University of Pisa

DEPARTMENT OF INFORMATION ENGINEERING
Master Course in Robotics and Automation

Navigation and Control for an Autonomous Sailing Model Boat

Candidate:
Marco Tranzatto

Thesis advisors:
Prof. Alberto Landi
Prof. Andrea Caiti

Research supervisors:
Dr. Sergio Grammatico
Mr. Alexander Liniger

Thesis submitted in 2015

Abstract

The purpose of this thesis is to develop navigation and control strategies for an autonomous sailing model boat. Autonomous sailboats are good candidates both for long term oceanic surveys and for patrolling and stealth operations, since they use wind power as their main mean of propulsion which ensures low power requirements, a minimal acoustic signature and a relatively small detectable body. Controlling a sailboat, however, is not an easy task due to high variability in wind, side drift of the boat and challenges encountered when attempting to traverse an upwind course. In this thesis, we describe how we design and set up a control architecture that allows Aeolus, an autonomous model sailboat provided by the Swiss Federal Institute of Technology in Zurich, ETH, to sail upwind and execute fast and smooth tacking maneuvers. We implemented different controllers to actuate the rudder in upwind sailing while tacking. We present experimental results obtained during several autonomous sailing tests conducted at Lake Zurich, Switzerland.

Ma se tu guardi un monte che hai di faccia,
senti che ti sospinge a un altro monte,
un'isola col mare che l'abbraccia,
che ti chiama a un'altra isola di fronte,
e diedi un volto a quelle mie chimere,
le navi cotruii di forma ardita,
concavi navi dalle vele nere,
e nel mare cambiò quella mia vita,
e il mare trascurato mi travolse,
seppi che il mio futuro era sul mare,
con un dubbio però che non si sciolse,
senza futuro era il mio navigare.

Odysseus, Francesco Guccini.

Contents

List of Figures

List of Tables

Glossary

ψ compass heading angle.

σ true wind direction.

α heading relative to the wind.

χ course over ground.

δ rudder command.

μ sail command.

ω yaw rate.

EKF extended Kalman filter.

NED North-East-Down local frame.

Chapter 1

Introduction

In the past, sailing was the only achievable means of transport over the sea. The importance of sailing cannot be easily summarized, but think about two events, one legendary and one real, to understand its importance. Sail power was used to transport Aeneas, the Trojan hero, when he left Troy and landed in Italy. Aeneas, according to Virgil's Aeneid, is one of the few Trojans who were not killed or enslaved when Troy fell. The Trojan hero, after being commanded by the gods to flee, traveled to Italy, where his dynasty founded Rome, on April 21st, 753 BC. Rome would become the center of the greatest and most important ancient empire all over the world: the Roman Empire, which at its greatest extentaions covered 5 million km^2. Coming back to real facts, when Christoper Columbus and his crew discovered the new world on October 12, 1492, it marked the onset of the early modern period. They used three wind powered vessels, the Niña, the Pinta and the Santa María, to sail from Spain to San Salvador Island, near Cuba. Nowadays however, sailing is used for sports, racing competitions and outdoor activities. Sailing is not just an expensive hobby, but is becoming a major area of research because it is able to use wind power as primary means of propulsion. Therefor it can be used to accomplish long term tasks, where endurance is a key feature.

1.1 Motivation

Nowadays unmanned surface vehicles are becoming a key research point, both for oceanographic and surveillance use. For example, the European project MORPH [1] aims to develop efficient methods and tools for underwater environment mapping, using both surface and underwater vehicles. Unmanned surface vehicles act as a link, between the underwater environment, where only acoustic communication is available, and the above water

environment, where radio link and GPS signal are available to coordinate the mission. Another example, is the REP10 AUV experiment [2]. The same link behavior is used. This project is focused on the demonstration of heterogeneous autonomous vehicles cooperation (air, surface and underwater unmanned vehicles) in mine clearance and rapid environmental assessment missions. However, endurance is a challenge for these unmanned vehicles. In fact, they cannot operate for a long period of time, without needing being charged. To overcome this problem and ensure a long term endurance, we can look at an example used in underwater environments. Seaglidres [3] have been developed to execute oceanographic surveys for a long period of time. These are small, reusable autonomous underwater vehicles designed to glide from the ocean surface to a programmed depth and back while collecting several data. They are used for missions exceeding several thousand kilometers and lasting many months. For example the commercial seaglider produced by Kongsberg, named The Seaglider [4] and a low-cost one produced by Graal Tech, named Fólaga [5]. An autonomous sailing model boat is the perfect surface substitute for seagliders to execute oceanographic surveys, as mentioned for example in [6, 7]. By employing wind power, it can provide a long term mission endurance, since the only electrical power required is consumed by the electronics. Moreover, such a surface vehicle can continuously use a radio link communication as well as GPS signal. Finally, the autonomy of surface vehicles can be increased by installing a solar pannel on the deck. These are the motivations to develop, test and employ an autonomous sailing model boat.

1.2 State of the Art

Many research groups have developed small-scale robotic sailboats in recent years. For instance, the University of British Columbia (UBC team [8]), the Tufts University (Trst team [9]), the Olin College of Engineering (Olin robotic sailing team [10]), and the University of Porto (FASt sailing boat team [11]). Every year a World Robotic Sailing Championship (WRSC) [12] is organized and competitors attempt to complete several tasks. For example, in 2014 the competition took place in Ireland, and the main tasks to be executed were upwind/downwind sailing, station-keeping, fleet race, endurance race and obstacle avoidance.

Most studies on sailing robots focus on decoupling the control system of the rudder from the one of the sail, assuming their coupling behavior is negletable. For example, [13] shows how to implement one control method for the rudder and another for sail control to enable the robot to sail following

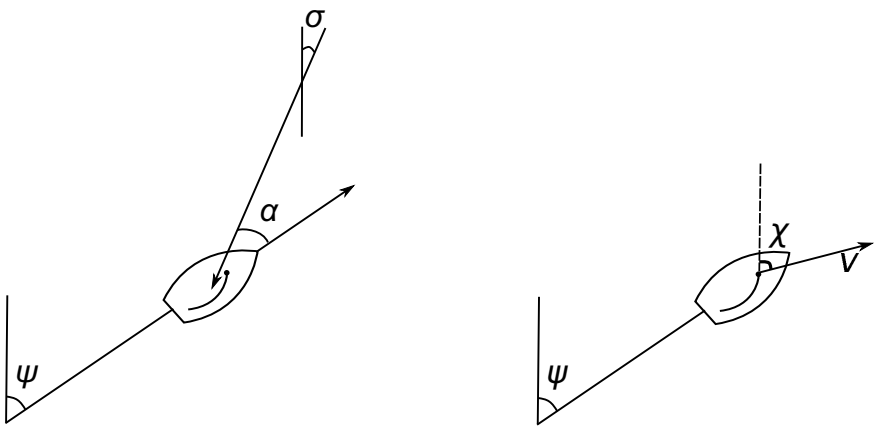
a straight line on the surface of the water. Moreover, most studies do not describe how to execute special sailing maneuvers such as the tack or the jibe. Since a mathematical model of the dynamics of a model sailboat is typically not easy to derive, several works use fuzzy logic [14–16] (namely, empirical rule-based logic) to control both the rudder and the sail. The main idea is to exploit the knowledge of the "helmsman", which can be expressed in terms of a rule-based control system. In [16], for example, two decoupled fuzzy controllers, one for the rudder and one for the sail, are developed using simple rules. These two controllers are used both to sail autonomously and to execute tack and jibe maneuvers. Two field results, one for the tack and one for the jibe maneuver, are provided. Some other works prefer to use a standard approach to control a sailing boat. For example, [17] explains how to identify a continuous linear second order model for the steering dynamics of a model sailboat, and to design a proportional-integral (PI) feedback law for rudder. Therein, the PI controller is only used to track a desired heading while sailing, while another controller regulates the sail. Another example is provided by [18], where a discrete-time transfer function for the steering dynamics of a model sailboat is derived, and then used to find all discrete-time proportional-integral-derivative (PID) controllers that satisfy a robust stability constraint for heading control. In [19] it is shown how to derive a four-degree-of-freedom (DOF) nonlinear dynamic model for a sailing yacht. Then a nonlinear heading controller using the integrator backstepping method is designed, which exponentially stabilizes the heading/yaw dynamics. The same authors in [20] design a time-invariant linear model, i.e. a first-order Nomoto model, for which a $\mathcal{L}_1$ adaptive controller is implemented to achieve the heading regulation.

1.3 Sailing Background

A sailboat cannot move in every direction without taking into account the direction from which the wind is blowing. In fact, there is a region relative to the wind where a sailboat cannot navigate at all, which is indeed called the "no-go zone" [21]. To better understand the next Chapters, we define some angles, to describe the state of the boat with respect to the environment:

1. yaw or heading angle, ψ. This is the compass angle and indicates where the bow of the vessel is pointing;

2. true wind direction angle, σ. It is the direction from where the wind blows;



(a) Case without drift (b) Case with drift

Figure 1.1: Representation of the angles ψ, χ, σ and α. ψ is the heading angle (where the bow is pointing), χ is the course-over-ground angle (where the boat is going toward), v is the velocity vector of the vessel, σ is the true wind direction and α is heading relative to the wind.

3. heading with respect to the wind, α. It is the relative heading of the boat, with respect to the true wind direction;

4. course over ground angle, χ. It is the direction to where the boat is actually going. It can be different from the heading angle ψ if there is drift, caused for example by the either the action of the waves or of the wind.

A graphical representation of these angles is depicted in Figure 1.1. Note that we named the variable σ using the adjective "true" on purpose: the true wind direction is the angle from which the wind is blowing, if it is observed by a stationary observer. On the other hand, the apparent wind direction is the direction of the wind experienced by a moving observer, and therefore it is influenced by the velocity of observer himself. We will always refer to the wind, as the the "true" one, if it is not specified something else. The heading relative to the wind, that is the α angle, is an important value, since it defines the position of the boat, relative to the wind, and it is heavily used in this project. The main points of sail and the "no-go zone" are depicted in Figure 1.2. The "no-go zone" is the area which a sailboat cannot traverse, because its motion would oppose the wind direction. The amplitude of this zone depends on the specific boat and on environment conditions.

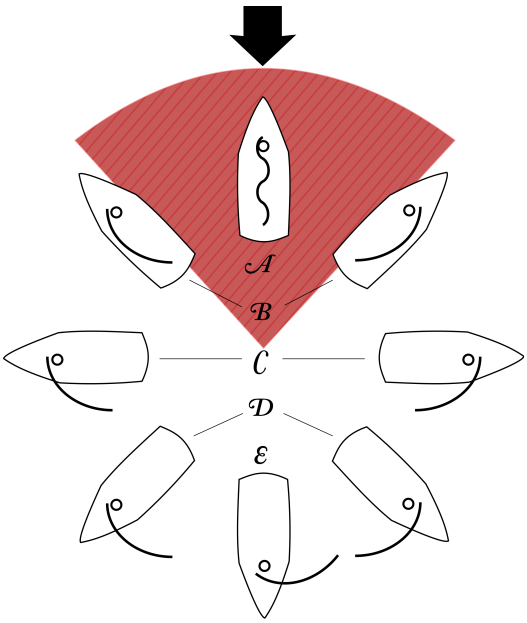


Figure 1.2: Points of sail: in irons (A), close-hauled (B), beam reach (C), broad reach (D) and running (E). The shaded red area is the "no-go zone". Credit by Wikipedia.

Figure 1.3: AEOLUS, the autonomous sailing model boat.

1.4 Aeolus, the Sailing Model Boat of ETH Zurich

AEOLUS, depicted in Figure 1.3, is the autonomous model sailing boat used in the Automatic Control Laboratory (Institute für Automatic) at ETH Zurich [22]. Its name comes from the ruler of the winds in Greek mythology.

1.5 Outline of the Thesis

This thesis is structured in eight main chapters. The first is the Introduction, and the last is the the Appendix, where some specific implementation details are explained.

Chapter 2 - Hardware and Software Setup
The main setup of AEOLUS is presented. The principal electronic devices mounted onboard are shown, as well as the implemented software architecture.

Chapter 3 - Modeling and Identification
Two linear model (ARX and state space) are used to approximate the steering dynamic of the vessel. Numerical and validation results, obtained from

several tests at lake Zurich, are presented.

Chapter 4 - Collecting and Filtering Data
The data collected from sensors are filtered and combined, before being used by the rudder and sail controllers to achieve the autonomous sailing goal.

Chapter 5 - Tracking a Constant Heading
Two controllers, a simple proportional gain and a nonlinear regulator for the rudder are shown. Moreover a rule based controller for the sail is presented. By controlling both the sail and the rudder, it is shown how AEOLUS can track a reference heading with respect to the wind.

Chapter 6 - Tacking Maneuvers
Five different controllers for the tack maneuver are presented. Each has been tested in several trials conducted in Lake Zurich. Performance of the various controllers has have been evaluated and compared.

Chapter 7 - Conclusion
The final results of the thesis are shown. It turns out the the last three controllers for the tack maneuver have got roughly the same performance and the *MPC* is slightly better than the others. A suggested standard configuration to sail upwind and execute tack maneuver is shown. Moreover, ideas for further development and improvements are discussed.

Chapter 2

Hardware and Software Setup

In this Chapter, we show the main hardware components and software architecture used to achieve the autonomous sailing goal. Starting from previous work [23] on our sailing model boat, we briefly explain the main setup and modifications that have been made on the initial state of Aeolus.

2.1 Hardware

We started with an international one-meter RC model sailboat, and installed specialized electronics. Our hardware is mainly composed of an autopilot control unit and a weather station.

Since sensing the wind is clearly fundamental for a sailboat, we use a dedicated hardware device for this. The weather station we employ is the Airmar WS-200WX [24], depicted in Figure 2.1, which we have mounted above the bow. It provides the apparent wind (direction and speed), estimated true wind (direction and speed) and GPS position. All these values are sampled every 200 milliseconds.

Our autopilot is the Pixhawk board [25], an independent, open-source, open-hardware board, shown in Figure 2.2. The Pixhaw is an all-in-one unit, combining a *FMU*(Flight Management Unit) and an *I/O* module (In-out/Output) in one single package. It is equipped with a 168 MHz ARM Cortex-M4 CPU, with a hardware floating point unit. This board provides a POSIX-compatible real time operating system, where many applications can run in parallel. Its *I/O* model is equipped with several sensors, such as accelerometers, gyroscopes, magnetometers, etc. Using a radio link, the Pixhawk sends the data collected online to a specific application, called QGroundControl [26], running on a PC located on the shore, near where Aeolus is sailing. Moreover, a microSD card slot is integrated into the

Figure 2.1: The AIRMAR 200WX weather station.

Figure 2.2: The PIXHAWK board.

board. This makes it possible to record on the micro SD card almost all the data collected. These data are then analyzed in post processing, supplying useful information.

Two hardware modifications, shown in Figure 2.3, have been carried out during this project: the radio antenna has been mounted on the deck, to increase the communication range, and a 3D printed box has been printed an installed inside the lower deck to store the electronics.

2.2 Software

We structure the software in a hierarchical way that emulates the tasks division between *tactician* and *helmsman* on a real sailing boat. The first is responsible for the positioning of the boat on the course, while the second concentrates on driving the boat as fast as possible. To emulate this division, we build the main software architecture as shown in Figure 2.4. The *high level controller* acts like the *tactician*: based on path-planning, it sets the reference action $\alpha^\star$ and sends a steering command called *tack-now* when the boat should tack. The *low level controller* operates as the *helmsman*: it reads the commands sent by the *high level controller*, and, looking at the information from the sensors, computes the input actions (sail and rudder commands) to follow the desired reference. The rudder command is indicated by δ, while the sail command is represented by μ. The controllers implemented in the *low level controller* block are explained in Chapter 5 and in Chapter 6. In the former, the regulators in charge of tracking the reference $\alpha^\star$, supplied by the *high level controller*, are shown. In the latter, the controllers that execute

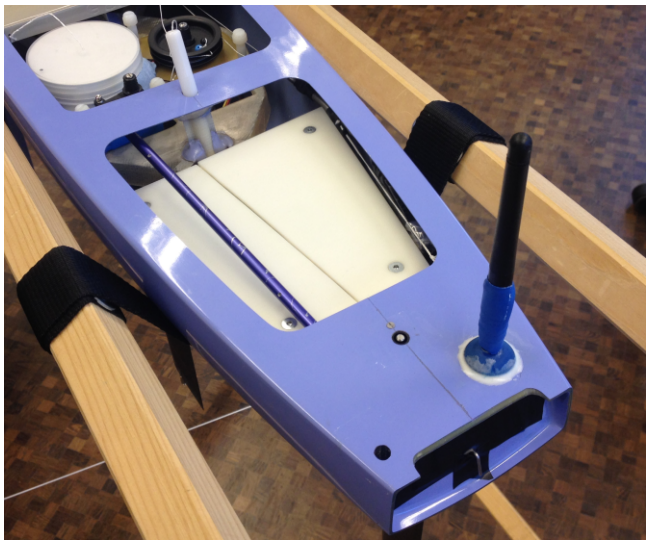


Figure 2.3: Printed 3D box mounted on the lower deck to store the electronics. The radio antenna has been mounted outside the lower deck.

the tack maneuver, when the *tack-now* command is sent, are explained.

