UPPSALA
UNIVERSITET

UPTEC F 23012
Examensarbete 30 hp
April 2023

Online Minimum Jerk Velocity Trajectory Generation

for Underwater Drones

Jakob Andrén

Civilingenjörsprogrammet i teknisk fysik

UPPSALA
UNIVERSITET

Online Minimum Jerk Velocity Trajectory Generation

Jakob Andrén

Abstract

This thesis studies real-time reference ramping of human input for remotely operated vehicles and its effect on system control, power usage, and user experience. The implementation, testing, and evaluation were done on the remotely operated Blueye Pioneer underwater drone.

The developed method uses minimum jerk trajectories for transitioning between varying target velocities with a constant end jerk target. It has a low computational cost and runs in real-time on the Blueye Pioneer underwater drone. The presented method produces a well-defined reference with continuous position, velocity, and acceleration states that can be used in the feedback loop.

Experiments and simulations show that the method produces a smoother and more predictable motion path for the user. The motions are better suited for video recordings and remote navigation, compared to the direct usage of human input velocity. The smoother reference reduces the controller tracking error, the peak control input, and the energy usage. The introduced acceleration reference state is used for feedforward control on the system. It improves the feeling of controlling the drone by reducing the system lag, the position tracking error, and the rise time for velocity changes.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Uppsala

Handledare: Johannes Schrimpf Ämnesgranskare: Hans Rosth

Examinator: Tomas Nyberg

Populärvetenskaplig sammanfattning

Antalet ting som vi människor styr och interagerar med har genom historien blivit fler och fler. Båtar, tåg, cyklar och bilar tillhör vardagen och har nu använts länge. Vi har flygplan som åker snabbare än ljudet och en mängd andra farkoster. Gemensamt för alla dessa fall är att vi har åkt med som en förare eller pilot och därmed upplevt de krafter, ljud och dynamik som fordonet upplever och utsätts för. De hjälper oss avsevärt med att förstå och känna vad som är möjligt för tekniken då vi har en direkt återkoppling av vad som sker vilket i sin tur hjälper oss i vår inlärning.

Det senaste deceniet har sett en explosion i användningen av obemanade farkoster där en människa fortfarande är ansvarig för att styra men från en annan plats. Piloten kan ge samma kommandon men använder sig av live video och sensordata för att se och känna vad som sker. Detta är dock inte helt lätt för oss människor då videoöverföringen introducerar lite fördröjning i vad man ser och känslan av acceleration och vibrationer är oerhört svåra att förmedla. Det innebär att man styr farkosten med reducerad känsla av kontakt. En liknelse är att dansa en pardans utan händer eller annan form och kontakt och basera all kommunikation på visuella intryck.

Det går att som distanspilot kompensera för avsaknaden av andra intryck än visuella via video genom träning, tålamod och helst möjligheten att se farkosten på nära håll. Att just se farkosten är inte alltid möjligt och det är extra svårt i sammanhanget med undervattensdrönare som det här arbetet är fokuserat på. Drönaren Blueye Pioneer har dessutom som mål att vara enkel att kontrollera från första dyket, i princip utan någon träning. Ett av de största problemen som identifierades var att nybörjare ofta gjorde alltför extrema rörelser, vilket orsakade kraftiga accelerationer och skakig video. För att sätta det i perspektiv så kommer en erfaren dykare att berätta för dig att långsamma och lugna rörelser är nycklen till framgång vid dykning. Slutprodukten av undervattensdrönaren är dessutom video och bra video är till stor del beroende av att kameran rör sig mjukt och kontrollerat.

Tanken med det här arbetet är att låta farkosten ta ett större ansvar för vilka rörelser som är passande och möjliga att genomföra och därmed lugna ner rörelserna. Inlärningsbehovet skulle drämed kunna reduceras genom att ta bort en av de saker som annars hade varit svårt för en nybörjare att ta in precis i början. Även erfarna användares körning förbättrades med exaktare och jämnare rörelserna som resultat.

Acknowledgements

Thank you Blueye for giving me a change at first developing this at first crazy idea and then allowing me to develop as a human and engineer at the company during the process of taking the amazing Blueye Pioneer to market. Thank you Johannes for introducing me to the team and supporting me as suppervisor even during my much belated and challenging finalization of the report. To my parents for nagging about finishing the written part as well not just the product. Thank you Maja for being there all the time along the way.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Setting

The work for this thesis has been conducted at Blueye Robotics AS in Trondheim as part of their underwater drone development. My focus, apart from gathering the data for my research, was always to deliver deployable work. Meaning tested and robust solutions ready for deployment on hundreds of units in the field. The work was done during an early phase of the company's development and required solving several other tasks apart from the thesis work. These are not included in the report as they were more in the nature of bug hunting and general development. The presented material is what I consider to be the core contribution that I hope might be interesting to future developers at Blueye as well as other control system developers around the globe.

1.2 Project Purpose and Goal

This thesis started with the idea of developing a control system for underwater drones optimal for cinematic video capture. The crucial thing here is providing smooth motion, which can be broken down into two parts:

- Static and stable if no motion is demanded.
- Any demanded motion starts and stops gradually.

The first part means that the control system should do whatever it can not to oscillate or drift away from the drone's current state. This is what a regular control

system is already good at achieving, and no new contributions are expected in this area. Starting and stopping gradually and smoothly is, on the other hand, something that is not yet an integrated part of most control methods. This is what this thesis is targeting: a better method for smooth responses to any human input than found in the available literature.

1.3 Scope

All work is targeted for the Blueye Pioneer underwater drone as a hardware platform, and the controllable dimensions of the drone are its depth and heading. Any other degrees of freedom are either free or inherently stable. The command input is real-time user input from a 4 axis gamepad which is interpreted as forces for horizontal movements and velocity for the heading and depth.

