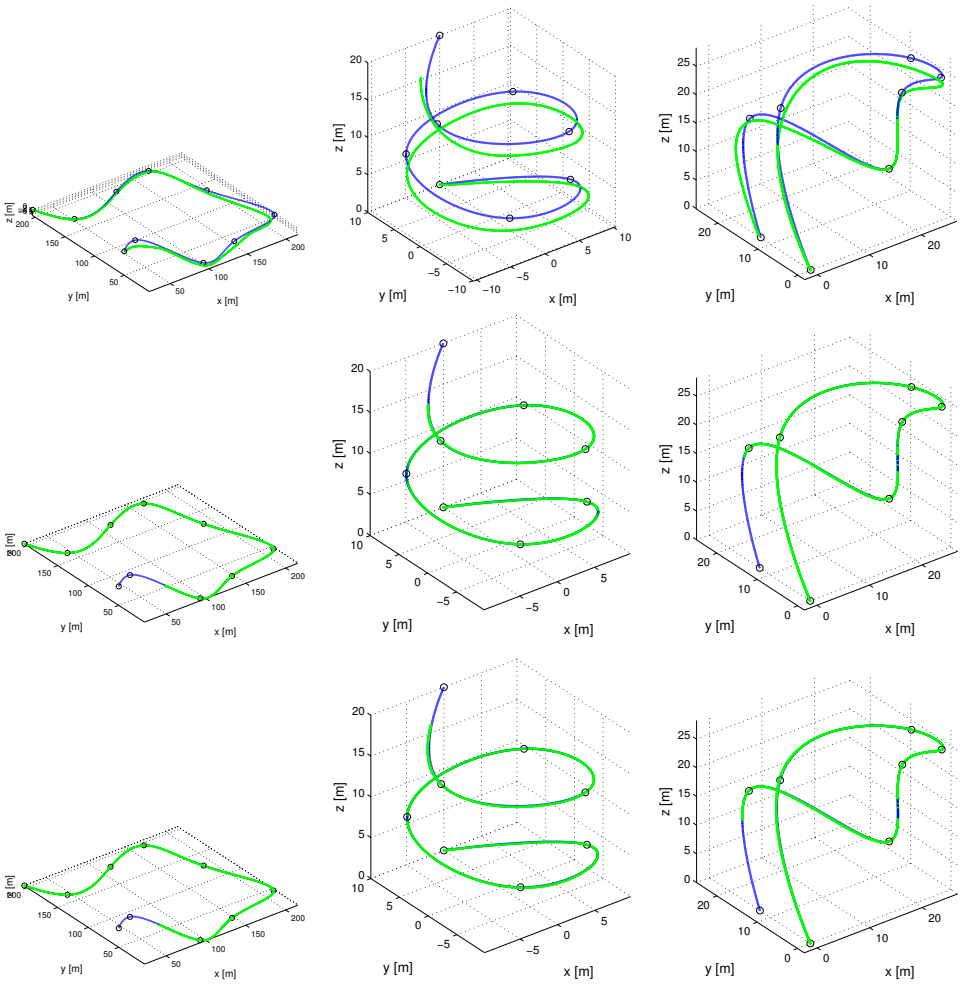


Figure A.4: Trajectory tracking with wrong system constraints (+20%). **Top**: *Trajectory following* control forces the system to reach the end within the given constraints. As trajectory demands too high performance, tracking gets bad. **Center**: *Pure pursuit* control is more robust to the wrong conditioned trajectory and tracking is quite accurate. In comparison to the ideal case it only makes 86-90% of the track within T_p. **Bottom**: *Cross track error* control is accurate and makes 87-93% of the track within T_p.

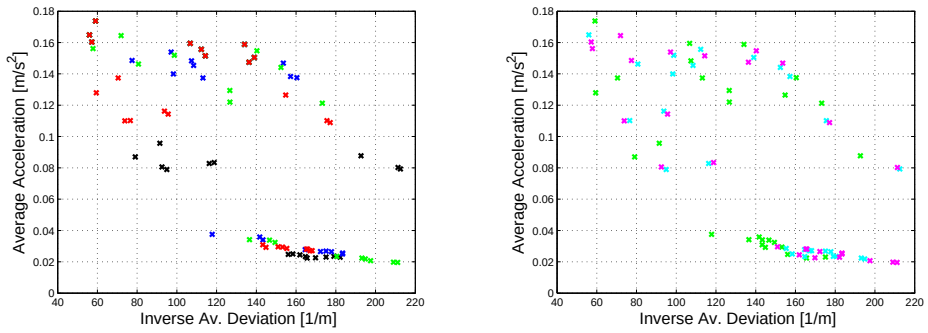


Figure A.5: Correlation between average deviation and average acceleration in the simulation