To estimate the state of Aeolus (position, velocity, attitude, etc), we rely on an indirect extended Kalman filter (EKF) readily available in the PIXHAWK firmware and developed by the open source community. It has a tightly coupled compensation to integrate inertial measurements from IMUs (Inertial Measurements Units) and GPS positions, as explained in [27]. Moreover, this filter uses a general kinematic model, so no specific tuning based on the parameters of Aeolus is required. It is developed as a stand-alone application, that uses the GPS position and the IMUs data to compute its output. As soon as a valid GPS position is acquired, this application defines a local reference frame, which follows the NED (North-East-Down) convention. The main outputs of the EKF are referred to this local frame. The most used values, estimated by this filter, are the local position coordinates, the velocity of the vehicle and its attitude. All of these values refer to the local NED frame.

To collect the data from the weather station, another software application has been developed in the firmware. It sends the information collected to the other applications that require it.

In conclusion, these are the three applications which run in the firmware. Their main purposes are summarized next.

1. *parser_200WX*: collects data from the weather station and sends them to the other applications;

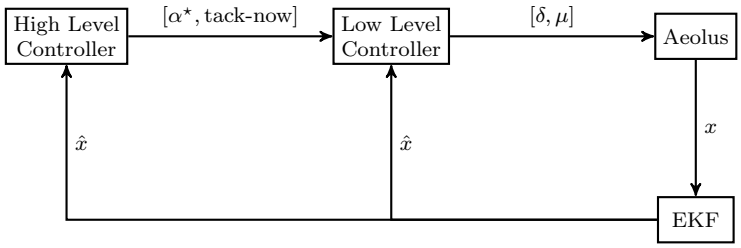


Figure 2.4: The software architecture emulates the tasks division on a real sailing boat. The *low level controller* acts as a *helmsman* and controls the rudder and the the sail. The *high level controller* acts as a *tactician* and sets the reference actions.

2. *autonomous_sailing*: it is the *helmsman* of the boat and computes the rudder and sail actions;

3. *path_planner*: it is the *tactician* of the vessel and sets the reference actions.

Chapter 3

Modeling and Identification

In this Chapter, we propose a simple technique to identify a linear state space model of the yaw dynamics relative to rudder commands. To achieve this result, we first exploit an ARX model, in order to assess the possibility of describing the yaw dynamics using a linear model. Finally, a Matlab GUI (Graphical User Interface) we have developed is shown; its role is to help a user to identify and validate several linear state space models.

In order to tune and analyze the closed-loop behavior induced by the controllers as explained in Chapter 5, a dynamical model of the sailboat is required. In [19] it is shown how to derive a nonlinear four-degree-of-freedom (4-DOF) dynamic model for a sailing yacht, including the roll dynamics, using the notation introduced in [28]. This procedure can in principle be applied to our model sailboat, but then the modeling phase must be followed by a parameter identification phase. Since identifying all the parameters of the nonlinear model is typically challenging, we decided to identify only the yaw and yaw rate dynamics of our sailboat using a linear model, as suggested in [17,18]. In [17] a continuous-time transfer function, from the rudder angle to the yaw rate output, is identified, which mathematically describes the input/output behavior of the system, meaning the dynamic behavior from the rudder input to the yaw rate. Once this model has been identified, a pole is added at the origin, so that a transfer function from the rudder to the yaw angle is also directly obtained. In [18] a discrete-time transfer function, also describing the yaw rate dynamics relative to the rudder angle input, is identified instead. Starting from these two works, we next describe a possible identification phase that can be executed by a "helmsman", using the remote controller to command the model sailboat. The boat sails upwind with a constant velocity, with the rudder being in the middle position; when the vessel has enough longitudinal speed, a step command on the rudder is given, that is, a strong steering input. This step command produces a fast

variation in the yaw rate and in the yaw angle of the boat, that are recorded and transmitted via the radio link to a PC located on the shore. A recorded identification maneuver, during a real test at Lake Zurich, is shown in Figure 3.1.

3.1 ARX Model

Using the data from the above identification phase, we identify a transfer function from the rudder command to the yaw rate, in a similar fashion of [17] and [18]. We choose to identify an Autoregressive model (ARX) for two main reasons: we want a discrete-time model since we use a discrete-time micro controller and the disturbances (wind and waves) act as control actions. We refer to the rudder signal as δ, to the yaw rate signal (the output of the system we want to identify) as ω and to the disturbance (wind and/or waves) as ε. A standard form for the ARX model is

$$A(q,\theta)\omega(t) = B(q,\theta)\delta(t) + \varepsilon(t), \tag{3.1}$$

where q is the time-shift operator, θ is the vector of parameter that describes our system and $A(q,\theta)$ and $B(q,\theta)$ are polynomials. Our purpose is to find the "best" vector θ, for any fixed q, such that the dynamical model in (3.1) matches as close as possible the experimental data. Usually, θ is estimated using a least square approach. Using this procedure, we identify several ARX models, using a different set of data collected during several tests at Lake Zurich. Each model describes the yaw rate response, as a function of the rudder command, using only one pole. In fact, it has been seen that using two or more poles, does not improve much the model response, therefore it does not seem beneficial to employ more than one pole. An example of one identified model is

$$(1 - 0.964z^{-1})\omega(t) = -0.035\delta(t). \tag{3.2}$$

In Figure 3.2, it is shown an example of cross validation of model (3.2). The fitting percantage shown, is computed using the following normalized root mean square error function:

$$fit = 100 \cdot \left(1 - \frac{\|x_{\text{ref}} - x\|}{\|x_{\text{ref}} - mean(x_{\text{ref}})\|}\right). \tag{3.3}$$

The data used to validate the model, are different from the ones used to identify the model.

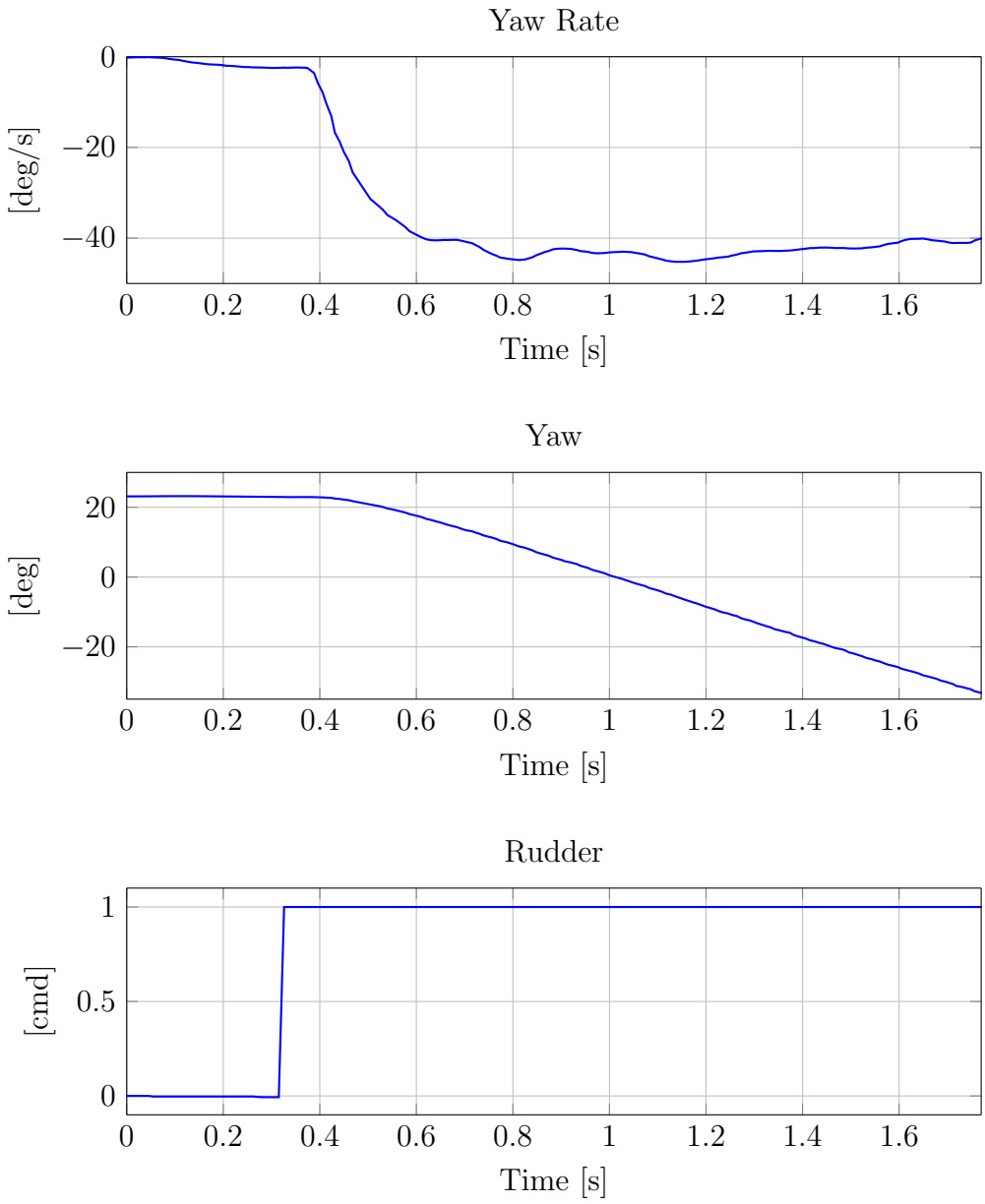
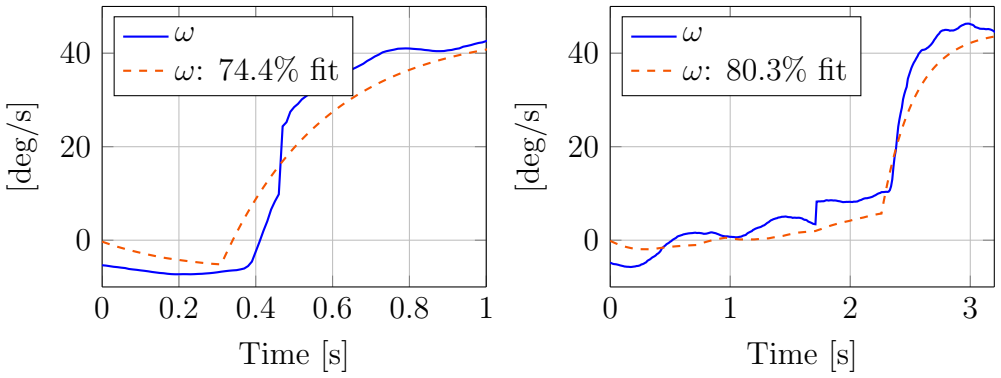


Figure 3.1: Identification maneuver: the first plot shows the yaw rate response, the second one shows the yaw response and the last one shows the rudder command given to AEOLUS while sailing.



(a) First cross validation (b) Second cross validation

Figure 3.2: ARX model from the rudder command to the yaw rate response, example of cross validation. Recorded yaw rate in solid blue, ARX response in dashed orange.

The identified ARX model shows that it is reasonable to describe the dynamic of yaw rate, using a linear model, at least when the boat is turning. This linear approximation, is relatively accurate during the initial phase of the turning maneuver, but starts to get worse the more time that elapses. In fact, the more the boat steers, the more the drag force and other nonlinear behaviors start to appear, and the linear approximation cannot closely follow the actual yaw rate dynamical evolution. This issue is shown in Figure 3.3. Since we aim at controlling AEOLUS both when sailing and tacking, we do not want these nonlinear dynamics to appear too much in the behavior of the vessel, because they make the boat slow down. Moreover, because hydrodinamics generated by the shape of the haul, sailing boats excel at sailing forward but they loose velocity when steering. Therefore, a steering maneuver should be executed only if either a desired course has to be followed or after the maneuver the boat can achieve a higher speed. In both cases, the maneuver should not overshoot the new desired heading. So, for our final purposes, we accept that our model is valid in the first phase of the turning maneuver. Since we want to control the yaw angle, we could add a pole on the unit circle in order to obtain a discrete time transfer function from the rudder command to the yaw angle. Even if the derivation of this transfer function is not challenging, especially from the ARX model (3.2), we would like to have a linear state space model for our design purposes. A first attempt to obtain it could be to find three matrices which describe the same dynamic in term of linear state space model and whose input/output transfer function is equal to the ARX model where a pole has been added on the unit circle.

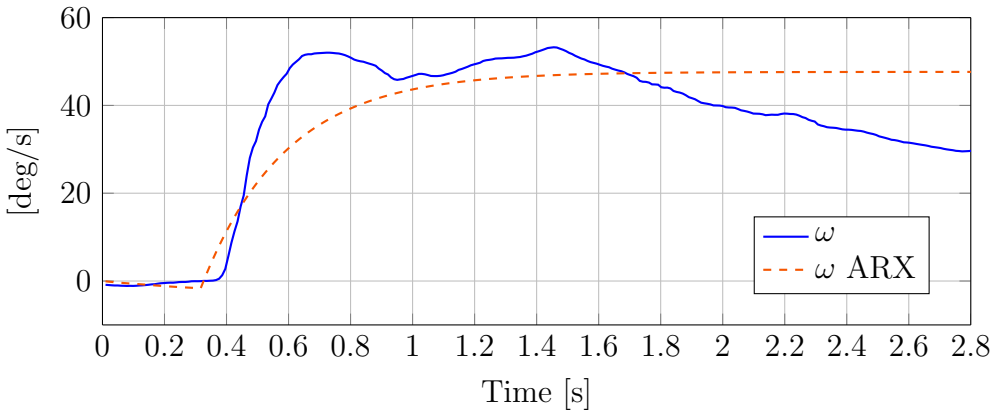


Figure 3.3: ARX model from the rudder command to the yaw rate response, example of the effect of the nonlinear dynamics. Here the rudder command was kept to one extreme position for the whole steering period. The more AEOLUS steers with a saturated rudder command, the more the nonlinear dynamics start to appear and the real behavior of the yaw rate differs from the one of the ARX model.

Even if it has been seen that the state space model obtained in this way can describe both the yaw rate and yaw dynamic, it is not clear the meaning of each variable that belongs to the state vector. Consequently, it is not easy to understand how to design and implement an estimator of the state, that is required by the controllers that are shown in Chapter 6. This is the main reason we follow a different way to obtain a state space representation. However, the ARX model shows that only one pole is sufficient to describe the yaw rate dynamic of the boat while steering, and this is the main field evidence used in the next section.

3.2 State Space Model

Using the results from the ARX model, we identify a state space description of the yaw and yaw rate dynamics. Specifically, we define the state vector at time k as

$$x_k = [\omega_k, \psi_k]^\top, \tag{3.4}$$

where ω is the yaw rate and ψ is the yaw, or heading, angle. We assume that the system dynamics can be described by the discrete-time linear system

$$x_{k+1} = Ax_k + B\delta_k. \tag{3.5}$$

Where x_k is the state of the boat and δ is the rudder command. A similar state space description has been shown in [20], where the unknown parameters are adaptively estimated. We use a slightly different approach, and estimate each parameter of the model without an adaptive procedure. The matrices in (3.5), which describe a two-state-space model, can be written as

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \; B = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}. \tag{3.6}$$

We identify two slightly different types of models: the *grey* and the *black* type models. In the *grey* type we use the knowledge about the physical meaning of the two variables in the state vector. We assume that the yaw rate ω (at time $k+1$) depends only on the previous yaw rate (at time k) and on the rudder command, thus in (3.6) we impose $a_{12} = 0$. Then, we assume that the yaw angle ψ (at time $k+1$) is the integral of the yaw rate and it is not directly affected by the rudder command. Therefore, in (3.6) we impose $a_{21} = \Delta_t$ (time interval between the time instants k and $k+1$), $a_{22} = 1$ and $b_2 = 0$. In this way, we are implicitly using the forward Euler method to integrate the yaw rate signal. It then follows that in the *grey* model we have to identify only the parameters a_{11} and b_1. In the *black* type model we do not make any prior assumption, thus we identify the full matrices A and B in (3.6).