The control system has access to an onboard pressure sensor for depth readings and an Inertial Measurement Unit for attitude and heading. No external sensors for velocity and position were available for the project. This means that the project does not cover position hold or attitude stabilization topics. The final result has to be computationally lightweight enough to run on the drone's microprocessor hardware with room for extending it to all six degrees of freedom.

1.4 Structure of the Thesis

The thesis begins by introducing the Blueye Pioneer hardware platform, its customers, and its expected usage areas. Subsequently, the thesis presents the available sensors and briefly describes the drone's dynamics and motion constraints. Chapter 3 focuses on the theory and concepts of smoothness and piecewise functions. It is presented from the perspective of mathematics as well as human motion and perception. Theory on control setups, observers, and reference generation is described in the next chapter. Related work for underwater robotics is also presented here.

Chapter 5 concentrates on the actual implementation details of the system for the thesis, including deriving the Online Minimum Jerk Velocity Reference and acceleration feedforward tuning. The results of using the presented reference generation method are then shown in both simulations and experiments. The thesis finally ends with a conclusion and proposals for future work.

Chapter 2

Blueye Pioneer

The Blueye Pioneer, Figure 2.1a, is an underwater drone developed by the Norwegian company Blueye Robotics AS in Trondheim. It is developed for easy inspection and exploration of assets, objects, structures, and marine life. The drone is sold to a large variety of customers, including aquacultures, water management, harbors, universities, and diving companies. One of its core principles is user-friendliness. This is achieved by not requiring complicated setup work and securing intuitive control. The starting point of this master thesis was to improve the control of the drone for higher quality video and easier piloting. All development has been done with the Blueye Pioneer platform in mind, and the resulting methods are integrated and released to the customers.

2.1 Movement

The Blueye Pioneer is equipped with four 350 W brushless outrunner electric motors, Figure 2.1b. These motors are placed and vectored to achieve control over four freedom dimensions. One is placed vertically in a central tube close to the center of mass for vertical movement. One is placed sideways through the body for sideways motion, and two are placed on the arms for forward and backward motion. The drone's heading can also be controlled by differential thrust on the thrusters placed on the arms. The two remaining dimensions of freedom, pitch, and roll of the drone are non-controllable but instead passively stable by design. This is achieved by concentrating heavy parts of the drone towards the bottom and adding buoyancy material on the top. This creates an offset between the center of

(a) The Blueye Pioneer.

(b) Thrusters.

(c) Pendulum principle.

(d) Hydrodynamic shape.

Figure 2.1: Some core design elements of the Blueye Pioneer.

mass and the center of buoyancy of the drone. The offset can be seen as a simple pendulum that always returns the drone upright.

The heaviest single part of the drone is the 96 Wh battery placed at the bottom of the drone and delivers about 2 hours of dive time. The reason for using an onboard battery instead of delivering power over the tether is to keep the tether as thin as possible. A thinner tether means lower drag in the water column which can make a significant difference, especially when there are significant water currents. The drone is also hydrodynamically designed for low drag when moving forward and vertically. The smooth shell of the drone adds another advantage. It is less likely for the Pioneer to get entangled in underwater objects.

2.2 Video and Communication

The primary purpose of the Blueye Pioneer underwater drone is to deliver an ”underwater eye” by providing high-quality, low latency live video for the user. The HD camera is optimized for streaming and is assisted by the onboard 3000 lumens of artificial light for deep dives where almost all-natural light is gone.

The video is compressed on the drone and transmitted to the surface using a wired connection. The cable between the surface unit and the drone is also known as a tether or umbilical. The tether connects to a surface unit which provides a Wi-Fi access point. A smart device running the Blueye App can then connect to the video stream from the drone with low latency. The same connection flow delivers telemetry data from the drone, such as depth, heading, battery status, and water temperature.

Human input commands such as motion requests, mode change, start and stop recording, and light changes are also sent over this communication pipeline. The pilot has two possible input interfaces, touch control on the smart device or a dedicated Bluetooth controller.

The drone is equipped with a tether because wireless communication cannot be done with radio waves through water. Electromagnetic radiation is quickly absorbed by water and thus impossible to use for high-speed data transmission. One option is to use sound as a carrier wave instead, but acoustic communication methods are too slow for live transmission of video. The transmission speed is counted in bits per second and would only be able to send low-resolution pictures occasionally. This can be good enough for autonomous vehicles but not when high-speed communication is needed.

Figure 2.2: Data flow for the Blueye system.

2.3 Software Platform

The Blueye Pioneer runs an entire Linux computer, an iMX6 System-on-Module with a Quad core ARM Cortex A9 processor and 4 GB of RAM. This provides enough performance for running everything from a video pipeline to the control system on the drone. The foundation of the software stack is the inhouse Linux distribution Blunux, which is based on the Yocto Project [26]. The control system is, based on the Robotics Operating System, ROS [17], which is not an operating system but an inter-process communication tool. It makes communication between processes on the same computer or several computers on the same local network easy to set up. Each process in ROS is called a node that can interact with other nodes on the same network using different communication methods. The two most common ones are topics and services. Any node can publish messages to a topic, and any node can subscribe to a topic. Topics thus provide the mechanism for one to many, many to one or many to many communication. A node can provide a service that a different node can request. The service can also provide an answer to the request and is therefore useful for triggering events.

This system proved to be advantageous for the development of this thesis as it is possible to subscribe and view data from a computer in real-time as well as in triggering services. Another helpful development setup was the automatic build server and the possibility to cross compile individual nodes locally and then install those binaries on a development drone for testing on actual hardware. ROS nodes can be written in both Python and C++. Most production code for the drone was written in C++, but Python was used extensively during the development process.