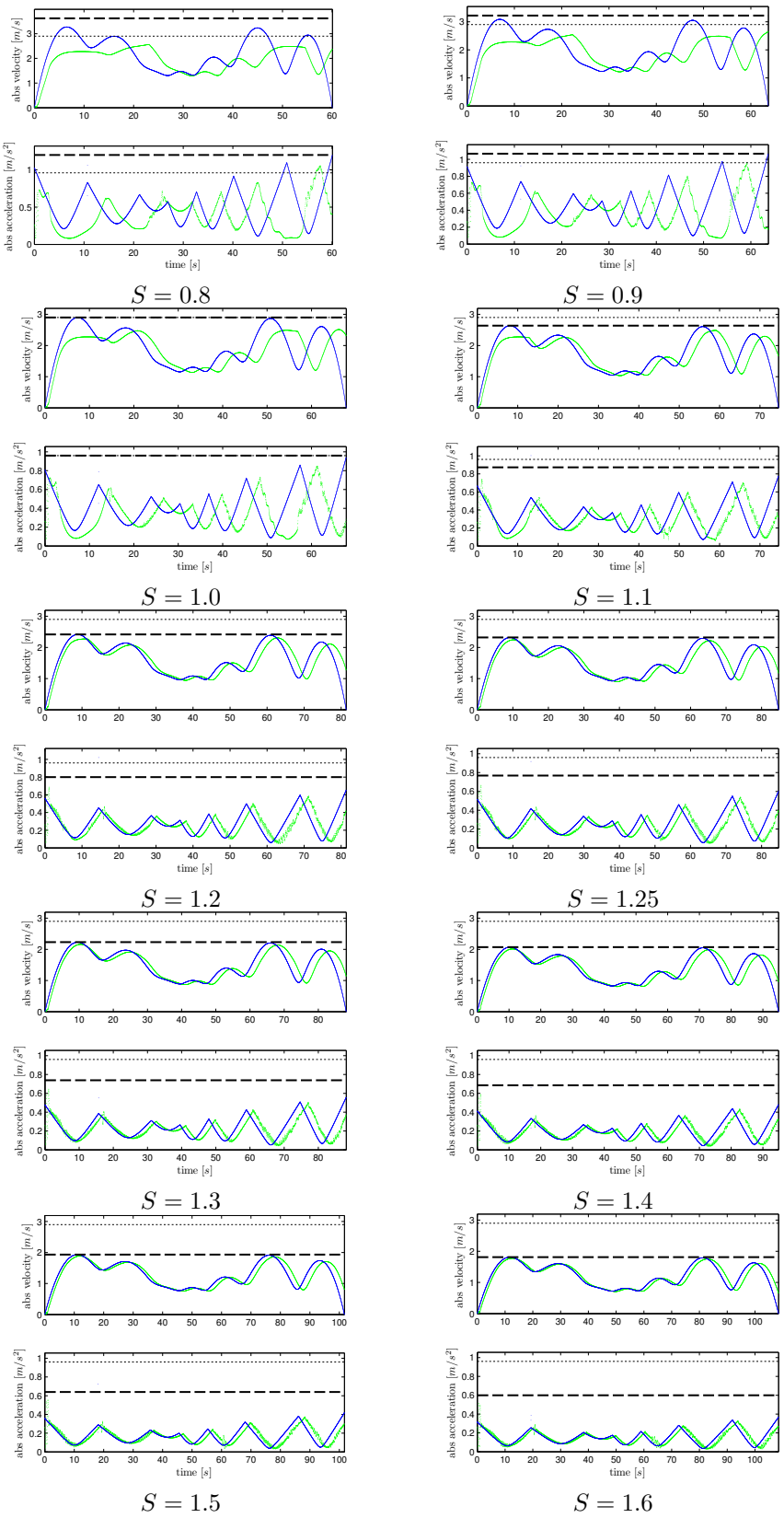


Figure A.6: Trajectory (blue), trace (green), $(\cdot)_{lim} = (\cdot)_{max}/S$ (dashed bold) and $(\cdot)_{max}$ (dashed)

Bibliography

[1] G. Engeln-Müllges, K. Niederdrenk, R. Wodicka: *Numerik-Algorithmen : Verfahren, Beispiele, Anwendungen*. Springer Verlag, 2011.

[2] L. Biagiotti, C. Melchiorri: *Trajectory Planning for Automatic Machines and Robots*. Springer Verlag, 2008.

[3] J. M. Snider: *Automatic Steering Methods for Autonomous Automobile Path Tracking*. Research Report CMU-RI-TR-09-08, Robotics Institute Carnegie Mellon University Pittsburgh, Pennsylvania, 2009.

[4] A. De Luca, G. Oriolo, C. Samson: *Feedback Control of a Nonholonomic Car-Like Robot*. In Robot Motion Planning and Control, pages 171-249, 1998.

[5] D. L. Williams: *Loitering Behaviors of Autonomous Underwater Vehicles*. MSc thesis, Naval Postgraduate School, Monterey, California, 2002.

[6] T. Kammermann: *Evaluation and implementation of a control device for a ballbot*. BSc thesis, ETH Zurich, 2010.

[7] S. Dössegger: *Time-optimal trajectories for a Ballbot*. BSc thesis, ETH Zurich, 2010.

[8] J. Weichart: *Agile Blimp Modeling and Simulation Environment*. BSc thesis, ETH Zurich, 2012.

[9] D. Meier, L. Müri: *Agile Blimp Controller Design*. BSc thesis, ETH Zurich, 2012.

[10] A. Schaffner, N. Vuilliomenet: *Blimp Actuation Chain Modeling and Optimization*. BSc thesis, ETH Zurich, 2012.

[11] M. Burri: *Communication Software for a Blimp*. BSc thesis, ETH Zurich, 2012.

[12] Won Y. Yang, [et al.]: *Applied Numerical Methods Using MATLAB* . Wiley-Interscience, Hoboken, 2005.

[13] J. Blanchette, M Summerfield: *C++ GUI Programming with Qt 4*. Prentice Hall, Upper Saddle River, N.J., 2010.

[14] E.T.Y. Lee: *Choosing nodes in parametric curve interpolation, Computer-Aided Design*. pages 363-370, (http://www.sciencedirect.com/science/article-/pii/0010448589900031), 1989

[15] U. Stammbach: *Analysis I/II, Teil A*. ETH Zurich, 2005

[16] W. Dahmen, A. Reusken: *Numerik für Ingenieure und Naturwissenschaftler*. Springer Verlag, Berlin, 2006.

[17] H. Haron, [et al.]: *Parameterization Method on B-Spline Curve*. Mathematical Problems in Engineering, vol. 2012.