By collecting the yaw rate, the yaw and the rudder signals experimentally, we follow a least square error procedure to compute the matrices A and B. In Chapter 8, it is shown how to compute both the *black* and *grey* type models. We have seen that both the *grey* and *black* type models can describe well the turning dynamics, but the latter one appears to be more general and environment-conditions independent. Numerically, the following *black* type model has been derived:

$$A = \begin{bmatrix} 0.7078 & -0.0124 \\ 0.0744 & 0.9986 \end{bmatrix}, \; B = \begin{bmatrix} -0.3089 \\ -0.0228 \end{bmatrix}, \tag{3.7}$$

where the yaw angle is meant in radians and the yaw rate is meant in radians/sec. The sampling time of this discrete-time model is $\Delta_t = 0.099$ s. We have validated the model in (3.7) using data collected in different navigation tests at Lake Zurich, and one example of validation is in Figure 3.4. It is important to point out that typically there are different wind and sea (wave) conditions between the day when a model is identified and the days when it is validated. We notice that despite the fit percentage for the yaw rate is not very high (57%), we achieve a quite good fit for the yaw response (88.2%). In Table 3.1 it is shown the average fitting percentages of the linear state

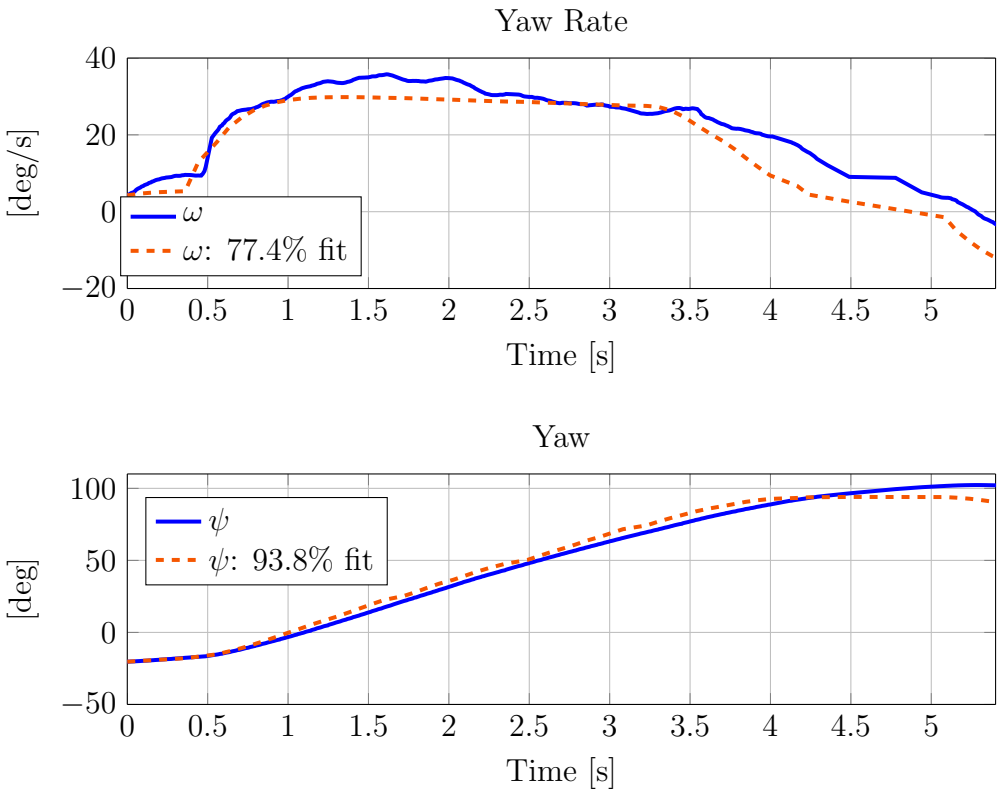


Figure 3.4: Linear state space model from the rudder command to the yaw rate and yaw angle responses, example of cross validation. Recorded data in solid blue, response of the model in dashed orange.

space model, computed using data obtained in several traials contucted at Lake Zurich. The fitting percentages have been computed using (3.3).

	yaw rate	yaw
average fit	79.7%	80.3%

Table 3.1: Benchmark of the linear state space model using average fitting obtained in 15 trials conducted at Lake Zurich.

3.3 Identification GUI

The models identified using real data, assuming either the ARX structure or the linear state space one, are heavily dependent on the daily conditions, such as strength of the wind or height of the waves. However, these conditions

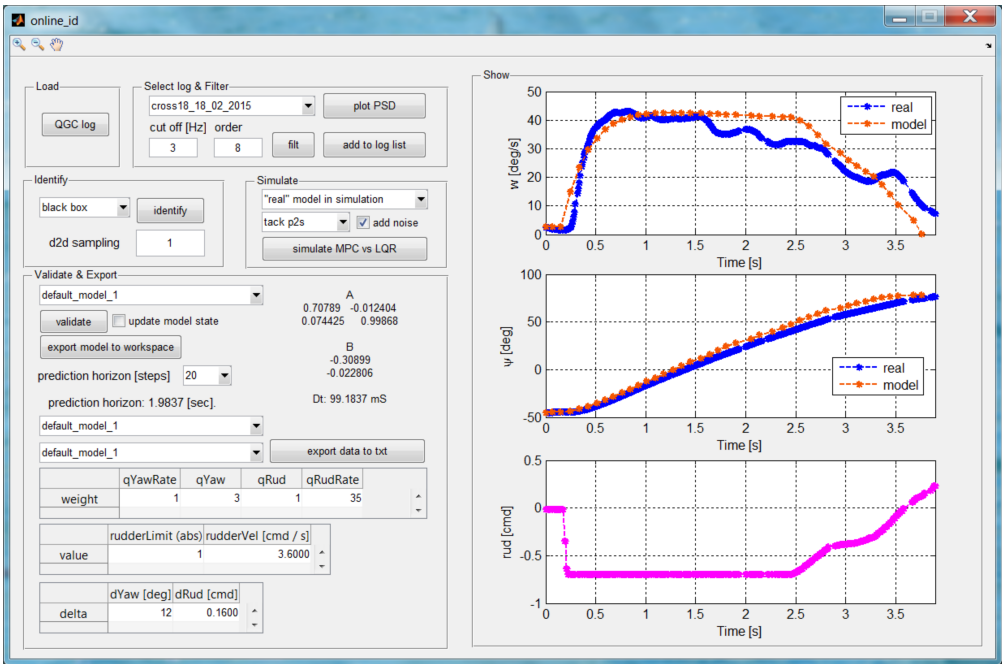


Figure 3.5: Matlab GUI used to identify a state space model, using the data transmitted by AEOLUS, while it is sailing.

are pretty much constant during the day, at least within a range of some hours. In order to have a numerical model, that can take into account the daily conditions, we designed the Matlab GUI (Graphical User Interface) depicted in Figure 3.5, that allows a user on the shore to identify a state space model. In fact, while AEOLUS is sailing, many data are transmitted over the radio link, such as the yaw angle, the yaw rate and the rudder command. With this GUI, a user on the shore can collect the data transmitted, while a "helmsman" is executing several identification maneuvers controlling the boat via the remote controller, and use them to identify several models. The user can select and plot the data collected, and decide which part should be used to identify a model, that can be either a *black* or a *grey* type linear state space model. Each identified model can be validated using the data recorded, so the user has an immediate feedback about how much the model is accurate. Once a satisfying model has been found, it can be used for further tuning purposes, as explained in Chapter 6.

Chapter 4

Collecting and Filtering Data

In this Chapter, we explain the main techniques we use to filter the data online, onboard of the PIXHAWK. We combine the main data provided by the EKF, with the wind information supplied by the weather station. This information is filtered, as explained in this Chapter, and used by the controllers implemented in Chapter 5 and Chapter 6.

The main output from the EKF, used here, is the heading angle ψ, that is, the compass direction where the bow is pointing to. The main data supplied by the weather station, are the estimated true wind direction σ, as well as the GPS course over ground χ. The course over ground is the direction over the ground the vehicle is currently moving in; it can be different from the heading, if there is drift (caused by either the wind or the waves), see Chapter 1. In order not to follow too much high-frequency wind shifts, we design a moving average filter for the raw measurement of the direction of the estimated true wind. The wind direction takes values between $-180°$ and $180°$, where $0°$ is the geographic North, $+90°$ is the East, $+90°$ is the West, etc. Note that the wind direction discontinuity at the beginning/end of the scale requires special processing to compute a valid mean value. We employ the single-pass procedure developed by Mitsuta in [29] to compute a mean wind direction, which is shown in detail in the Appendix. Figure 4.1 shows the effect of these filters on the raw wind direction measurement. Note that, this moving average filter introduces a delay between the raw measurement, and the final averaged value. This behavior heavily influences the performance of the *implicit* tack controller, explained in Chapter 6.

We now consider the heading angle α with respect to the wind direction, which reads as

$$\alpha = \psi - \sigma, \tag{4.1}$$

Setting a reference value $\alpha^\star$ for this angle, the *high level controller* tells Aeolus the desired orientation relative to the wind. For example, if $\alpha^\star = 45°$ Aeolus

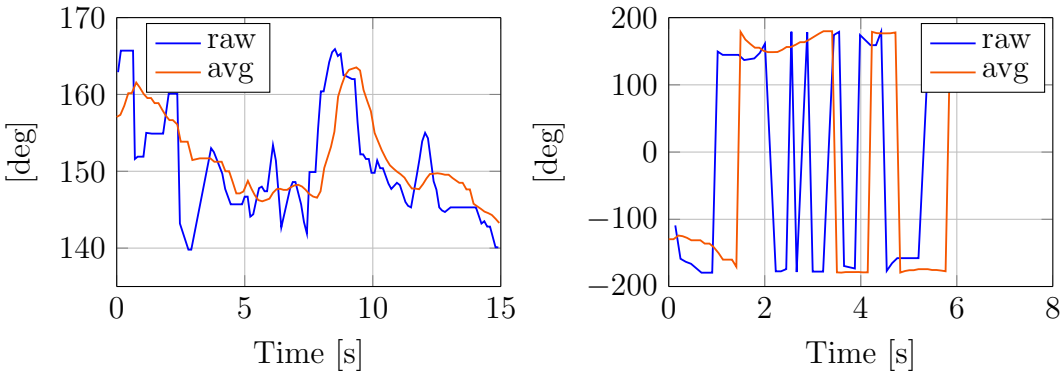


Figure 4.1: True wind direction from the weather station, in blue, and averaged value from the filter, in orange. On the left an example of the delay introduced by the moving average filter and how this filter can smooth rapid shifting peaks. On the right the mean value of the wind direction when the raw measurement switches between $-180°$ and $180°$, obtained by using the Mitsuta mean.

should sail upwind (that is, between "close hauled" and "beam reach"), if $\alpha^\star = 90°$ Aeolus should sail at "beam reach", etc. The sign of $\alpha^\star$ determines the haul: a positive value corresponds to starboard haul (the wind is blowing from the right side of the vessel), meanwhile a negative value corresponds to port haul (the wind is blowing from the left side of the vessel). An example is depicted in Figure 4.2. In order to overcome drift due to currents and waves, it is possible to use the course over ground value χ, provided by the GPS signal, instead of the heading ψ, to compute α. Unfortunately, the GPS signal sometimes drops, even for long periods of time, up to 10 seconds, especially during fast steering maneuvers. Therefore, here we define two angles to take into account both the drift compensation and the updated measurements:

$$\alpha_\psi = \psi - \sigma \tag{4.2}$$

$$\alpha_\chi = \chi - \sigma. \tag{4.3}$$

Our estimated α angle is computed as a convex combination between these two angles as follows:

$$\alpha = (1 - \lambda)\alpha_\chi + \lambda\alpha_\psi \tag{4.4}$$

where λ is a function of the last time when the course-over-ground (cog) signal has been updated, that takes values in $[0, 1]$. Specifically, λ is computed as

$$\lambda(t) = \begin{cases} \frac{t - t_{\text{last\_cog}}}{T_{\text{threshold}}} & \text{if } t - t_{\text{last\_cog}} \leq T_{\text{threshold}} \\ 1 & \textit{otherwise} \end{cases}. \tag{4.5}$$

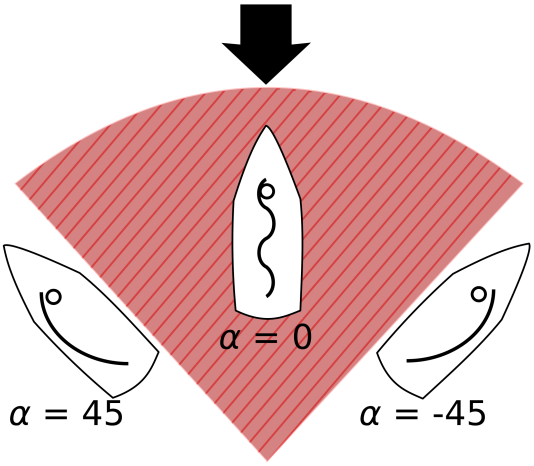


Figure 4.2: Heading with respect to the wind: graphical explanation of the α angle. The black arrow indicates the direction from which the wind is blowing. The red area is called "no-go zone".

Where the constant $T_{\text{threshold}}$, can be set by the user while AEOLUS is sailing. Computing λ as (4.5), the more time has elapsed without a course-over-ground update, the more λ tends to 1; when a new course over ground is received, λ is then reset to 0. Since this design can cause discontinuities in the α estimate when a new course over ground is received, we employ a moving average filter on the α value to smooth out the actual estimate. The effect of the convex combination, combined with a average filter, and the difference between the final α and α_χ and α_ψ, is depicted in Figure 4.3.

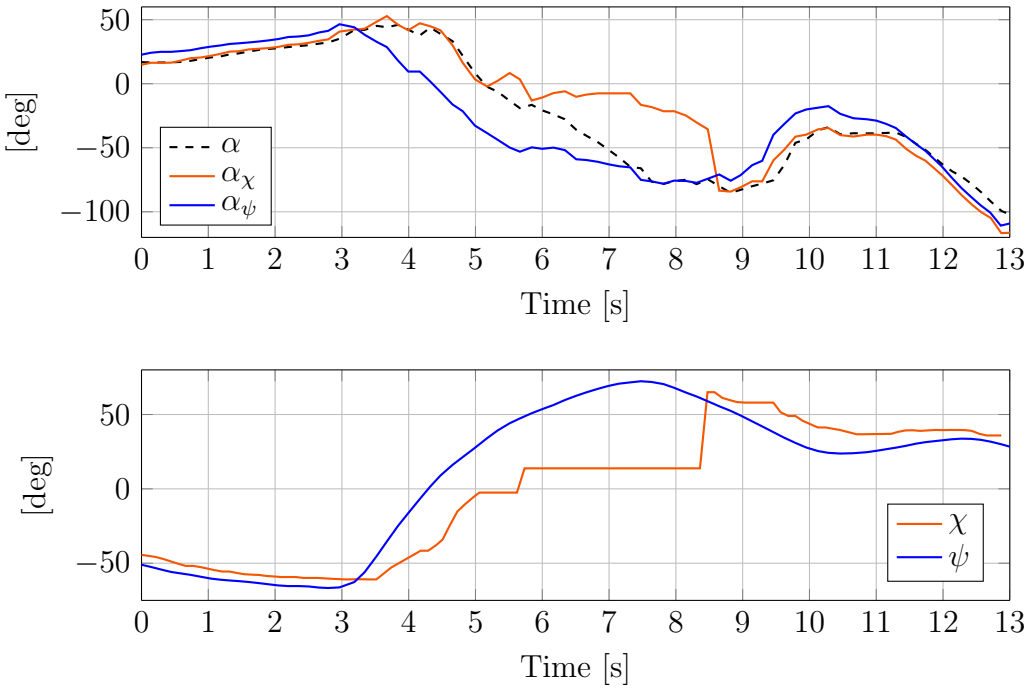


Figure 4.3: Effect of the convex combination in the computation of the final value of α. The upper plots shows the difference between α_χ (orange), α_ψ (blue), and the final α (black). The lower plot shows where the GPS signal was lost, that is, where the χ is constant.

Chapter 5

Tracking a Constant Heading

In this Chapter, we define two main controllers, a standard proportional (P) controller and a more sophisticated nonlinear one (NL), each of them can be selected to take care of the rudder command. These two controllers are implemented in the *low level controller* block, shown in Chapter 2. Finally, a simple rule based controller for the sail, is explained at the end of the Chapter.

5.1 Rudder Controller

The reference heading angle for the low regulator is $\alpha^\star$, and is set by the *high level controller*. Using the equation (4.1), it is possible to specify either a constant compass course or a constant heading to the wind as reference. The first case is simply obtained by setting $\sigma = 0$; the second case uses a more general formulation where σ is provided by the weather station. Let use normalize the rudder command δ to be in the range $[-1, 1]$. Given the reference angle $\alpha^\star$ and the current estimate α, the heading error reads as $e := \alpha^\star - \alpha$, and the P rudder controller sets the rudder command δ as

$$\delta = \delta_{\mathrm{P}}(e) = k_{\mathrm{p}}\, e, \tag{5.1}$$

for some $k_p > 0$.