2.4 Pressure Sensor

The Blueye Pioneer drone is equipped with a pressure sensor that is capable of up to 42 Hz sampling rate, with five mBar precision and 30 Bar maximum pressure. The measured pressure, p, is a linear function of depth h and the density of water ρ.

$$p = \rho g h \tag{2.1}$$

Note that the measured pressure is the absolute measured pressure from both atmospheric and water pressure. It is thus necessary to remove the current atmospheric pressure from the measurement to get the correct depth. This was solved by

measuring the pressure at every start-up of the drone and applying that observation as the atmospheric reference.

$$h = \frac{p_{\mathtt{abs}} - p_{\mathtt{atm}}}{\rho g} \tag{2.2}$$

The water density, ρ, is dependent on the salinity and temperature of the water that the dive is performed in and must be manually defined depending on where you are. Freshwater has a density of 999 $\mathrm{kg\,m^{-3}}$ at a temperature of 10 $^{\circ}\mathrm{C}$ and the average seawater density is 1025 $\mathrm{kg\,m^{-3}}$. Note that the pressure in these calculations is in the SI unit pascal, Pa. The sensor is thus able to measure pressures down to about 300 meters with a precision of 5 cm.

2.5 Inertial Measurement Unit

An Inertial Measurement Unit, IMU, is a sensor made for measuring motion based on the kinetic forces of motion. It typically has an accelerometer and a gyroscope. The accelerometer measures the sum of gravity and the linear acceleration experienced by the sensor. The gyroscope measures the angular velocity of the sensor. Both usually consist of an orthogonal triple of accelerometers and gyroscopes, making up a total of six sensors in one package. The IMU used on the Blueye Pioneer also includes a 3 axis magnetic flux sensor which can be used as a compass. Especially the gyroscope is sensitive to drift and calibration errors, so extra care was taken to implement a robust calibration routine of always calibrating the sensor at every startup.

Chapter 3

Smoothness

As the purpose of this thesis is to improve the quality of video recordings, it is thus essential to take a closer look at the definition and concept of smoothness. The chapter introduces the fundamentals of smoothness in mathematics and examples of motion related to humans and how we experience them.

The foundation of a successful inspection video recording is stability, which can be achieved by using a tripod or similar construction to anchor the camera, thus reducing vibrations and shaking. Underwater objects are commonly too large to be seen all at once due to short visibility, and the operator will need to move around to see them. There is also a need to make minor corrections due to underwater currents pushing the drone away from objects of interest. The problem formulation then becomes how to change the framing of a shot during a video recording. Or in other words, how is the transition between these two frames performed? It turns out that the key to making these transitions usable and look good in the video is to make them smooth.

This chapter introduces the fundamentals of smoothness in mathematics and examples of motion related to humans and how we experience them.

3.1 Mathematical Definition

A function is considered smooth if its derivatives exist and are continuous up to some order n [29]. It is denoted as C^n smooth and is a useful initial metric to evaluate smoothness. Functions like sinus are infinitely differentiable and are therefore considered C^∞ smooth. Polynomials are also C^∞ smooth as they will reach the

derivative 0, which is differentiable forever. It is helpful to introduce naming for first derivatives for functions of time for further discussion. Let the position, p, at time t be defined by $x(t)$, the names and common notations [27] for the six first derivatives are as follows.

Position	p	x	x
Velocity	v	$\dot{x}$	$\frac{dx}{dt}$
Acceleration	a	$\ddot{x}$	$\frac{d^2x}{dt^2}$
Jerk	j	$\dddot{x}$	$\frac{d^3x}{dt^3}$
Snap	s	$\overset{4}{\dot{x}}$	$\frac{d^4x}{dt^4}$
Crackle		$\overset{5}{\dot{x}}$	$\frac{d^5x}{dt^4}$
Pop		$\overset{6}{\dot{x}}$	$\frac{d^6x}{dt^6}$

A C^5 smooth function thus has a continuous crackle but a discontinuous pop, to name an example. A C^0 function has a discontinues velocity like $x(t) = |t|$ which is not differentiable at $t = 0$ because the velocity steps from -1 to 1.

$$x(t) = \begin{cases} 2t - t^2, & \text{if } t \geq 0 \\ 2t + t^2, & \text{otherwise} \end{cases} \tag{3.1}$$

Is an example of a C^1 smooth function because the velocity, $\dot{x}$, is continuous but the acceleration, $\ddot{x}$, is discontinuous at $t = 0$ and changes from -2 to 2.

3.1.1 Piecewise Functions

A piecewise function is a function which is defined by different functions for different segments of the variable its defined over. Equation 3.1 is an example of a piecewise function defined by two parts separated at $t = 0$. A piecewise function can consists of an arbitrary number of pieces joint together and there is no restriction on continuity. A piecewise function is continuous if all pieces are continuous and if all connections between pieces are continuous, meaning $x_1(t_1) = x_2(t_1)$ if they connect at $t = t_1$. This extends to higher derivatives such that the function is C^1 smooth if $\dot{x}_1$ and $\dot{x}_2$ also are continuous and $\dot{x}_1(t_1) = \dot{x}_2(t_1)$.

3.1.2 Minimum Jerk Trajectories

The smoothness of a function or piecewise function determines the difference between a function that is C^1 and C^2 smooth. However, it does not help determine

which function is better if they both, for example, are C^2 smooth. One powerful tool is the use of a cost function for some metrics. This also opens the door for applying optimization methods for finding the optimal function, from now on, known as trajectory, between two states. One useable trajectory cost function is defined in equation 3.2, which takes the square sum of jerk during the transition. This means that one trajectory can be determined to be smoother than another by comparing their total cost, i.e., the total amount of jerk during the transition.

$$C = \frac{1}{2}\int_{t_1}^{t_2} \dddot{x}^2 \, dt \tag{3.2}$$

It turns out that jerk is very problematic in robotics [18][24], industrial applications [6][1], and human-centric applications [16]. High jerk causes vibrations, instabilities, and unnecessary damage to equipment and lowers the equipment's maximum possible performance. High jerk is uncomfortable and sometimes even dangerous to humans. There are thus good reasons to minimize the amount of jerk for the trajectory given by a defined state transition.

This minimization problem has an analytical solution if the time t_1 and t_2 are known and there are no constraints on the states. The function that solves this minimization problem is the Euler-Lagrange equation.

$$x^*(t) = \underset{x(t)}{\operatorname{argmin}} \int_{t_1}^{t_2} \mathcal{L}(\dddot{x}, \ddot{x}, \dot{x}, x, t) \, dt = \underset{x(t)}{\operatorname{argmin}} \frac{1}{2}\int_{t_1}^{t_2} \dddot{x}^2 \, dt \tag{3.3}$$

$$\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) + \frac{d^2}{dt^2}\left(\frac{\partial \mathcal{L}}{\partial \ddot{x}}\right) - \frac{d^3}{dt^3}\left(\frac{\partial \mathcal{L}}{\partial \dddot{x}}\right) = 0 \tag{3.4}$$

The cost function only contains $\dddot{x}$, so all partial derivatives except the last one will be zero, leaving us with the condition that the sixth derivative of time, pop, should equal zero.

$$\frac{d^6 x}{dt^6} = 0 \tag{3.5}$$

The trajectory is thus defined by the fifth-order polynomial that satisfies our state transition.

$$x(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5 \tag{3.6}$$

A quintic function has six parameters, and we thus need to define six conditions for the trajectory at given times. The most usual and most straightforward approach is to define the state at the initial and final times and to solve it using a matrix

inversion to find the parameter vector c.

$$\begin{aligned} x(t_1) &= p_1, \quad x(t_2) = p_2 \\ \dot{x}(t_1) &= v_1, \quad \dot{x}(t_2) = v_2 \\ \ddot{x}(t_1) &= a_1, \quad \ddot{x}(t_2) = a_2 \end{aligned} \tag{3.7}$$

The minimum jerk trajectory can also be found for a velocity transition without considering the position. Setup the Euler-Lagrange equation again, but this time with v for velocity and $\ddot{v}$ for jerk.

$$v^*(t) = \underset{v(t)}{\operatorname{argmin}} \int_{t1}^{t2} \mathcal{L}(\ddot{v}, \dot{v}, v, t)\, dt = \underset{v(t)}{\operatorname{argmin}} \frac{1}{2} \int_{t1}^{t2} \ddot{v}^2\, dt \tag{3.8}$$

$$\frac{\partial \mathcal{L}}{\partial v} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{v}}\right) + \frac{d^2}{dt^2}\left(\frac{\partial \mathcal{L}}{\partial \ddot{v}}\right) = 0 \tag{3.9}$$

Which gives us the condition that the forth derivative of velocity, crackle, should equal zero.

$$\frac{d^4 v}{dt^4} = 0 \tag{3.10}$$

The minimum jerk trajectory for a velocity transition is thus given by a cubic polynomial that satisfies our state transition.

$$v(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 \tag{3.11}$$

A cubic function has four parameters, and we thus need to define four conditions for the trajectory. The most straightforward implementation is again to define a set of boundary conditions and then solve for c using a matrix inversion.

$$\begin{aligned} v(t_1) &= v_1, \quad v(t_2) = v_2 \\ \dot{v}(t_1) &= a_1, \quad \dot{v}(t_2) = a_2 \end{aligned} \tag{3.12}$$

3.2 Human Motion and Perception

3.2.1 Hand Motion

Flash and Hogan [23] conducted an interesting practical experiment on natural human motion between waypoints. The experiment was constructed as follows. A set of lights mounted through a table could be turned on or off at different posi-

tions. The test subjects were told to move their hand to the currently glowing lamp. The hand motions were then recorded and analyzed as one-dimensional trajectories between the points. The result showed that the way a human naturally moves their hand from one point to another closely resembled a minimum jerk trajectory. Note that this is independent of speed. A faster hand transition will still resemble a minimum jerk trajectory, just with a different time scale.

We assume that motions resembling natural hand movements will be perceived as smooth and non-robotic as they will be similar to a type of movement that humans are accustomed to. It is also observed that humans twist their heads in a similar fashion to reduce the stress on their bodies. Controlling the drone in a similar way to people's hand and head motion should be a good starting point for motion that feels natural for the human operator.

3.2.2 Partner Dance

An excellent example of jerk trajectory and its human perception is partner dances. Hence, in this section, we are going to take a closer look at the partner dance style called Lindy Hop, which originated in the thirties and forties in the Afro-American communities in Harlem, New York. It is danced in couples, with two dance roles, leading and following. The leader's task is to suggest what to do next, and the follower will follow these suggestions while at the same time adding his or her styling. Lindy Hop is danced to a wide range of music tempos, with moves ranging from slow and heavy to fast and acrobatic moves [20].

A core concept that is taught to new dancers is the notion of stretch and compression, the not so easily definable "the feel of the dance." This is, in simplified physical terms, the tensions between the dancers' connected hands and how this can be modulated for communication. A typical beginner lead problem is that this connection changes far too quickly, which makes it hard for the follower to respond in time. This behavior can range from annoying to painful and might even cause injuries. The rate of this change is, in physical terms, the jerk of the dance. An interesting additional observation is that it is possible to use a high magnitude of force when dancing to offer mutual support as long as the jerk is kept low. One technique of reducing jerk when high tension is needed is to signal the change early, giving both dancers more time to adjust to a connection equilibrium, with less jerk as a result. It is also possible to compare dance styles as having more or less jerk.

Popping and Locking is a style of HipHop [3] with very robotic and mechanical-looking motions. The name is very descriptive as it aims at popping the motion out from rest and quickly locking to zero speed when the endpoint is reached. The velocity between the start and stop is usually also kept constant for greater robotic-like effect. It is commonly performed to medium or low tempo music, so it is not the rhythm that is fast but really the way the motion is performed that makes all the difference for the viewer.

Another example is Blues [25], which is a partner dance performed to soft and relaxing blues music. This style of dancing is characterized by its low jerk and high connection awareness between the dancers and should, above all, feel soft. That does not stop people from making fast motions at times, but it is done with low jerk, which gives the dance its signature lag feeling. The follower is supposed to lag the motions suggested by the lead. The significant lag time, which both dancers have to consider, sets a good start for low jerk motions as accelerations in new directions can be extended over longer periods while still reaching their final state in time.