Instead, the NL controller defines a nonlinear gain $k(e)$ as

$$k(e) = \frac{k_{\mathrm{p}}}{1 + c_{\mathrm{p}}|e|} \tag{5.2}$$

for some $k_{\mathrm{p}}, c_{\mathrm{p}} > 0$, and hence sets the rudder command to

$$\delta = \delta_{\mathrm{NL}}(e) = k(e)\, e. \tag{5.3}$$

The main idea of this nonlinear controller was first introduced in [30]. The controller (5.2) acts as a proportional gain when the error e is small, but its behavior changes when the error is large. The c_{p} constant can be in fact used to tune the control action when the error is large; here we tune (5.2) so that the larger c_{p} is, the smaller the gain $k(e)$ and so the rudder action. This behavior is used also in Chapter 6 to execute a special tack maneuver. These two controllers are able to track a reference angle. An example of tracking the reference $\alpha^\star$, using the NL controller is depicted in Figure 5.1. Note that, since the controller is working "near" the reference, applying the P regulator would have provided a similar behavior.

Using the identified numeric model in (3.7), we can study the stability of the closed-loop system, both in the linear and the nonlinear case. If we use the P controller (5.1), the closed loop system is a linear discrete-time model, whose stability can be studied applying the Jury stability criterion [31]. The Jury criterion in the discrete-time equivalent of the more famous Routh-Hurwitz stability criterion for continuous-time systems. In general, a closed loop system has got the following characteristic equation:

$$P(z) = a_0 + a_1 z + a_2 z^2 + \cdots a_N z^N \tag{5.4}$$

There are four tests to be performed, to check the stability of the system.

1. $P(1) > 0$
2. $(-1)^N P(-1) > 0$
3. $|a_0| < a_n$, if rule $1, 2$ and 3 are satisfied, the Jury array must be constructed
4. The Jury Array must satisfy:

$$\begin{aligned} |b_0| &> |b_{N-1}| \\ |c_0| &> |c_{N-2}| \\ |d_0| &> |d_{N-3}| \end{aligned}$$

   and so on until the last row of the array.

If all these conditions are satisfied, the system is stable. The Jury array is constructed in a recursive way, by filling the table:

a_0	a_1	a_2	a_3	$\cdots$	a_N
a_N	$\cdots$	a_3	a_2	a_1	a_0
b_0	b_1	b_2	$\cdots$	b_{N-1}	
b_{N-1}	$\cdots$	b_2	b_1	b_0	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

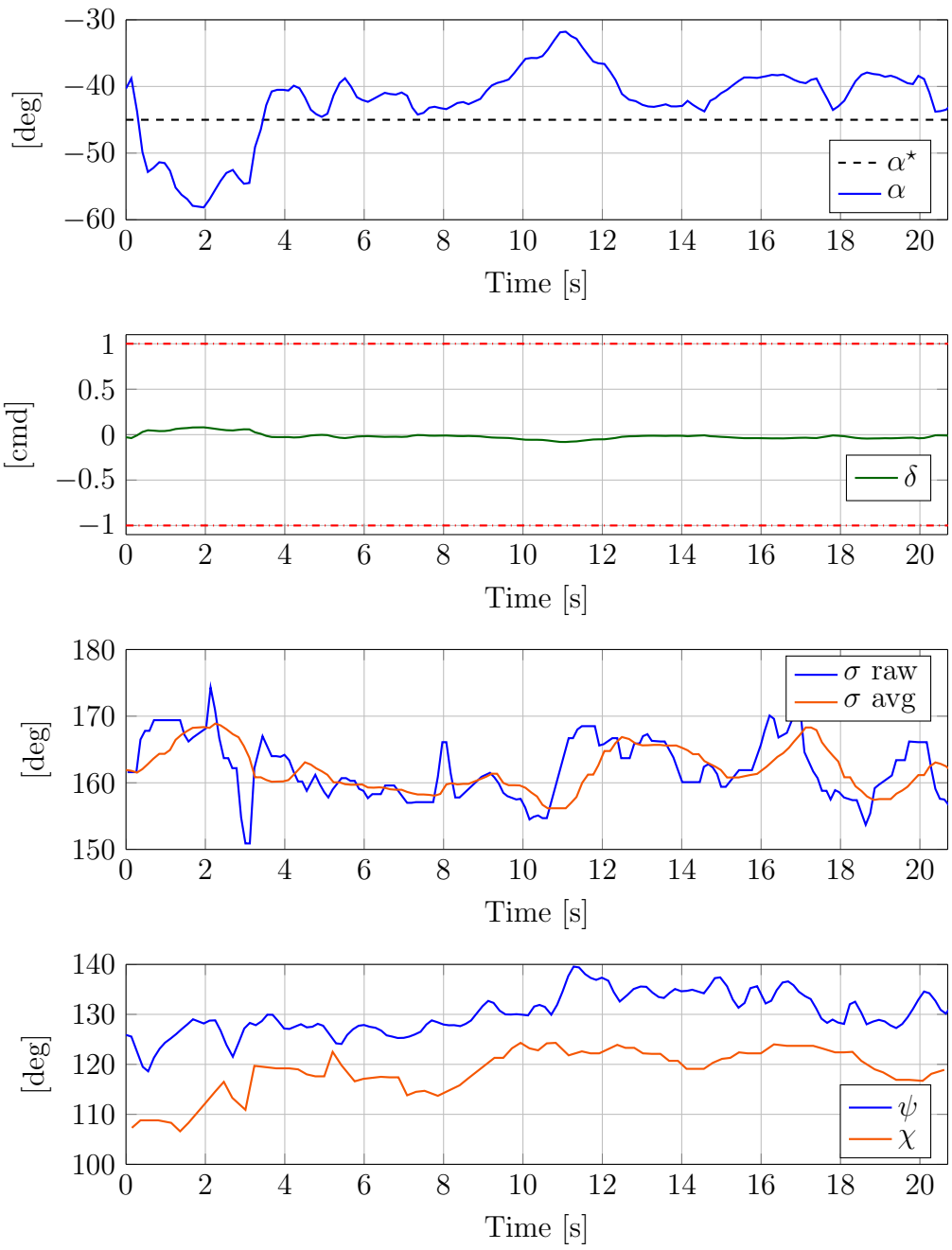


Figure 5.1: Example of tracking a reference $\alpha^\star$, using the nonlinear controller. The first plot shows the reference $\alpha^\star$ in dashed black, and the obtained α angle in blue. The second plot shows the rudder command in green and the rudder limits in dotted red. The third plot shows the true wind direction, in blue the raw data and in orange the average value. The fourth plot shows the heading angle in blue and the course over ground angle in orange, note the difference caused by the drift.

where the following formula can be applied recursively to construct each element of the array:

$$b_k = \begin{vmatrix} a_0 & a_{N-k} \\ a_n & a_k \end{vmatrix}$$
$$c_k = \begin{vmatrix} b_0 & b_{N-1-k} \\ b_{N-1} & b_k \end{vmatrix}$$

This criterion is applied on the numerical model (3.7), where the rudder command is computed by (5.1). The characteristic equation, in our case, is given by

$$\begin{aligned} P(z) &= z^2 + p_1 z + p_0 \\ p_1 &= -b_2 k_\mathrm{p} - a_{22} - a_{11} \\ p_0 &= (-a_{21} b_1 + a_{11} b_2) k_\mathrm{p} + a_{11} a_{22} - a_{12} a_{21} \end{aligned} \tag{5.5}$$

By checking the previous four rules, we obtain that

1. $P(1) > 0$ if $k_\mathrm{p} > -0.0442$
2. $P(-1) > 0$ if $k_\mathrm{p} < 214.02$
3. $|p_0| < p_n$, if $-249.22 < k_\mathrm{p} < 42.629$
4. The Jury Array satisfies the fourth rule if either $-0.0442 < k_\mathrm{p} < 42.629$ or $42.629 < k_\mathrm{p} < 214.02$

In order to satisfy these four rules, we must have $-0.0442 < k_\mathrm{p} < 42.629$, and since we want a positive value for k_p, at the end we obtain that the closed loop system, controlled using the P regulator, is stable if

$$0 < k_\mathrm{p} < 42.629 \tag{5.6}$$

After several experimental tests, we tune k_p for both stability and tracking purposes to the value $k_\mathrm{p} = 0.35$.

Based on the results from the P controller, we tune the NL one in (5.2) with $k_\mathrm{p} = 0.35$, $c_\mathrm{p} = 0.35$. In this way, the nonlinear rudder command is similar to the one obtained with the P controller if the heading error is small, but the control action is limited as desired when the heading error is large. For instance, using the previous settings, an error $e = 90°$ produces a rudder command $\delta = 0.55$ with the P controller and a rudder command $\delta = 0.35$ with the NL one (the error used to compute δ must be expressed in radians). If we use the nonlinear controller, we obtain a nonlinear closed loop system,

which has got an equilibrium point in the origin that is locally stable. The error e, in (5.2), can be seen as the value of the yaw angle ψ, that is $x(2)$, according to (3.4) ,which must be controlled to 0. The closed loop system is

$$x_{k+1} = f_{\text{cl}}(x_k) = Ax_k + B\delta_{\text{NL}}(x(2)) \tag{5.7}$$

The system (5.7) is autonomous, thus $x = 0$ is an equilibrium point where we can linearize the system. The transition matrix of the linearized system is

$$A^{\text{CL}}_{k_\text{p},c_\text{p}} = \left.\frac{\partial f_{\text{cl}}}{\partial x}\right|_{x=0} = A + B\frac{\partial}{\partial x}\left(\delta_{\text{NL}}(x(2))\right)\Big|_{x=0} \tag{5.8}$$

Since in (5.2) there is the absolute value of the error $x(2)$, which is not necessarily differentiable in $x(2) = 0$, we must take into account both the right and the left derivative. Right derivative:

$$\lim_{\epsilon\to 0^+}\frac{\delta_{\text{NL}}(\epsilon)-\delta_{\text{NL}}(0)}{\epsilon} = \lim_{\epsilon\to 0^+}\frac{\frac{k_\text{p}}{1+c_\text{p}\epsilon}\epsilon}{\epsilon} = \lim_{\epsilon\to 0^+}\frac{k_\text{p}}{1+c_\text{p}\epsilon} = k_\text{p} \tag{5.9}$$

Left derivative:

$$\lim_{\epsilon\to 0^-}\frac{\delta_{\text{NL}}(\epsilon)-\delta_{\text{NL}}(0)}{\epsilon} = \lim_{\epsilon\to 0^-}\frac{\frac{k_\text{p}}{1-c_\text{p}\epsilon}\epsilon}{\epsilon} = \lim_{\epsilon\to 0^-}\frac{k_\text{p}}{1-c_\text{p}\epsilon} = k_\text{p} \tag{5.10}$$

So, by linearizing the system in $x = 0$, we obtain in each case

$$A^{\text{CL}}_{k_\text{p}} = A + [0_{2\times 1}, B]k_p, \tag{5.11}$$

which is the same closed loop transition matrix that is obtained using the P controller. So, if k_p satisfies (5.6), then the origin is an equilibrium point that is locally stable. In Figure 5.2 there are displayed the state trajectories sampled simulating the numerical model (3.7) using the NL controller (5.2). Every sample trajectory, starting "close" from the origin, converges to the locally stable equilibrium point.

5.2 Sail Controller

We decouple the control of the rudder and the sail, similarly to [16,17]. Since at the moment of the development of this project, no feedback measurements of the position of the sail were available, a simple open loop controller for them has been implemented. This controller is developed as a rule based regulator: the more AEOLUS is sailing opposite to the wind direction, that is, the closer to 0° α is, the more we close the sail. This simple idea, is coded

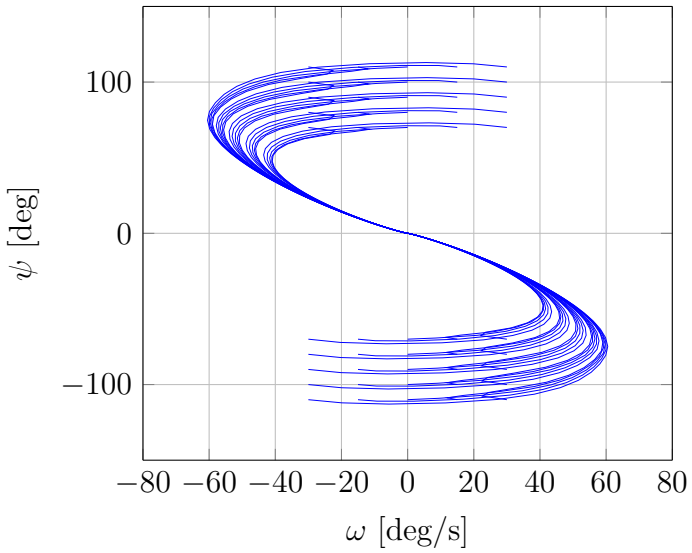


Figure 5.2: Sampled state trajectories, obtained simulating the system behavior when the NL controller is applied.

in the firmware of the PIXHAWK, using a list of "*if-then-else*", that describes the behavior shown in Figure 5.3. The parameters x_1 and x_2, shown in the last Figure, can be changed at run time, while AEOLUS is sailing, by an user, using QGROUNDCONTROL.

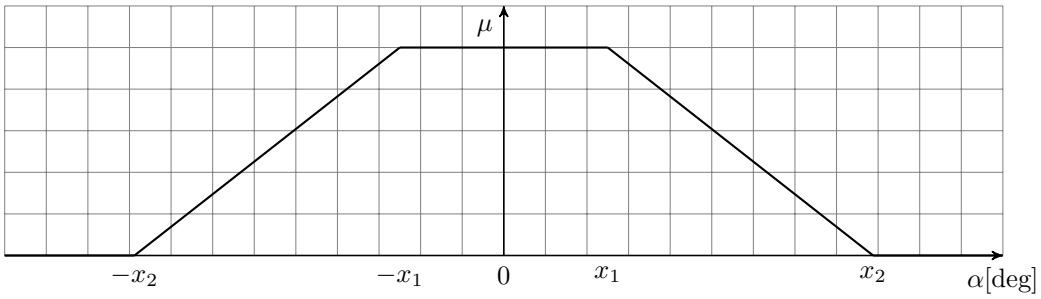


Figure 5.3: Rule based controller for the sail. The more AEOLUS is sailing opposite to the wind direction, that is, the closer to 0 α is, the more we close the sail. The parameters x_1 and x_2 can be changed at run time, while AEOLUS is sailing. The higher the sail command μ, the more the sail is closed.

Chapter 6

Tacking Maneuvers

In this Chapter, we derive several controllers to execute the tack maneuver. Tacking is a sailing maneuver by which a sailing vessel turns its bow into the wind through the "no-go zone" so that the direction from which the wind blows changes from one side to the other. An example of a tack maneuver is shown in Figure 6.1: before tacking the boat is sailing at port haul (wind from the left side); after the maneuver the vessel is sailing at starboard haul (wind from the right side). Since during the maneuver the boat crosses the "no-go zone", it should be executed in the fastest and smoothest possible way, in order not to get stuck against the wind. First of all, we try to exploit the knowledge of the "helmsman" and implement a rule based controller for the rudder. Second, we take advantage of the controllers implemented in Chapter 5 and we use them as the simplest way to execute a tack. Third, we improve the performance of the previous controllers, by designing three new rudder regulators that compute the command only during the steering maneuver.

Five possible ways of carrying out a tack are developed and tested: the *helmsman*, the *implicit*, the *dedicated*, the *LQR* and the *MPC* one. The *implicit* maneuver does not require a switch from the rudder controller used during upwind sailing, meanwhile the others require an ad-hoc rudder regu-

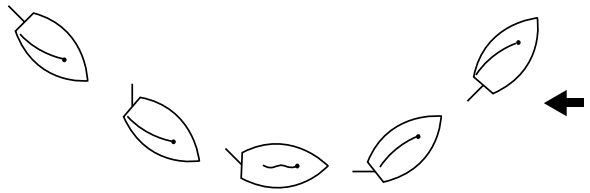
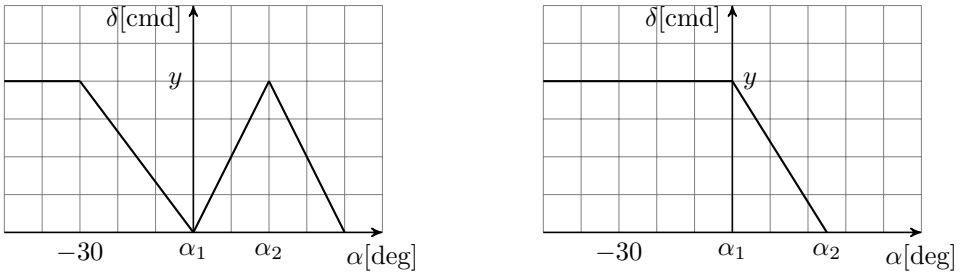


Figure 6.1: Example of a tack maneuver. The black arrow shows the wind direction.