3.2.3 Buses and Braking

A braking vehicle is a classic jerk example. Imagine standing on a bus that is about to stop. The driver starts to brake smoothly, and you manage counteract the force, but you lose your balance when the bus finally stops. The reason for the loss of balance is not the braking force in itself but the sudden change in acceleration when the bus becomes stationary. The braking force only acts as long as we are moving forward but drops to zero instantly when the bus becomes static. This instant drop in acceleration produces a peak of high jerk, which is very hard to counteract, even if you could handle the braking force until that point. This is why a smooth braking procedure includes both a progressive press and a release of the brake pedal.

Strong breaking and jerky rides were among the top reasons for uncomfortable rides in public transport, according to Kottenhoff [16]. The same study mentions 0.6 - 0.9 $\mathrm{m\,s^{-3}}$ as the threshold for when passengers will lose their balance or even fall due to high jerk.

3.2.4 Animation and Movies

The main inspiration for this thesis was to improve the video quality produced by the drone by optimizing its motions for good video. Some informal discussions with filmmakers were done in the early phase of the project, and the key take-home message was ensuring stability. A video is only usable for production if it is not shaky. Keeping a static state is thus of high importance. Nevertheless, motions are still commonly used to increase interest in a sequence.

A common example is the use of sliding motion, where the camera is slowly moved sideways while still keeping everything else stable, especially rotations. These slides are primarily performed horizontally, so the heading control must be good to guarantee a stable heading. An interesting case happens when a small correction of the heading is needed to keep a framing as intended. An aggressive correction would need to be cut out of the final delivery as it would look unsatisfactory compared to all other footage. A smooth correction could instead be used as one clip in the editing process, as the heading correction would not be a disturbing movement. There is one more thing that could make even a smooth transition unusable: overshoot, which would cause an unstable look. Effectively rendering the correction move unusable in the editing process.

When asked, few filmmakers will be able to explain with what function their camera moves, but everyone will say that it has to be smooth. A computer animator, on the other hand, will be able to say what interpolation method they are using. There are several options, but some of the most common are listed in table 3.1.

Name	Order	Smoothness
Constant	0	-
Linear	1	C^0
Smooth Step	3	C^1
Bezier Curve	3	C^1
Smoother Step	5	C^2

Table 3.1: Common easing types for animation and viusal effects softwares, their polynomial order and smoothness.

Constant interpolation is used for jump cuts and instantly moving or changing some property. Linear interpolation is useful when a constant velocity is needed, such as a train traveling on a railroad. The two most common types are smooth step and bezier curves. These are, in essence, the same cubic polynomial, but the

smooth step is defined with zero velocity at the initial and final state, while it is possible to have non-zero velocities at the boundary state for the bezier curve. The smoother step is not as common, but it is found in some software packages. It defines both the boundary velocity and acceleration as zero. This makes smoother step C^2 smooth over multiple connecting boundaries as the acceleration always is zero at keyframes. Smooth step and bezier curves are only C^1 smooth as they have unbounded acceleration connecting boundaries and, therefore, discontinuous acceleration. The smoother step was suggested by Perlin [5] as a fix to the discontinuous acceleration and the non-optimal behavior of the smooth step.

The wide use of only cubic-based interpolation for animation puts a dent in this thesis argument that minimizing jerk and C^2 smooth is the optimal goal for natural trajectories. However, the disadvantage of using cubic polynomials has been noted by authors like Perlin. It should instead be believed that animation packages still use these C^1 smooth curves because of their intuitive editing interface and the direct hardware support in GPU drivers [14].

3.3 A Smooth Summary

This chapter presented motion smoothness as the central aspect that everything is based on for the remaining of this thesis. Piecewise functions with at least C^2 smoothness and minimum jerk turn out to be a good approximation for several real-world motions primarily related to humans and body motion. Quintic minimum jerk polynomials are the basis of this thesis's continued work.

Chapter 4

The Control Problem

The Control Problem
Given a system (S) with measured signals (y), find a control input (u) such that the system state (z) is as close as possible to a reference signal (r), despite process noise (w), measurement noise (v) and system variations, while maintaining reasonable control inputs for the system.

As defined by Glad and Ljung in [9]. A device or software that automatically generates a suitable control input is usually called a controller C. The complexity of a controller can vary greatly but is usually dependent on the complexity of the system that it acts upon. One key component of adding a controller to a system is that it forms a feedback loop which is essential for achieving stability and robustness.

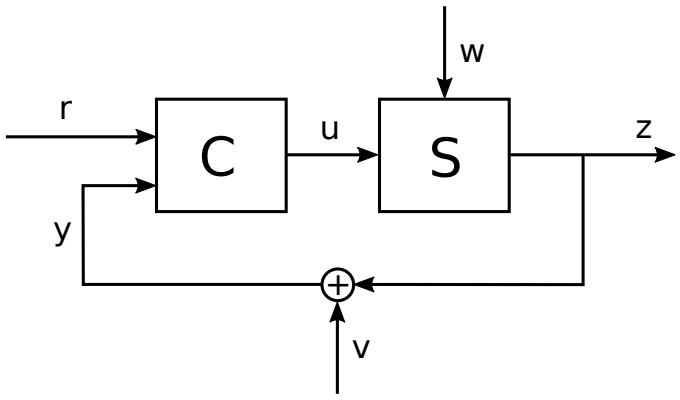

Figure 4.1: The basic structure of a feedback controlled system S, under the disturbance w, using the control system C which reads the state z using some sensor y with measurement noise v. The goal is to keep z as close as possible to the reference r.

The following chapter aims at shortly introducing some of the components for solving the control problem. This is not an exhaustive list, but it brings forward some previous work in this area. The controller in Figure 4.1 is generally broken down into two pieces, an observer and a controller. This is because the measured signal y contains noise, and it might not be possible to measure the wanted system state directly. An observer's job is to make the best possible estimate of the system S from the measured signals y. The controller can then use the estimated states to find a suitable control input u. The observer and the controller can be further divided into two main categories, linear and non-linear. A linear setup is usually much easier to implement and analyze and will also be the optimal solution for a linear system. Some non-linear systems are controllable even with a linear observer and controller, but complex non-linear observers and controllers are needed for many non-linear systems.

The final part of the control problem is the reference generation. The reference can sometimes be as easy as a static number. For example, keep this motor rpm stable forever, but it is usually more involved. The reference is a request from either a human operator, a type of artificial intelligence, or a logic program that decides what should be done by the controlled system. Who or what the request comes from is not a relevant question here, as the thesis focuses on how. The how is divided into two categories: real-time input and preplanned input. A real-time input means that a pilot or an operator continuously guides the system to where he, she, or it wants the state to be. An example is a fighter jet pilot who requests the airplane to roll in a fly-by-wire system. The pilot does not have direct control of the actuator outputs but instead requests a result, roll the aircraft. In that way, the control system can work with fulfilling the request and battle any unwanted motion from the turbulence that affects the airplane. A preplanned example would be a robotic arm or a vehicle going to a predefined waypoint. The entire trajectory can, in this case, be calculated beforehand.

4.1 Controller

The controller's task is, as stated above, to compute a suitable input signal to the system S such that the state y is kept as close as possible to the reference signal. There are many different approaches for designing a control law for the calculation of suitable control input for a given system. It should be noted that the subset of lin-

ear controllers is much more studied than the larger field of non-linear controllers and that their properties and behavior are easier to predict. Another important aspect is that the optimal controller for a linear system is also a linear controller. This means that non-linear controllers only have to be considered if the system is non-linear enough to have a significant impact on the result [9].

4.1.1 Linear Control

Linear controllers can, in general, be seen as calculating the control input u as the difference between a reference and, by the observer estimated states, $\hat{x}$, multiplied by some constant gain.

$$u = L_r r - L\hat{x} \tag{4.1}$$

There exist many methods of organizing and determining the values for L_r and L, but two common methods are PID control and Linear Quadratic control.

PID or Proportional, Integral, and Derivative has a very long history and is still commonly used due to its pedagogical structure and well-documented implementation details. Its main characteristic is that $L_r = L$ and the introduction of an integral term I. Determining the constants P, I, and D is usually done through manual tuning in simulations and on the real system, but there exist a lot of tools for calculating good starting points.

$$u = P(r - \hat{x}) + I \int (r - \hat{x})dx \; + D(\dot{r} - \dot{\hat{x}}) \tag{4.2}$$

Linear Quadratic control or LQ is another linear control law but approaches appropriate gains differently. It offers a framework for finding the optimal gains L_r and L by finding the set that minimizes the weighted quadratic error and the input actuation.

$$min \|r - \hat{x}\|_{Q_1}^2 + \|u\|_{Q_2}^2 \tag{4.3}$$

Note that both r and $\hat{x}$ can define multiple time derivatives and multiple degrees of freedom. The system input u can also be multidimensional. The gains found might very well be exactly the same for a PID controller as for the LQ controller for single input and a single output, SISO systems. However, PID controllers have no structure for handling multiple inputs and multiple outputs, MIMO systems. Tuning is moved from a direct operation on the feedback gains to changing the weight/cost of the terms to be minimized.

4.1.2 Non-Linear Control

Non-linear control is needed when some part of the system is not linear or linear enough to behave well under linear control laws. One common problem for linear controllers is actuator saturation of the control signal. The demanded control input is larger than what the system's actuator can produce. Examples of this are higher than possible voltages or currents, higher flows than possible, or larger angles than mechanically designed for. Saturation is a common problem for underwater vehicles with a maximum possible thrust from the thrusters. One common approach for solving this problem is Model Predictive Control, MPC.

The name introduces the method quite well, and it works by predicting the system response from a future input signal using system modeling. The control input should minimize the same quadratic minimization problem as the LQ controller but with the difference that the MPC can consider system constraints such as saturation. It does this by finding the minimum using iterative Quadratic Programming instead of an analytical solution. The technique is very powerful and can solve many problems where a linear controller would underperform or even be unstable.

The big drawback of using Model Predictive Control is its computational cost. Model-based optimization has to be performed on every time step, which is possible for slow systems or controllers with access to enough computational power. The Blueye Pioneer does not have enough free computational resources for its speed, so an MPC approach is unfortunately not feasible.

4.2 Observer

An observer is a filter that takes available noisy sensor readings as input and uses that data to estimate the true state of the system. The estimation task is two-fold, reduce noise, and find needed implicit states. An implicit state is, for example, the velocity of an object when we only have positional measurements. There are many different approaches and types of observers, and they can, like the controllers, be divided into linear and non-linear observers.

4.2.1 Linear Observation

A simple example of a linear observer is the task of estimating a static state from a stream of noisy measurements. Let the measurement be called y_i, the true state x,

and assume white noise v.

$$y_i = x + v \tag{4.4}$$

The optimal observer for minimizing the mean square error for this problem is the average of all available measurements. It is done by summation of all measurements and divided by the current sample index k. y is here a vector of k observations, one for each timestep.

$$\hat{x}_k = \frac{1}{k}\sum_{i=0}^{k} y_i \tag{4.5}$$

Storing every single measurement and performing the summation for every time step would be computationally expensive and wasteful on a real-time system, so it is better to rewrite it on a recursive form.

$$\hat{x}_k = \frac{k-1}{k}\hat{x}_{k-1} + \frac{1}{k}y_k \tag{4.6}$$

k starts at 1 and $\hat{x}_0$ is intialized as zero. This version can run in real-time with a very low memory footprint and will converge to the true value quickly. A state that needs a control loop is usually not static, so a method for handling varying states is needed.