(a) *Helmsman* type 1 (b) *Helmsman* type 2

Figure 6.2: *Helmsman* rules to execute a tack maneuver. The parameters α_1 and α_2 determine the shape of the command. The parameter y sets the maximum amplitude of the rudder input.

lator to be executed during the tack.

6.1 Helmsman Tack

Our first attempt is to emulate what a real helmsman would do to execute a tack maneuver. After speaking with a helmsman, we can write two different rudder commands to execute a tack: the first one is shown in Figure 6.2a and the second one is shown in Figure 6.2b. The first rudder command is similar to the one a helmsman would use on a real sailing boat, meanwhile the second one is something easier. In both cases, there are three parameters, α_1, α_2 and y, that can be tuned by a user on the shore, while AEOLUS is sailing, in order to better exploit the controller capabilities.

Both type of tacks, *helmsman* 1 and 2, are implemented in the firmware of the PIXHAWK and tested in several trials at Lake Zurich. One of these two controllers can be selected by a user on the shore, and it is executed when the *high level controller*, shown in Chapter 2, sends the *tack-now* command. These two tacks can be seen as fuzzy controllers to execute a steering maneuver, since they exploit the knowledge of the helmsman and express it in terms of rules. In this case the rudder command δ is a function of the heading relative to the wind α. We develop these two rudder controllers in the firmware using a list of nested "*if-then-else*", instead of using the common way to implement a fuzzy controller. Since the rudder command is computed online the PIXHAWK, the choice of a list of "*if-then-else*" is required because of the small amount of memory and computation power available. After several tests at Lake Zurich, where many values for the parameters α_1, α_2 and y were tried, the best responses we could achieve are depicted in Figure 6.3. In

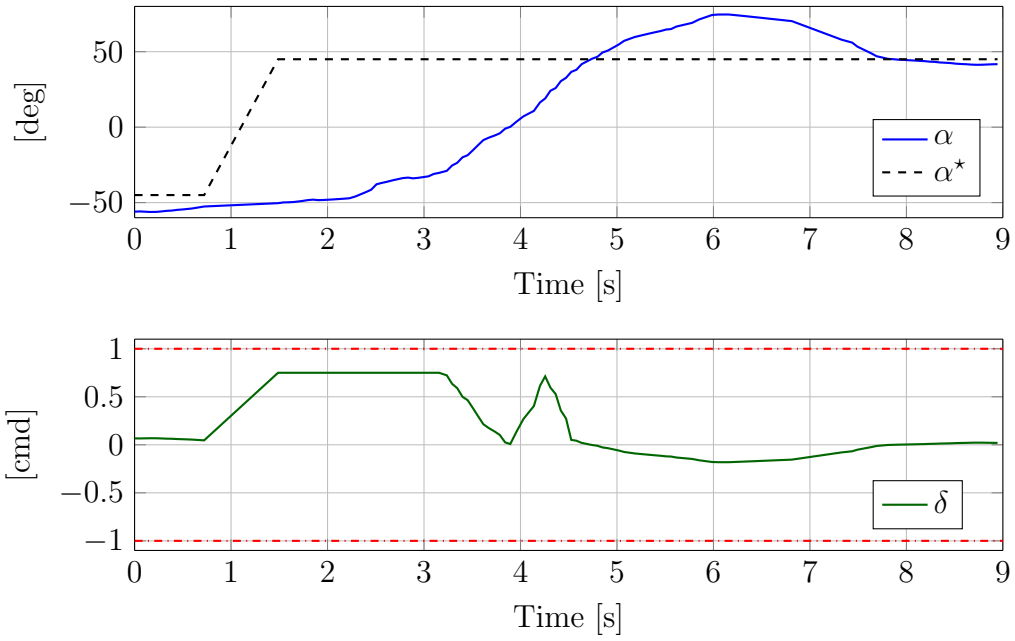
both *helmsman* 1 and 2, there is a big overshoot between the desired heading to the wind, $\alpha^\star$, and the obtained one, α. So, our first attempt to perform a tack as a helmsman would do, by employing a rule based controller, failed to execute a good steering maneuver. This result makes us thinking about more sophisticated control laws, that can execute a good tack maneuver, as shown in the next sections.

6.2 Implicit Tack

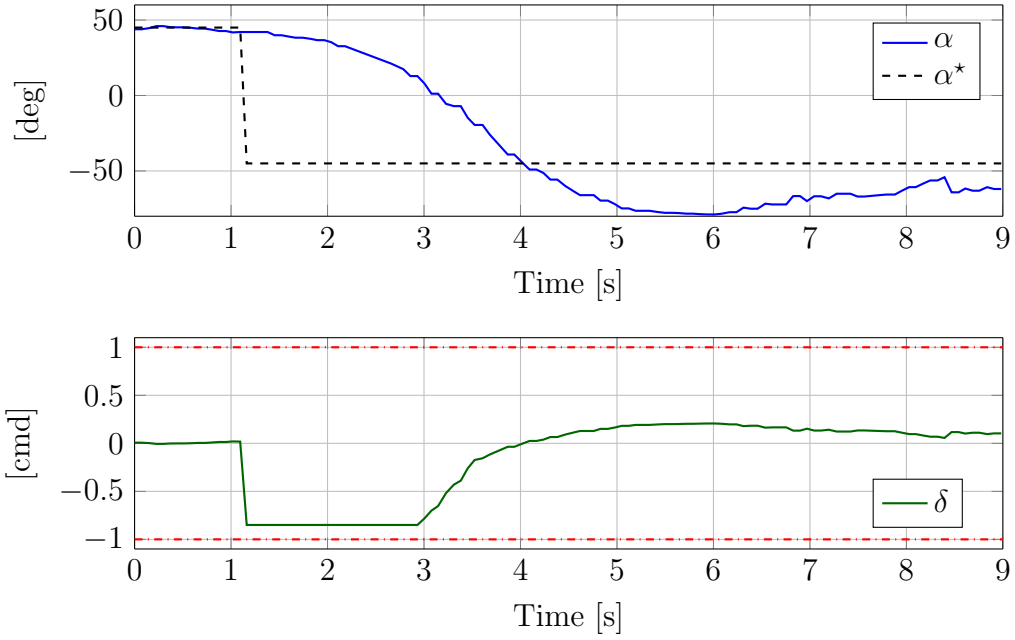
The implicit tack is done if the *high level controller* simply changes the reference $\alpha^\star$ of the low level heading controller, for example from $-45°$ to $45°$.In this way, the *low level* application does not really "know" that a tack maneuver must be done. The new reference is followed by the rudder controller that is on, either the P controller (5.1) or the NL one (5.2). We have noticed that the implicit tack done using the P controller produces larger overshoot, compared to the NL implicit one. Two main reasons cause undesired overshoots: a strong initial rudder command and the delay introduced by the average filters (applied to the wind direction and to the heading relative to the wind) shown in Chapter 4. When the reference is changed, the P controller sets a more aggressive rudder command, compared to the NL controller, in which the c_p coefficient is actually tuned to be less aggressive when the error is large. Moreover, the moving average filters used to filter the raw measurements introduce a delay of about 2 seconds, thus if the tack maneuver is executed too fast, for example by using an aggressive rudder command, the delay introduced by the average filters induces an overshoot in the heading angle response. To avoid this overshoot, a less aggressive implicit tack maneuver should be executed. All of these considerations can be seen in Figure 6.4, where it is shown a comparison between the *implicit* P tack and the *implicit* NL one. Summarizing, our NL controller can both control the rudder to sail upwind at constant reference and execute a smooth tack maneuver without significant overshoot. This result is achieved without any further special action during the maneuver.

6.3 Dedicated Tack

The *dedicated* tack maneuver, as well as the *LQR* and the *MPC* one, are executed by the *low level controller* when the *high level* layer sends the *tack-now* command and updates $\alpha^\star$. Therefore, the *low level* application is now aware that a tack must be carried out, and can hence perform special actions

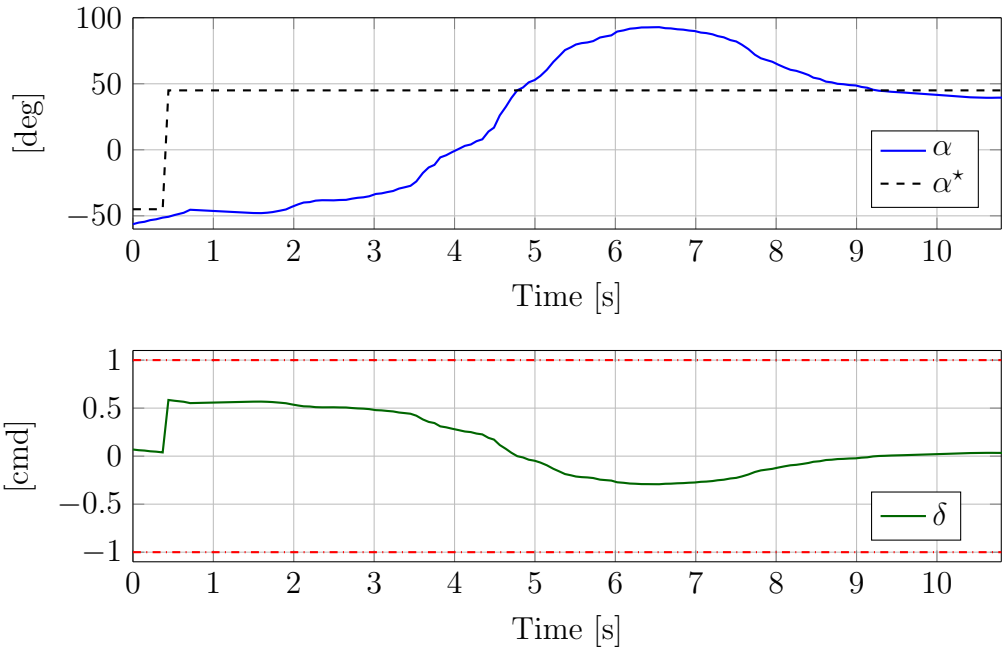


(a) *Helmsman* type 1, tack from port to starboard haul.

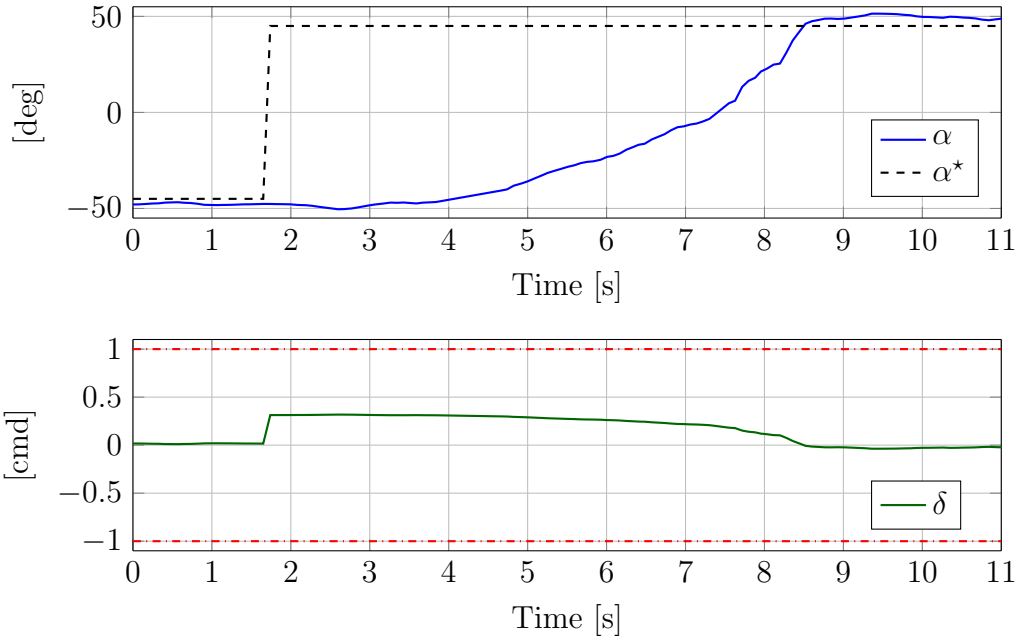


(b) *Helmsman* type 2, tack from starboard to port haul.

Figure 6.3: *Helmsman* tack, type 1 and 2. The plots show the reference $\alpha^\star$ in dashed black, the heading relative to the wind α in blue, the rudder command δ in solid green and the rudder limits in dotted red.



(a) *Implicit* tack, using the P controller, from port to starboard haul.



(b) *Implicit* tack, using the NL controller, from port to starboard haul.

Figure 6.4: *Implicit* tack. The plots show the reference $\alpha^\star$ in dashed black, the heading relative to the wind α in blue, the rudder command δ in solid green and the rudder limits in dotted red.

for the maneuver. From our field experience, we mention three critical actions to be taken into account when tacking:

1. the window size of two average filters (on the wind direction σ and on the heading relative to the wind α angle) is set to 1 sample only;
2. the α angle is computed using only α_ψ in (4.2), not by using α_χ in (4.4);
3. a specialized tack regulator takes control of the rudder during the maneuver.

The first action overcomes the undesired delay typical of the *implicit* tack caused by the filters. The second action is needed because during a fast tack maneuver, it is likely to loose the updated course-over-ground measurement. In our implementation, a specialized tack regulator controls the rudder until the tack is considered completed, that is until the error signal is within specified bounds for about 1 second. Namely, we then switch back to the course-navigation controller when the actual heading is close enough to the new reference. We consider a *dedicated* tack maneuver completed if and only if the following holds for at least $\bar{T}$ seconds, where $\bar{T}$, $\bar{\alpha}$ and $\bar{\delta}$ can be set by a user while AEOLUS is sailing:

$$|\alpha^\star - \alpha| \leq \bar{\alpha} \quad \text{and} \quad |\delta| \leq \bar{\delta}. \tag{6.1}$$

In real tests we set $\bar{T} = 0.8\,s$, $\bar{\alpha} = 12^\circ$ and $\bar{\delta} = 0.15$. In this way, we switch from the *dedicated* controller to the normal one, used for tracking $\alpha^\star$, when Aeolus is "close" enough to the new reference.

Let us now discuss in detail our *dedicated* tack maneuver. The *dedicated* controller is just the nonlinear one in (5.2), with $k_\mathrm{p} = 0.7366$ and $c_\mathrm{p} = 0.1$ tuned to obtain a more aggressive behavior and execute a faster, as well as smooth, tack. With our numerical choice, when the error is $|e| = 90^\circ$ we have $|\delta| = 1$, thus the rudder is working at its saturation boundary. Figure 6.5 shows how the *dedicated* tack behaves, during a tack from port to starboard haul. We point out that this more aggressive regulator does not produce overshoot thanks to the special actions explained above.