The optimal observer for any observable linear system with white noise is described by the Kalman Filter [13] first described in the sixties and used on the NASA Apollo missions on the guidance computers. It is optimal in the sense that it minimizes the quadratic error between the estimated and true state.

$$min\|x - \hat{x}\|^2 \tag{4.7}$$

The linear system has to be described in discrete state-space form as follows.

$$x_{k+1} = Ax_k + Bu_k + Nw_k \tag{4.8}$$

$$y_k = Cx_k + v_k \tag{4.9}$$

A describes the system's dynamics, B is the system response to inputs, and C is what we measure with available sensors. The system noise at timestep k is described by w_k, and the measurement noise at timestep k by v_k. w_k and v_k are assumed to be mutually independent. The system is observable, meaning it is

possible to estimate its state from the chosen measurements if and only if the observability matrix rank is full. n is the size of the square matrix A.

$$rank \begin{bmatrix} C \\ CA \\ CA^2 \\ ... \\ CA^{n-1} \end{bmatrix} = n \tag{4.10}$$

The optimal discrete observer is then given by a two-step predict and update routine, beginning with prediction.

$$\hat{x}_{k+1} = A\bar{x}_k + Bu_k \tag{4.11}$$

The update step takes the predicted state and compares it with the latest measurement y_k such that a refined a refined estimate, $\bar{x}_k$, of the state vector x_k can be made.

$$\bar{x}_k = \hat{x}_k + \bar{K}(y_k - C\hat{x}_k) \tag{4.12}$$

$\bar{K}$ can be iteratively estimated for varying disturbances. The expected process noise is for example higher when the thrusters of the drone are active. This can be done in the following way.

$$P_{k+1} = A\bar{P}_k A^T + NR_1N^T \tag{4.13}$$

$$S_k = CP_kC^T + R_2 \tag{4.14}$$

$$\bar{K}_k = P_kC^T S_k^T \tag{4.15}$$

$$\bar{P}_k = (I - \bar{K}_kC)P_k \tag{4.16}$$

R_1 and R_2 are the covariance matrices describing the process noise w and the measurment noise v.

The Kalman Filter is both computationally efficient and optimal for problems where it is applicable, which makes it popular in a wide range of applications. It works well for some of the situations encountered by an underwater drone but not all, and it is therefore important to introduce a couple of non-linear observers.

4.2.2 Non-Linear Observation

There are a couple of reasons why non-linear observers are essential to underwater drone state estimation. Three examples of common problems found in Fossen's book *Guidance, Navigation, and Control of Marine Craft* [7] are quadratic damping, positioning, and attitude estimation. The impact of non-linearities varies from minor insignificant errors to significant impacts, making the system unstable. Taking extra care when developing the observer is therefore of importance.

Hydrodynamic drag from moving in the water is not linear as it is on the form

$$D(v) = D_{\texttt{depth}}v + D_{\texttt{quadratic}}v|v| \, . \tag{4.17}$$

A Kalman Filter which includes the system input signal u, will, with a system that includes quadratic drag, overestimates the predicted speed, which will cause errors in the state estimate. A common solution to this problem is the Extended Kalman Filter (EKF), which, as the name suggests, extends the linear Kalman Filter to non-linear functions by linearizing them around the current state. This works well for dealing with quadratic drag as it is easy to linearize at any point, and the linearization is valid for a large change in velocity. An Extended Kalman Filter also works well for the situation of stabilizing the horizontal position of an underwater vehicle [4]. The position and velocity can be expressed in earth frame, while the dynamics are expressed in body frame. The non-linear part of such a setup is the rotation matrix from body-frame to earth frame and its corresponding sine and cosine functions.

The third common need for a non-linear observer is attitude estimation. This is the task of estimating the current orientation of the underwater drone. The Euler angles representation, pitch, roll, and yaw are common and easily understood but highly non-linear. An EKF can solve the non-linearity of the space [7], [11], but the implementation is both cumbersome and not optimal. One of its main drawbacks is the presence of gimbal lock, which is when its two rotation axes align, and the system loses one degree of freedom.

A better approach is presented by the work of Madgwick [19] who uses quaternions for the rotation representation and a gradient descent method for the filter. The advantage of using quaternions as the mathematical representation of rotations is that they do not exhibit gimbal lock. They are efficient, and any rotation is described as a simple multiplication. The easiest way to describe how they work

is to introduce them as an extended complex number with three imaginary parts instead of one. A one-axis rotation can be described by the multiplication of two complex numbers. The multiplication of two quaternions can describe a three-axis rotation. Say that we initially have a rotation A to B and want to add the rotation B to C, then multiply them to get the combined rotation.

$$q = w + xi + yj + zk \tag{4.18}$$

$$_{C}^{A}q = {}_{C}^{B}q \otimes {}_{B}^{A}q \tag{4.19}$$

A vector v can be rotated from frame A to frame B by the following multiplication, where the vector is put into a quaternion container with the element $w = 0$. And q^* is the quaternion conjugate.

$${}^{A}v = 0 + v_x i + v_y j + v_z k \tag{4.20}$$

$${}^{B}v = {}_{B}^{A}q \otimes {}^{A}v \otimes {}_{B}^{A}q^* \tag{4.21}$$

A filter based on this method can be constructed by a similar predict and update structure as the Kalman Filter. The prediction is based on angular velocity measurements provided by the gyro in the IMU sensor and is done through simple discrete integration. Note that prescript notation of sensor readings being in sensor frame S and the estimated quaternion as earth frame to sensor frame ${}_{E}^{S}\hat{q}$ rotation.

$${}_{E}^{S}\dot{q}_k = \frac{1}{2}\, {}_{E}^{S}\hat{q}_{k-1} \otimes {}^{S}\omega_k \tag{4.22}$$

$${}_{E}^{S}\hat{q}_k = {}_{E}^{S}\hat{q}_{k-1} + {}_{E}^{S}\dot{q}_k dt \tag{4.23}$$