6.4 LQR Tack

The fourth steering controller implemented is a Linear Quadratic Regulator (LQR), where "optimality" is with respect to an infinite-horizon cost function. The LQR is a state-feedback controller, meaning that the control action

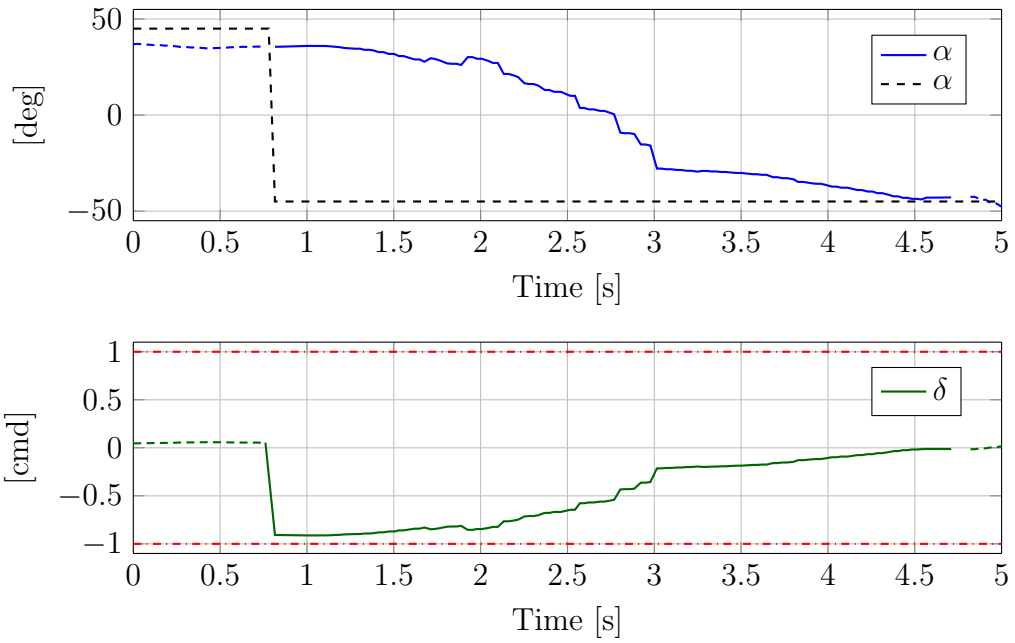


Figure 6.5: *Dedicated* tack from starboard to port haul. The plots show the reference $\alpha^\star$ in dashed black, the heading relative to the wind α in blue, the rudder command δ in solid green and the rudder limits in dotted red. Where the *dedicated* regulator is not controlling the rudder, the α and the δ lines are dashed. The *tack-now* command has been received at time 0.88 s; from that moment, the *dedicated* regulator takes care of the rudder, until the tack maneuver is not considered completed, that is around time 4.7 s.

is a static feedback law of the state x as follows: $\delta = \delta(x) = K_{\text{LQR}}x$. In our specific application, we aim at computing the optimal rudder command δ to carry out a fast and smooth maneuver. To exploit the model derived in the modeling section in Chapter 3, we make the assumption that the true wind direction σ does not change during the tack maneuver, which is a practically reasonable assumption if the maneuver is fast enough. In this way, a tack maneuver can be just seen as a change in the heading angle ψ. For example, a tack from port ($\alpha^\star = -45°$) to starboard haul ($\alpha^\star = 45°$), can been seen in two equivalent ways: (a) we require a change in the α angle of $\Delta\alpha = 90°$; (b) we require a change in the heading angle ψ of $\Delta\psi = 90°$, i.e., $\Delta\psi = \Delta\alpha$, based on (4.1) assuming a constant wind-direction angle σ. Thus, tacking results in steering the state of the system in (3.4) from the initial value $x_{\text{i}} = [\omega_{\text{i}}, \Delta\psi]^\top$ to the final value $x_{\text{f}} = [0, 0]^\top$, where the latter consists in achieving the desired $\alpha^\star$ with zero yaw rate. Let us rewrite the model in (3.4) in state-space form with

$$\hat{\delta}_k := \delta_k - \delta_{k-1} \tag{6.2}$$

$$\hat{x}_k := [\omega_k, \psi_k, \delta_{k-1}]^\top, \tag{6.3}$$

where $\hat{\delta}$ is the new control input and $\hat{x} \in \mathbb{R}^3$ is the extended state. Namely, the extended state at time k, $\hat{x}_k$, contains the yaw rate and yaw angle at time k and the rudder command δ_{k-1} injected into the system at the previous step $k-1$. The input $\hat{\delta}$ is the difference between the actual rudder command at time k, and the previous one; in other words, the real rudder command δ provided at the time k is then $\delta_k = \delta_{k-1} + \hat{\delta}_k$. Since the LQR problem does not take into account any constraint, such as the rudder saturation limits, the final command given to the rudder is a clipped version of the optimal one computed, in order to respect the actuator limits. The state space matrices $\hat{A}$, $\hat{B}$ corresponding to the extended state dynamics become

$$\hat{A} := \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}, \; \hat{B} := \begin{bmatrix} B \\ I \end{bmatrix}. \tag{6.4}$$

This state-vector extension allows us to define the following LQR problem:

$$\begin{cases} \min & \displaystyle\sum_{k=0}^{\infty} \hat{x}_k^\top Q \hat{x}_k + r\hat{\delta}_k^2 \\ \text{s.t.} & \hat{x}_{k+1} = \hat{A}\hat{x}_k + \hat{B}\hat{\delta}_k, \quad \forall k \in \mathbb{N} \\ & \hat{x}_0 = \hat{x}(t) \end{cases} \tag{6.5}$$

where the matrix $Q \succ 0$ and $r > 0$ are design choices, $\hat{x}$ is the extended state and $\hat{\delta}$ is the input command of the extended system. The LQR gain

K_{LQR} is computed such that the state feedback law $\hat{\delta}(\hat{x}(t)) = K_{\text{LQR}}\hat{x}(t)$ solves the problem (6.5), subject to the unconstrained discrete-time dynamics $\hat{x}_{k+1} = \hat{A}\hat{x}_k + \hat{B}\hat{\delta}_k$. Moreover, the LQR gain K_{LQR} ensures that the closed loop system is stable. The main reason to extend the state vector as in (6.2) is to assign a cost penalty to the control action δ as well as to the control variation $\hat{\delta}$. Once we obtain the matrices of the extended model in (6.4) using the values in (3.7), we can tune Q and r, both via numerical simulations and via field tests. We tested several values, both for Q and for r, and we saw that the best configuration for these parameters is given by

$$Q = \text{diag}(1, 3, 1), \; r = 35. \tag{6.6}$$

With this choice, we sensibly penalize fast variation in the rudder command, which should not vary with too much velocity and should avoid "chattering". Moreover, we give a higher priority in controlling the yaw angle ψ than the other state variables ω and δ. The values for Q, r and K_{LQR} can be set using the GUI shown in Chapter 3. In this way, every time when a new model is identified, it can be used to compute a new matrix gain K_{LQR}, that is sent to Aeolus and used online in the Pixhawk to compute the rudder command δ. Moreover, the weight matrices can be set while sailing, helping the user on the shore to better tune the *LQR* tack controller. An example of a *LQR* tack is depicted in Figure 6.6. The rudder command from the *LQR* stays for a long period close to the saturation of the rudder. This behavior allows the optimal regulator to execute a faster maneuver, without overshoot. The path of Aeolus during a *LQR* tack maneuver is depicted in Figure 6.7.

6.5 MPC Tack

The last tack controller implemented is the *MPC*: Model Predictive Control. The *MPC* is an advanced nonlinear controller, that uses the information about the system dynamics to compute an optimal control law, where "optimality" is with respect to a finite-horizon cost function. Despite of the *LQR*, the *MPC* can take into account several constraints. Other than the system dynamic, it can allow for input bounds, actuator velocity, restriction in values assumed by the state, etc. At each step, the *MPC* returns a control action for the rudder, based on the actual state of the system: $\delta = \delta(x)$. This controller uses the same main assumption of the *LQR*: to exploit the model derived in the modeling section in Chapter 3, we make the assumption that the true wind direction σ does not change during the tack maneuver. Moreover, as in the *LQR* case, the *MPC* brings the state of the system in (3.4) from the initial value $x_{\text{i}} = [\omega_{\text{i}}, \Delta\psi]^\top$ to the final value $x_{\text{f}} = [0, 0]^\top$. In the

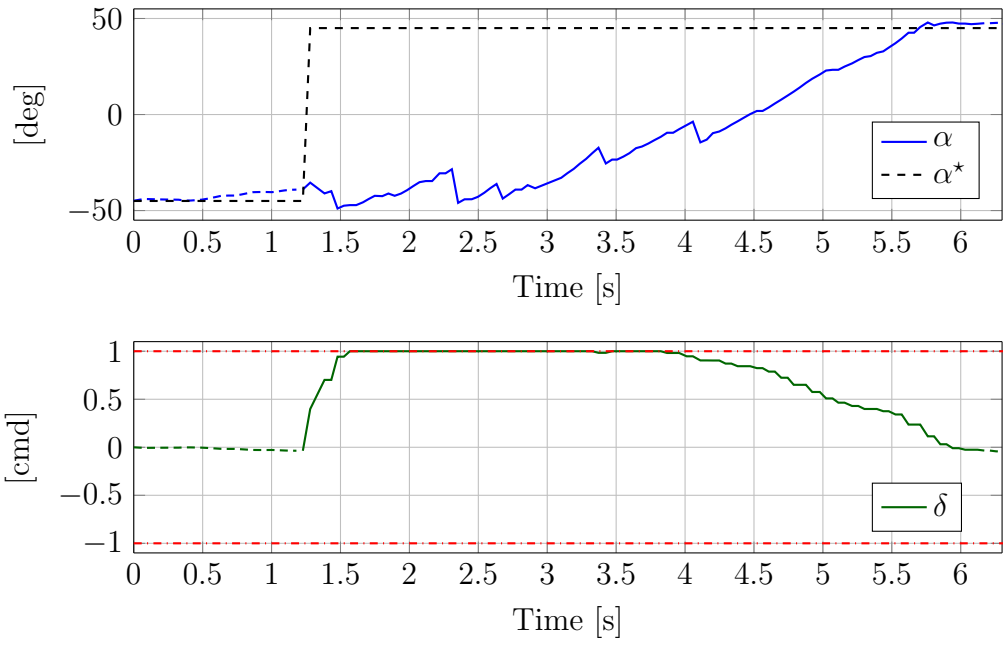


Figure 6.6: *LQR* tack from port to starboard haul. The plots show the reference $\alpha^\star$ in dashed black, the heading relative to the wind α in blue, the rudder command δ in solid green and the rudder limits in dotted red. Where the *LQR* regulator is not controlling the rudder, the α and the δ lines are dashed. The *tack-now* command has been received at time 1.22 s; from that moment, the *LQR* regulator takes care of the rudder, until the tack maneuver is not considered completed, that is around time 6.17 s. The ripples displayed in the α response (blue) are due to the action of the waves, while Aeolus is steering, and are not caused by the given rudder command.

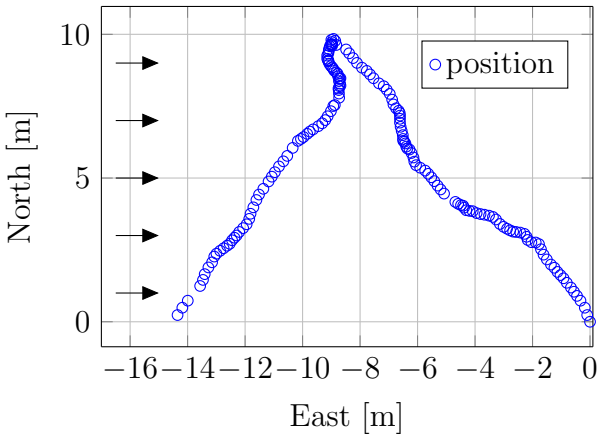


Figure 6.7: Path of AEOLUS (the GPS receiver is located above the bow) while executing an LQR tack. Starting from the origin position, AEOLUS sails upwind at port haul. A tack is executed at (−9,10), and then AEOLUS sails upwind at starboard haul. The black arrows show the mean wind direction measured by the weather station.

formulation of the *MPC* problem, we adopt the same extended state model displayed in (6.4). In this way, we can define the following MPC problem:

$$\begin{cases} \min \quad \sum_{k=0}^{N-1} \left[\hat{x}_k^\top Q \hat{x}_k + r\hat{\delta}_k^2 \right] + \hat{x}_N^\top P \hat{x}_N & & \\ \text{s.t.} \quad \hat{x}_{k+1} = \hat{A}\hat{x}_k + \hat{B}\hat{\delta}_k, & & \forall k \in 0, 1, ..., N-1 \\ |\delta_k| \leq \overline{\delta}, & & \forall k \in 0, 1, ..., N-1 \\ |\delta_k - \delta_{k-1}| \leq \overline{\Delta}, & & \forall k \in 0, 1, ..., N-1 \\ \hat{x}_0 = \hat{x}(t) & & \end{cases} \tag{6.7}$$

where the matrix $Q \succ 0$, $r > 0$ and $P \succeq 0$ are design choices and N is the length of the finite horizon. The matrix P defines the terminal cost of the cost function to be minimized. The problem takes into account both the rudder bounds and the rudder velocity, other than the system dynamics. Note that, at each step the *MPC* problem (6.7) is solved, it is not ensure the stability of the closed loop system when the optimal control computed is applied. To ensure the stability of the closed loop, we should have both a terminal cost and a terminal set. By a proper selection of these two terms, we could ensure feasibility and stability of the closed-loop system [32]. But, there are some practical issues to face about the terminal set: it reduces the region of attraction and is not easy to compute. Finally, it is often unnecessary, since starting from a stable system and using a "long enough"

horizon, the application of the optimal control, computed at each step, will produce a stable closed loop system, when starting from a neighborhood of the origin [33]. What we can easily do, is to design the terminal cost in order to mimic an infinite-horizon cost function. This is done by choosing the final cost matrix P as the solution of the discrete-time algebraic Riccati equation, using as input the system matrices $\hat{A}$ and $\hat{B}$, and the chosen weights Q and r.

The *MPC* problem (6.7) must be solved on board the PIXHAWK, while AEOLUS is sailing. To do so, we employ FORCES PRO [34], a powerful tool that allows the user to generate high performance numerical optimization solvers. Its main target is the field of optimal control and estimation, with strong support for parametric problems. Through FORCES PRO we can define the MPC problem in Matlab, as the one set in (6.7), and generate the code of the solver of the problem. This solver can be generated both for the Matlab environment and for an embedded platform. In our specific case, we generate the solver for the platform ARM CORTEX-M4; the settings used to define the *MPC* problem are displayed in the Appendix. This solver is an object file, that is linked in the firmware of the PIXHAW at the compilation time. Additionally, FORCES PRO allows us to define a parametric problem, that is, a problem where some parameters can be changed at run-time, without compiling a new version of the solver and link it in the firmware. Thus, in our case we make $\hat{A}$, $\hat{B}$, Q, r and P, present in the problem (6.7), to be parameters. In this way, the user located on the shore, can use the Matlab GUI shown in Chapter 3, identify and validate a model, tune the parameters Q and r and simulate the model response using FORCES PRO on a PC. Once the model and the parameters tuning are satisfactory, the user can send the tuned parameters back to AEOLUS via the radio link. Note that, the firmware in the PIXHAWK does not need to be compiled every time the user would like to either use a new model or change the design parameters. By following this procedure, we tested several weights and different identified model, in real tests. For example, we used the same weights of the *LQR* controller, shown in (6.6). One parameter that cannot be changed at run-time, is the length of the horizon, that is the parameter N in (6.7). In order to allow a user to try different horizon while AEOLUS is sailing, we generate different solvers through FORCES PRO, using different lengths for the horizon. Every solver is linked in the firmware, and can be selected at run-time. We implement three different horizon length: 10, 20 and 30 steps. According to the mean sample time of the model identified in Chapter 3, one step of prediction is roughly equal to 0.1 seconds. As a result of both simulation and real tests, these three horizons are "long enough" to execute a good tack maneuver.

An example from the field of a *MPC* tack, computed with a horizon of 20 steps, is depicted in Figure 6.8. The rudder command from the *MPC* stays for a long period close to the saturation of the rudder. This behavior allows the optimal regulator to execute a faster maneuver, without overshoot.

6.6 Benchmark

In this section, we analyze the performance of the tack controllers that are able to execute a tacking maneuver quickly and without overshoot. The controllers we consider here are the *implicit*, *dedicated*, *LQR* and *MPC*. We do not consider the *helmsman* controller since it is unlikely to execute a good tack. During the upwind phase of a regatta, the tack maneuver is essential to change the haul. Without the tack maneuver, the boat will be unable to take the "zig-zag" path needed for it to sail upwind. In the tack maneuver, it is important that the boat completes the turn so it does not get stuck in "irons" heading into the true wind. Simultaneously, it is undesirable for the controller to produce too much overshoot as this reduces the boat's speed over water. To accomplish these objectives, we define two different parameters, which we used to evaluate the different controllers:

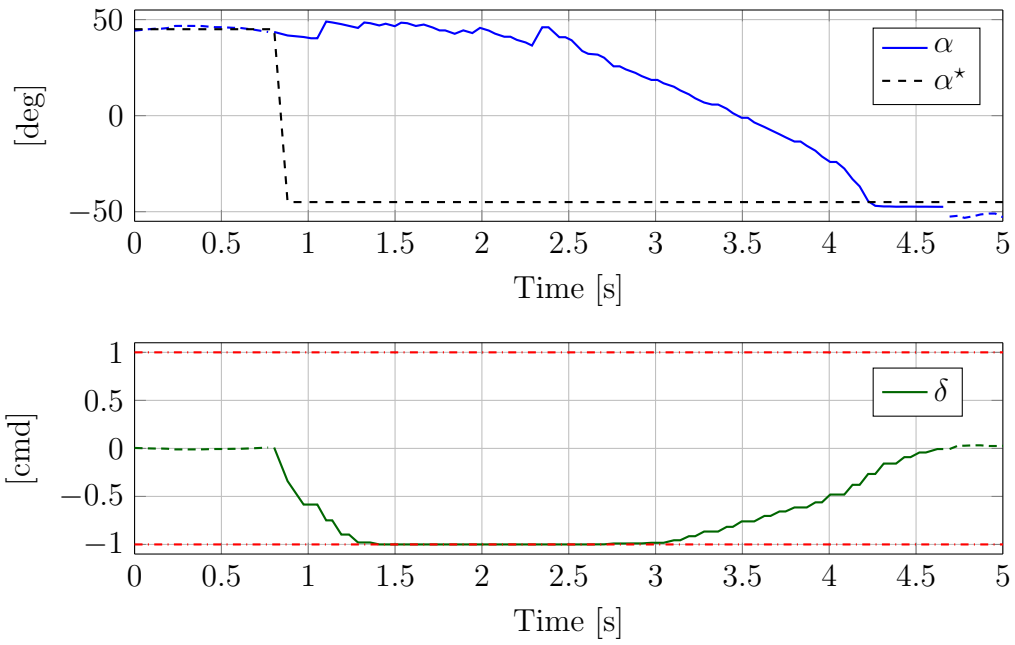
1. T: time required to reach the new reference $\alpha^\star$;
2. V: magnitude of the velocity at the end of the tack divided by the magnitude of the velocity before starting to steer.

The less the value of T and the greater the value of V, the better the tack. Note that since waves and wind change continuously, we cannot have the same environmental conditions while testing. We use the *implicit* NL tack controller as benchmark, and present T and V for the other regulators as values relative to its T and V. This comparison in shown in Table 6.1. A

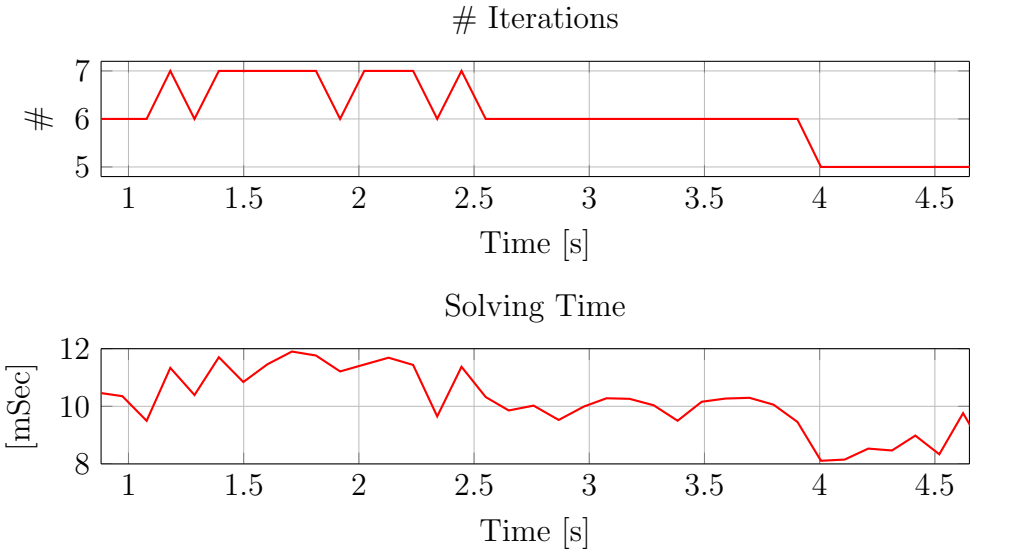
Tack type	$T_{\text{rel}}\%$	$V_{\text{rel}}\%$
Implicit NL	-	-
Dedicated NL	+43	+48
LQR	+43	+47
MPC	+45	+60

Table 6.1: Benchmark of the tack regulators with respect to the *implicit* NL controller. Average values over 22 experiments.

positive value means use of the controller provided an improvement over the *implicit* NL controller for that metric. A positive value of $T_{\text{rel}}\%$ means



(a) The plots show the reference $\alpha^\star$ in dashed black, the heading relative to the wind α in blue, the rudder command δ in solid green and the rudder limits in dotted red. Where the *MPC* regulator is not controlling the rudder, the α and the δ lines are dashed. The *tack-now* command has been received at time 0.8 s; from that moment, the *MPC* regulator takes care of the rudder, until the tack maneuver is not considered completed, that is around time 4.6 s.



(b) Number of iterations and solving time required by FORCES Pro each time the MPC problem had to be solved, when the *MPC* was controlling the rudder.

Figure 6.8: *MPC* tack from starboard to port haul. Horizon length 20 steps.

that the time required to tack is less than that required by the *implicit* NL controller. A positive value of $V_{\text{rel}}\%$ means that the loss of velocity during the tack is less than that lost by the *implicit* NL controller. Second, we instead use the *dedicated* NL tack controller as benchmark, and express the performance of the other controllers with respect to its parameters. This comparison is shown in Table 6.2.

Tack type	$T_{\text{rel}}\%$	$V_{\text{rel}}\%$
Dedicated NL	-	-
LQR	0	-1.5
MPC	$+2$	$+8$

Table 6.2: Benchmark of the tack regulators with respect to the *dedicated* NL controller. Average values over 22 experiments.

Referring to these average values, shown in Table 6.1, we can say that, the last three specialized controllers improve the performance of the steering maneuver, compared to the *implicit* NL regulator. Looking at Table 6.2, we can see how the *dedicated* NL controller and the *LQR* provide roughly the same performance. Finally, the *MPC*, by using knowledge about rudder saturation and velocity, is able to achieve better performance both in terms of time required to tack and loss of velocity due to the tacking maneuver. However, these improvements require advanced tools such as the linear state space model shown in Chapter 3 and an online solver, like the one generated by FORCES Pro. A user who cannot, or does not want to use these tools can easily implement the *dedicated* NL controller, which yields good performance and requires very little time to be implemented in an embedded platform.

Chapter 7

Conclusion

This thesis has focused on developing an autonomous sailing model boat starting from a system that could sail only if remotely operated. This autonomous sailing goal is achieved by completing four tasks:

1. develop a model for the turning dynamic;
2. collect and filter data in real time while sailing;
3. sail upwind;
4. execute tack maneuvers.

The first task has been completed with the simple, but useful, approach described in Chapter 3. This model is used to identify a linear state space model capable of describing both the yaw and yaw rate dynamic. A Matlab GUI has been created and provided to the user to identify the parameters for and validate several models. The second sub-task has been carried out by collecting and filtering the data required to sail autonomously. This is performed in real time on the onboard PIXHAWK unit and the result is supplied to the appropriate controllers. The third task has been accomplished by developing two rudder controllers, selectable at run time, and one sail regulator. Each rudder control has been tested in several trials at Lake Zurich, Switzerland. In these trials, the autonomous sailing model boat achieved its objective and sailed upwind autonomously. The fourth and last sub-goal has been achieved by implementing five different tack controllers which have been tested in trials at Lake Zurich. Four of these controllers are good candidates to execute a tack maneuver. The performance of each controller has been compared to that of the others in Chapter 6.

In order to provide a usable platform for further development on this work, an *API* (Application Programming Interface) has been provided within

the *high level controller* shown in Chapter 2. In this way, development of high level applications is independent from the low level control system. Through this *API*, future users can use the features and functionality provided by the *low level controller* without knowing its implementation.

7.1 Suggested Configuration

The best configuration begin start sailing autonomously is to employ the nonlinear controller to follow a constant heading and use the *dedicated* tack controller to steer. This configuration is the most attractive since it uses the nonlinear controller, shown in Chapter 5 both for tracking and for tacking. This controller can be easily coded in any programming language and does not require a model of the system because the controller can be tuned with a simple "*trial & error*" procedure. Finally, it requires a small amount of computation from the processor. The *dedicated* regulator was the second best tack controller in trials. However, the fastest and least decelerating tacking performance was achieved with the *MPC* controller for steering. However, before using this controler, an identification phase must be undertaken before sailing and an online optimization solver must be employed, such as the one generated through FORCES Pro.

7.2 Future Works

We think that in order to improve the autonomous sailing skill, future works should focus on three main objectives. First, a waterproof sensor, capable of detecting the sail position, could help both the sailing and the tack maneuver task. Real helmsmen do not "decouple" the rudder action from sail position while steering; however, they take advantage of the position of the sail to regulate the rudder during a tack maneuver and improve tacking performance. Therefore, by sensing the position of the sail, a controller could be developed which accounts for the fact that the sail and rudder position are not actually decoupled. Such a controller could achieve better performance than those implemented for this thesis by taking advantage of the complex sail dynamics resulting from sail position during a tack.

Second, a more advanced path-planner can be implemented. With the existing autonomous sailing skills, such a path planner can be used to perform, for example, a complete regatta task while avoiding obstacles. Other tasks are also possible with more advanced and capable path planning capability. This would enable the system to move toward being practical for

specific applications and more complex operations.

Third, looking at the roll angle of AEOLUS could be useful to regulate the sail position. While sailing, boats with displaced haul, like AEOLUS, should have a little heel as it minimizes the length of the water line and increases speed. Usually 15° to 30° is good, but it is hull specific. It could be useful, for example, to define a threshold regarding the roll angle: if the boat is too tilted, the sail should be "eased off" to bring the roll angle below the threshold again. By easing off the sail the force applied from the wind on the sheets is lowered, and so the sailboat is less heeled, that is, the roll angle is reduced. Moreover, a dynamical model of the roll and roll rate behavior could be used to develop advanced control laws which can compute "how much" the sail should be "eased off." These will help increasing the boat's speed, which depends on the roll angle, as shown in [35]. It is reasonable to think that this dynamical model can be identified in a similar fashion to that used to identify the yaw and yaw rate dynamics for the linear state space model, shown in Chapter 3.

7.3 Outcome

Two scientific articles have been written on the work developed for this thesis. The first, [36], was accepted in the *IEEE Oceans Conference*, which took place in Genoa, Italy, where it was presented in May 2015. We attended the conference and presented the work. The second article, [37], has been accepted in the *International Robotic Sailing Conference*, that will take place in Mariehamn, Finland, in September 2015.

Chapter 8

Appendix

In this chapter, are shown the main pieces of code, used to develop the whole project.

8.1 Grey and Black Type Models

Code used to identify the linear state space model, as explained in Chapter 3.

```
function [A, B, Dt] = computeGreyModel(W, Y, U, time)
% Compute black type model
% W: vector containing yaw rate sampled data
% Y: vector containing yaw angle sampled data
% U: vector containing rudder command sampled data
% time: vector containing the time when each value was sampled

%compute average delta dt over the sampled measurements
Dt = mean(diff(time)); %mean deltaT in uSec
%convert Dt from milliSec to sec
Dt = Dt / 1e3;

%number of measurements
N = length(W);

%create matrix for pseudoinverse computation
M = [   W(1 : N-1),     U(1 : N-1);
        W(1 : N-2),     U(1 : N-2)];

c = [   W(2 : N);
        (1 / Dt) * (Y(3 : N) - Y(2 : N-1))];

%compute pseudoinverse
```

```
x = M \ c;

%take result
a11 = x(1);
b1 = x(2);

A = [a11,   0;
     Dt,    1];

B = [b1;
     0];

end
```

```
function [A, B, Dt] = computeBlackModel(W, Y, U, time)
% Compute black type model
% W: vector containing yaw rate sampled data
% Y: vector containing yaw angle sampled data
% U: vector containing rudder command sampled data
% time: vector containing the time when each value was sampled

%compute average delta dt over the sampled measurements
Dt = mean(diff(time)); %mean deltaT in uSec
%convert Dt from milliSec to sec
Dt = Dt / 1e3;

%number of measurements
N = length(W);

%create matrix for pseudoinverse computation
M = [   W(1 : N-1),     Y(1 : N-1),    U(1 : N-1)];

c1 = W(2 : N);

c2 = Y(2 : N);

%compute pseudoinverse
x1 = M \ c1;
x2 = M \ c2;

%take result
A = [x1(1),     x1(2);
     x2(1),     x2(2)];

B = [   x1(3);
        x2(3)];
end
```

8.2 Mitsuta Mean

Code used to compute the an average value of the true wind direction, as shown in Chapter 4.

```
function m_mean = mitsuta_mean(meas)
% Compute Mitsuta mean
% measurements vector (meas, values in [0,2*pi])

D = meas(1);
sum = 0;

%compute Mitsuta mean
for i = 2 : length(meas)

    delta = meas(i) - D;
    if(delta < -pi)
        D = D + delta + 2 * pi;
    elseif(delta < pi)
        D = D + delta;
    else
        D = D + delta - 2 * pi;
    end

    sum = sum + D;
end

m_mean = sum / k;
```

8.3 FORCES Pro Settings

Settings used in FORCES Pro to generate the object file, used to resolve the MPC problem online the Pixhawk.

```
%% FORCES Pro Solver settings

% embedded platform
codeoptions.platform = 'ARM Cortex-M4 (FPU)';
% infinity norm of residual for inequalities
codeoptions.accuracy.ineq = 1e-4;
% infinity norm of residual for equalities
codeoptions.accuracy.eq = 1e-4;
% absolute duality gap
codeoptions.accuracy.mu = 1e-4;
```

```
% relative duality gap := (pobj-dobj)/pobj
codeoptions.accuracy.rdgap = 1e-4;
% type of variables
codeoptions.floattype = 'float';
% information printing level
codeoptions.printlevel = 0;
% initial value of mu0
codeoptions.mu0 = 1;
```

8.4 Setting Up a Real Sailing Test

In this section we briefly show how to setup AEOLUS, in order to execute a real sailing test.

8.4.1 Firmware Version

The latest firmware of the whole project, updated until the end of March 2015, can be found at [38]. It must be flashed into the PIXHAWK board, following the instructions that can be found at [25]. Before flashing it, an user should check if the "*outdoor*" version is set. To do so, open the following files and follow the comments inline:

1. `src/modules/path_planning/pp_config.h`
2. `src/modules/autonomous_sailing/settings.h`
3. `src/modules/parser_200WX/settings.h`

8.4.2 Acquire a GPS Position

Once AEOLUS has been positioned in an open space, under clear sky, it can be switched on. Now, the user can open the application QGROUNDCONTROL [26], and establish the radion link communication to the PIXHAWK. The default communication bad rate should be 57 600. The user can now look at the *Mission* pannel, in QGROUNDCONTROL, to check when the PIXHAWK has acquired a good enough GPS position.

8.4.3 Arm the System

Once AEOLUS has acquired a good enough GPS position, it can be "armed". To do so, first of all switch on the remote control, and make sure it is set into the "manual" model. Second, press the safety switch for roughly two

seconds, until it starts blinking with a "high" frequency. Now bring the left stick into the lower right position, for roughly two seconds. At this step, you should be able to control both the rudder and the sails with the remote control.

8.4.4 Autonomous Sailing

Once AEOLUS is sailing into the water, the user can use the remote control and switch from "manual" to "autonomous" mode. In this way the sails and rudder commands computed into the firmware are effectively applied to the actuators. The following list, explains the main parameters, and their default values, that can be used to set the behavior of AEOLUS, when it is sailing autonomously. They can be set into the *Onboard Paramters* panel, under the *Plots* section of QGROUNDCONTROL. In Table 8.1, there are shown the most important parameters, that can be set by an user. Other parameters, such as the ones about the *LQR* and the *MPC*, can be set using the Matlab GUI shown in Chapter 3.

Name	Description	Default Value
AS_SAIL	Use -1 to let autonomous controller use its computed value for the sails. If you want to force the sails to be in a certain position set this parameter to a positive value within $[0, 0.56]$, where 0 mean sails fully opened, 0.56 means sails fully closed	-1
AS_MAX_RUD	Set a new value for the maximum rudder command. Must be in $[0, 1]$	1
AS_SAI_CL_CMD	Sails command value when sails should be considered fully closed	0.56
AS_SAI_X1_AL	Positive α angle, in degrees, at which sails should start opening. Look at Figure 5.3	45
AS_SAI_X2_AL	Positive α angle, in degrees, at which sails should be fully opened. Look at Figure 5.3	150
AS_TY_TCK	Type of tack maneuver: *implicit* (0), *LQR* (1), *MPC* (2) and *dedicated* (3)	3
AS_TCK_USE_Y	Use both α_χ and α_ψ (0) in equation (4.4), or only α_ψ (1), to compute α, while tacking	1
AS_RUD_P	k_p gain, valid for both equation (5.1) and (5.2)	0.35
AS_RUD_CP	c_p value in equation (5.2)	0.35
AS_RUD_TYPE	Type of rudder controller in upwind phase: P (0) or NL (1)	1

AS_COG_DELAY_S	T value in equation (4.5)	1.5
AS_WIN_AL	Number of samples used in the moving average filter for the α angle	10
AS_WIN_TWS	Number of samples used in the moving average filter for the true wind speed	10
AS_WIN_TWD	Number of samples used in the moving average filter for the σ angle	10
AS_USE_FIXED_TWD	Compute α in (4.1) using σ provided by the moving average filter (0), or using a fixed σ (1), see ASP_MEAN_WIND_D	0
AS_TCK_P_K	k_{p} gain of the *dedicated* NL controller	0.7366
AS_TCK_P_C	c_{p} value of the *dedicated* NL controller	0.1
ASO_STP_TCK_S	T used in (6.1), in seconds	0.8
ASP_ALST_ANG_D	Reference $\alpha^{\star}$, in degrees, works only if ASP_ALST_SET is 1	45
ASP_ALST_SET	use ASP_ALST_ANG_D to change $\alpha^{\star}$ (1)	1
ASP_DO_MANEUV	Send a "tack-now" command to the *low level controller*, see Chapter 2	0

Table 8.1: Settable autonomous sailing parameters.

8.5 Further Results From The Field

Here we show further tack tests, executed at Lake Zurich, Switzerland.

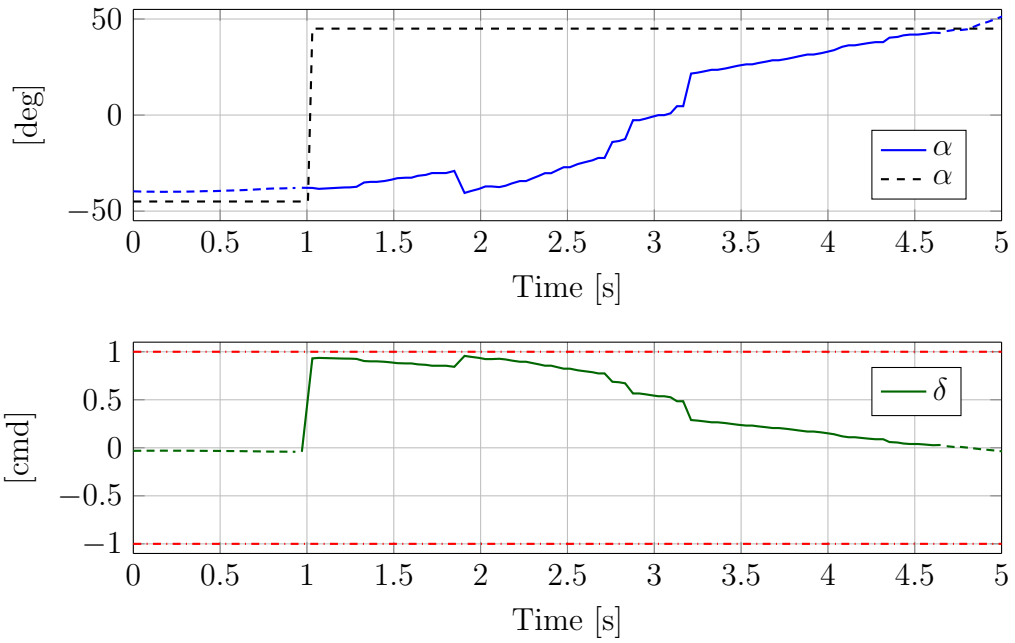


Figure 8.1: *Dedicated* tack from port to starboard haul. The plots show the reference $\alpha^\star$ in dashed black, the heading relative to the wind α in blue, the rudder command δ in solid green and the rudder limits in dotted red. Where the *dedicated* regulator is not controlling the rudder, the α and the δ lines are dashed. The *tack-now* command has been received at time 1 s; from that moment, the *dedicated* regulator takes care of the rudder, until the tack maneuver is not considered completed, that is around time 4.6 s.

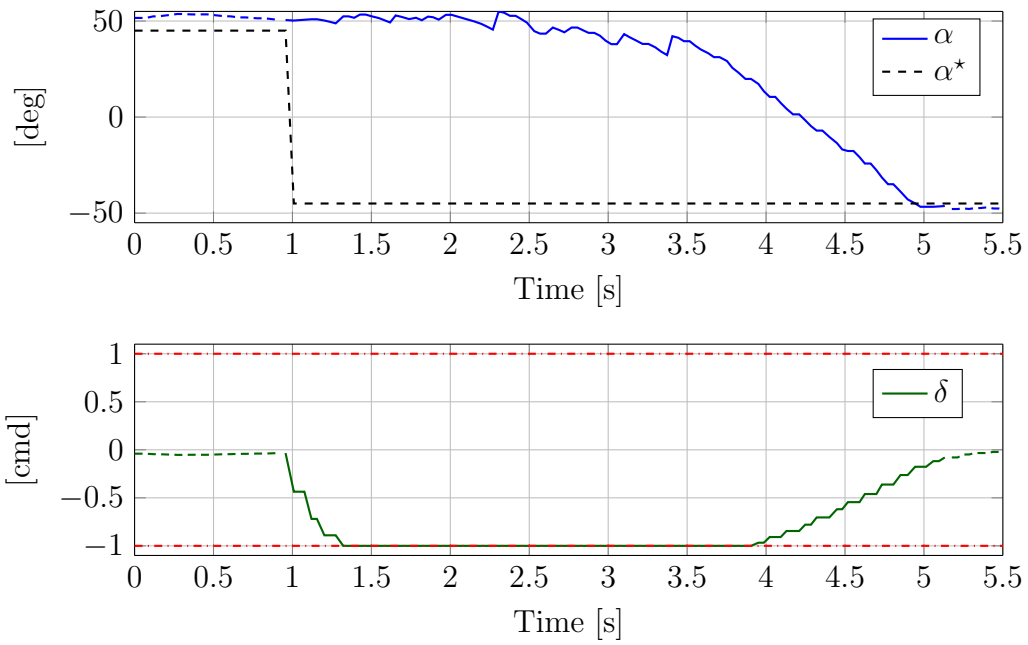
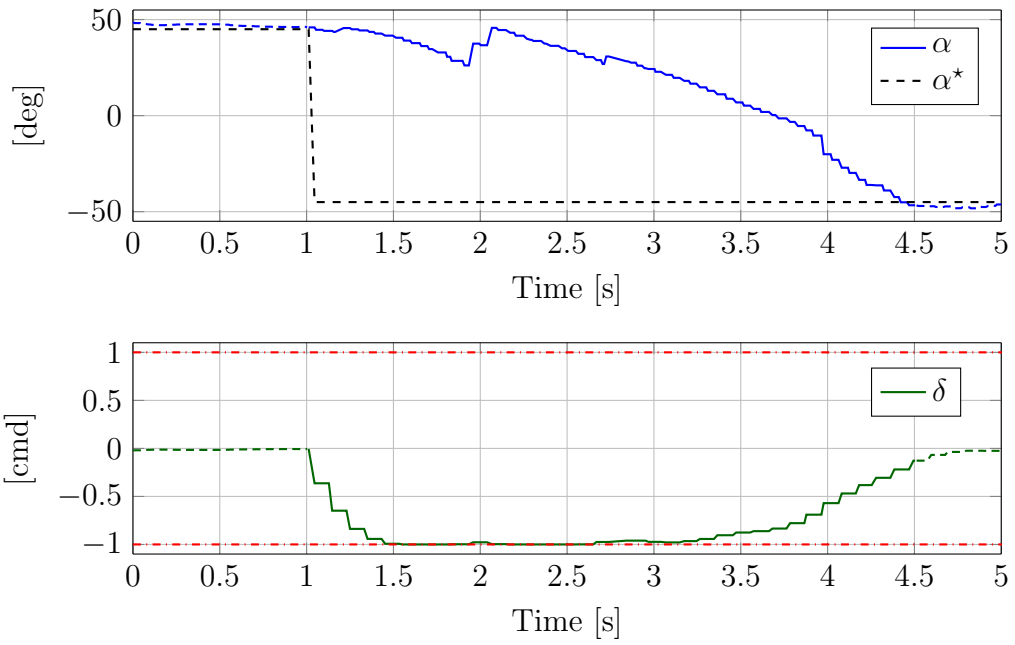
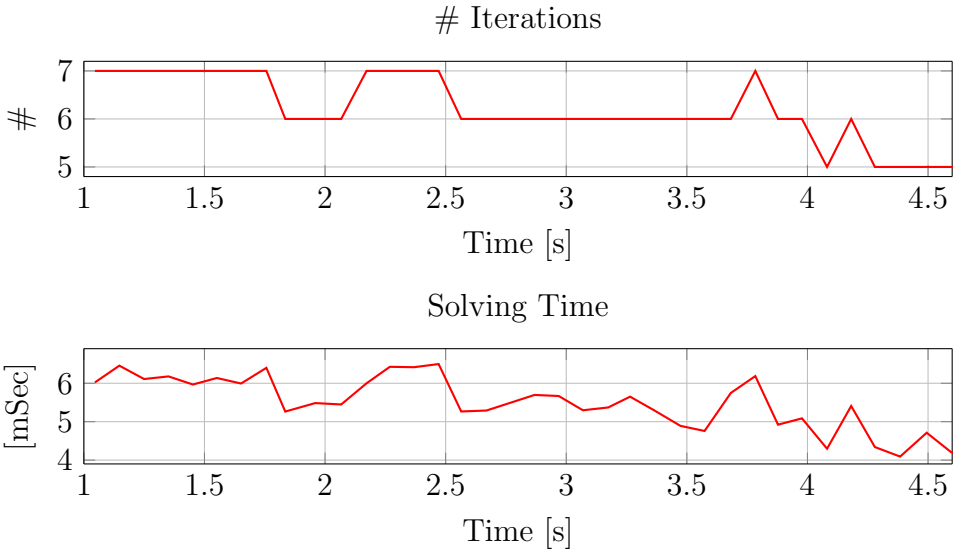


Figure 8.2: *LQR* tack starboard to port haul. The plots show the reference $\alpha^\star$ in dashed black, the heading relative to the wind α in blue, the rudder command δ in solid green and the rudder limits in dotted red. Where the *LQR* regulator is not controlling the rudder, the α and the δ lines are dashed. The *tack-now* command has been received at time 1 s; from that moment, the *LQR* regulator takes care of the rudder, until the tack maneuver is not considered completed, that is around time 5.1 s.



(a) The plots show the reference $\alpha^\star$ in dashed black, the heading relative to the wind α in blue, the rudder command δ in solid green and the rudder limits in dotted red. Where the *MPC* regulator is not controlling the rudder, the α and the δ lines are dashed. The *tack-now* command has been received at time 1 s; from that moment, the *MPC* regulator takes care of the rudder, until the tack maneuver is not considered completed, that is around time 4.5 s.



(b) Number of iterations and solving time required by FORCES Pro each time the MPC problem had to be solved, when the *MPC* was controlling the rudder.

Figure 8.3: *MPC* tack from starboard to port haul.Horizon length 10 steps.

Bibliography

[1] J. Kalwa, M. Carreiro-Silva, F. Tempera, J. Fontes, R. Santos, M.-C. Fabri, L. Brignone, P. Ridao, A. Birk, T. Glotzbach, *et al.*, "The morph concept and its application in marine research," in *OCEANS-Bergen, 2013 MTS/IEEE*, pp. 1–8, IEEE, 2013.

[2] R. Martins, J. de Sousa, C. Afonso, and M. Incze, "Rep10 auv: Shallow water operations with heterogeneous autonomous vehicles," in *OCEANS, 2011 IEEE - Spain*, pp. 1–6, June 2011.

[3] C. C. Eriksen, T. J. Osse, R. D. Light, T. Wen, T. W. Lehman, P. L. Sabin, J. W. Ballard, and A. M. Chiodi, "Seaglider: A long-range autonomous underwater vehicle for oceanographic research," *Oceanic Engineering, IEEE Journal of*, vol. 26, no. 4, pp. 424–436, 2001.

[4] Kongsberg seaglider, "Web: `http://www.km.kongsberg.com/ks/web/nokbg0240.nsf/AllWeb/EC2FF8B58CA491A4C1257B870048C78C?OpenDocument`," 2015.

[5] A. Alvarez, A. Caffaz, A. Caiti, G. Casalino, L. Gualdesi, A. Turetta, and R. Viviani, "Folaga: a low-cost autonomous underwater vehicle combining glider and auv capabilities," *Ocean Engineering*, vol. 36, no. 1, pp. 24–38, 2009.

[6] N. Cruz and J. Alves, "Autonomous sailboats: An emerging technology for ocean sampling and surveillance," in *OCEANS 2008*, pp. 1–6, Sept 2008.

[7] O. Mnage, A. Bethencourt, P. Rousseaux, and S. Prigent, "Vaimos: Realization of an autonomous robotic sailboat," in *Robotic Sailing 2013* (F. L. Bars and L. Jaulin, eds.), pp. 25–36, Springer International Publishing, 2014.

[8] UBC SailBot, "Web: `http://ubcsailbot.org`. University of British Columbia," 2015.

[9] TRST, "Web: `http://sites.tufts.edu/roboticboat`. Tufts University," 2015.

[10] Olin Robotic Sailing Team, "Web: `http://olinroboticsailing.com`. Franklin W. Olin College of Engineering," 2015.

[11] FASt, "Web: `www.fe.up.pt/fast`. Faculty of Engineering, University of Porto," 2015.

[12] World Robotic Sailing Championship, "Web: `http://www.roboticsailing.org/`," 2015.

[13] L. Jaulin and F. L. Bars, "A simple controller for line following of sailboats," in *5th International Robotic Sailing Conference*, (Cardiff, Wales, England), pp. 107–119, Springer, 2012.

[14] E. C. Yeh and J.-C. Bin, "Fuzzy control for self-steering of a sailboat," in *Proc. of the Int. Conf. on Intelligent Control and Instrumentation*, vol. 2, (Singapore), pp. 1339–1344, 1992.

[15] J. Abril, J. Salom, and O. Calvo, "Fuzzy control of a sailboat," *International Journal of Approximate Reasoning*, vol. 16, no. 3, pp. 359–375, 1997.

[16] R. Stelzer, T. Proll, and R. I. John, "Fuzzy logic control system for autonomous sailboats," in *Proc. of the IEEE Int. Fuzzy Systems Conference*, pp. 1–6, 2007.

[17] N. A. Cruz and J. C. Alves, "Auto-heading controller for an autonomous sailboat," in *Proc. of the IEEE Oceans*, (Sydney, Australia), pp. 1–6, 2010.

[18] T. Emami and R. J. Hartnett, "Discrete time robust stability design of PID controllers autonomous sailing vessel application," in *Proc. of the IEEE American Control Conference*, pp. 1993–1998, 2014.

[19] L. Xiao and J. Jouffroy, "Modeling and nonlinear heading control for sailing yachts," *IEEE Journal of Oceanic Engineering*, vol. 39, no. 2, pp. 256–268, 2014.

[20] L. Xiao, T. I. Fossen, and J. Jouffroy, "Nonlinear robust heading control for sailing yachts," in *IFAC Conf. in Maneuvering and Control of Marine Craft*, (Arenzano, Italy), pp. 404–409, 2012.

[21] F. Plumet, C. Petres, M.-A. Romero-Ramirez, B. Gas, and S.-H. Ieng, "Toward an autonomous sailing boat," *IEEE Journal of Oceanic Engineering* (in press), 2014.

[22] Aeolus Project, "Web: `http://control.ee.ethz.ch/~autsail/`," 2015.

[23] J. Eisenberg, "Hardware setup for an autonomous sailing boat," 2014. Semester project, Automatic Control Laboratory, ETH Zurich.

[24] Airmar, "Web: `www.airmartechnology.com`," 2015.

[25] Pixhawk, "Web: `http://pixhawk.org/start`," 2015.

[26] QGroundControl, "Web: `http://http://qgroundcontrol.org`," 2015.

[27] D. H. Titterton and J. Weston, *Strapdown Inertial Navigation Technology*, ch. 13.7, pp. 409–417. United Kingdom: IET, 2004.

[28] T. I. Fossen, *Guidance and Control of Ocean Vehicles*, ch. 2, pp. 5–55. John Wiley & Sons Ltd, 1994.

[29] Y. Mori, "Evaluation of several single-pass estimators of the mean and the standard deviation of wind direction," *Journal of climate and applied meteorology*, vol. 25, no. 10, pp. 1387–1397, 1986.

[30] C. Scali, G. Nardi, A. Landi, and A. Balestrino, "Variable structure PI controllers in the presence of uncertainty and saturation," in *IFAC Symposia Series-Proceedings of the 12th Triennal World Congress*, vol. 5, (Oxford, UK), pp. 637–642, Goodwin and Evans ed., Pergamon Press Inc., 1994.

[31] E. Jury, "A simplified stability criterion for linear discrete systems," *Proceedings of the IRE*, vol. 50, no. 6, pp. 1493–1500, 1962.

[32] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[33] M. Morari, F. Borrelli, and A. Bemporad, *MPC Book*, pp. 249–281. 2014. Web: `http://www.mpc.berkeley.edu/mpc-course-material`.

[34] A. Domahidi and J. Jerez, "FORCES Professional." embotech GmbH (`http://embotech.com/FORCES-Pro`), July 2014.

[35] L. Hertel and A. Schlaefer, "Data mining for optimal sail and rudder control of small robotic sailboats," in *Robotic Sailing 2012*, pp. 37–48, Springer, 2013.

[36] M. Tranzatto, S. Grammatico, A. Liniger, and A. Landi, "The debut of Aeolus, the autonomous model sailboat of ETH Zurich," in *IEEE Oceans Conference*, May 2015.

[37] J. Wirz, M. Tranzatto, A. Liniger, M. Colombino, H. Hesse, and S. Grammatico, "Aeolus, the ETH Autonomous Model Sailboat," in *International Robotic Sailing Conference*, Sept. 2015.

[38] Pixhawk firmware, "Web: `https://github.com/aliniger/Firmware/tree/as_gridlines_pathplanning`," 2015.