The gyro-based prediction works great for tracking fast motions, but it will fail over time as the estimate will drift due to bias in the measurements. The bias can be corrected by rotating the expected gravity ${}^{E}d$ vector by the estimated quaternion and comparing it to an acceleration sensor reading ${}^{S}s$. A correction step can then be done to minimize the error between the expected gravity vector and measured gravity.

$$min\|f({}_{E}^{S}\hat{q})\| = min\|{}_{E}^{S}\hat{q}^* \otimes {}^{E}d \otimes {}_{E}^{S}\hat{q} - {}^{S}s\| \tag{4.24}$$

An efficient optimization method for finding the above minima is to use an iterative

gradient descent algorithm.

$$\overset{S}{E}\hat{q}_{k+1} = \overset{S}{E}\hat{q}_k - \beta\frac{\nabla f}{\|\nabla f\|} \tag{4.25}$$

The above algorithm will eventually find a quaternion q such that the error between the expected gravity vector and the measured gravity vector is minimized. β is the correction step size and can be seen as the parameter weighting accelerometer measurements against the gyro measurements. This works very well, and acceleration that is not due to gravity can be filtered away using small β values. A correct heading is added in the same way as for the acceleration by comparing an expected and measured magnetic flux vector.

4.3 Reference Generation

Fossen [7] introduces three overall cases for reference generation for marine crafts which also applies to other systems.

- Setpoint Regulation, the basic approach, keeps a constant state
- Trajectory-Tracking Control, time varying changes in some or all of the states. Used for velocity, position, and heading changes when the timing is important.
- Path Following Control, time-independent navigation, follows a predefined state change as the system propagates along the defined path.

The problem statement for this thesis is related to Trajectory-Planning Control as the work is focused on velocity and position changes due to pilot input commands. The methods for generating these trajectories vary from simple linear filters to advanced minimization problems with motion constraints and collision avoidance. Three groups of trajectory generation are presented; filter-based, geometrical, and optimization-based, with two different cost metrics.

4.3.1 Filter Based Trajectory Generation

Filter-based trajectory generation takes some usually computationally inexpensive linear filter and filters the input commands from a human. This is commonly a low pass filter or a model-based filter that moves the reference for the controlled

system. One model-based filter is a mass-damper-spring system which is used by Dukan [4] on the SF30k ROV at NTNU for velocity ramping. A problem for linear filters is that they only work well for a small envelope of step sizes and a non-symmetrical shape for the start-up and converge phases. The start-up phase tends to have high jerk, and the converge phase is comparatively slow, causing a reported feeling of drift and being out of control.

A filter can also be on the form of an integrator. A pilot can directly define the velocity, and the position is then integrated over time. Velocity integration produces a continuous position which makes it usable for position hold. It was the initial method of the Blueye Pioneer and thus served as the baseline case for this thesis.

4.3.2 Geometrical Trajectory Generation

A different approach to trajectory generation is geometrically based approaches. This method is, for example, used on naval crafts [7] where it provides a suitable trajectory to follow. This approach assumes that the operating velocity is mostly a constant cruise velocity and the direction straight. Imagine the trajectory being a series of position waypoints and draw straight lines between them. The linear parts will work fine with a constant cruise towards the next waypoint. Passing a waypoint is problematic as it instantly changes to a new direction. This can be mitigated by placing a circle with some design radius tangental to both the inbound and outbound lines. This provides a C^1 instead of C^0 smooth trajectory but will still be discontinuous in acceleration when entering and exiting the circle. The technique is mainly used for transitions between position waypoints, and it is thus not a suitable approach for handling human real-time velocity inputs.

4.3.3 Optimal Trajectory Generation

Reference generation can be approached similarly to the optimal control problem by formulating cost metrics that should be optimized. A common approach is to formulate the reference generation as a minimum time problem given a set of constraints, usually with bounded velocity, acceleration, and jerk. This results in a trajectory with constant jerk segments and linearly ramped accelerations which can be efficiently solved using iterative solvers [10], [15]. Jerk limited motion is fundamental for CNC operation, and Trinamic has a good presentation page on

why, among other reasons, less beer is spilled when the motion jerk is limited [28]. Dukan [4] implemented this method for ROV waypoint transitions and called it a constant jerk trajectory, improving performance and usability as a result. Time-optimal trajectories are essential when there is a need for pushing hardware to the limit while still guaranteeing operation within its physical constraints. The goal of this thesis is, in contrast, to find an optimal shape of motion transitions. This requires a different cost function, determined to be the minimum jerk trajectory given a maximum jerk.

Most examples of the usage of minimum jerk trajectories use either constant time [8], time derived from a known event such as intercepting a ball [22], or direct minimization as part of the control loop of a model predictive controller [21]. The limitation and minimization of jerk for a smoother response to human input should not affect the responsiveness against disturbances. Therefore, it is not suitable to include the jerk minimization as part of the control loop. The next chapter presents a method of using minimum jerk trajectories for online velocity reference generation between any state transition.

4.3.4 Comparison of Reference Generation Methods

It can be hard to appreciate the difference between the reference interpolation methods given above from just a description. Plotting them together in one figure helps with showcasing the difference. Figure 4.2 and 4.3 illustrate the difference between pure integration, linear filter, minimum time and minimum jerk given then same jerk constraint of 10 $\mathrm{m\,s^{-3}}$ when tuned at a setpoint step in velocity of 0.1 $\mathrm{m\,s^{-1}}$, as shown in Figure 4.2. The pure integration case is the direct usage of the user input without taking any constraints or dynamics into consideration. It has the fastest possible response to the input, which can be seen in Figure 4.2 leading the position reference and always equal to the user input for the velocity reference. The downside is that it requires enormous and unrealistic amounts of acceleration and jerk for the controlled system to perform the transition. The other methods look very similar for the position reference with a slight lag compared to the pure integration case. The lag is caused by the smooth ramping of velocity, and the position stays slightly behind until the reference stops, and the same lag in stopping results in the same final position.

The difference between the methods is more evident when looking at their time derivatives. The minimum time reference utilizes the maximum possible

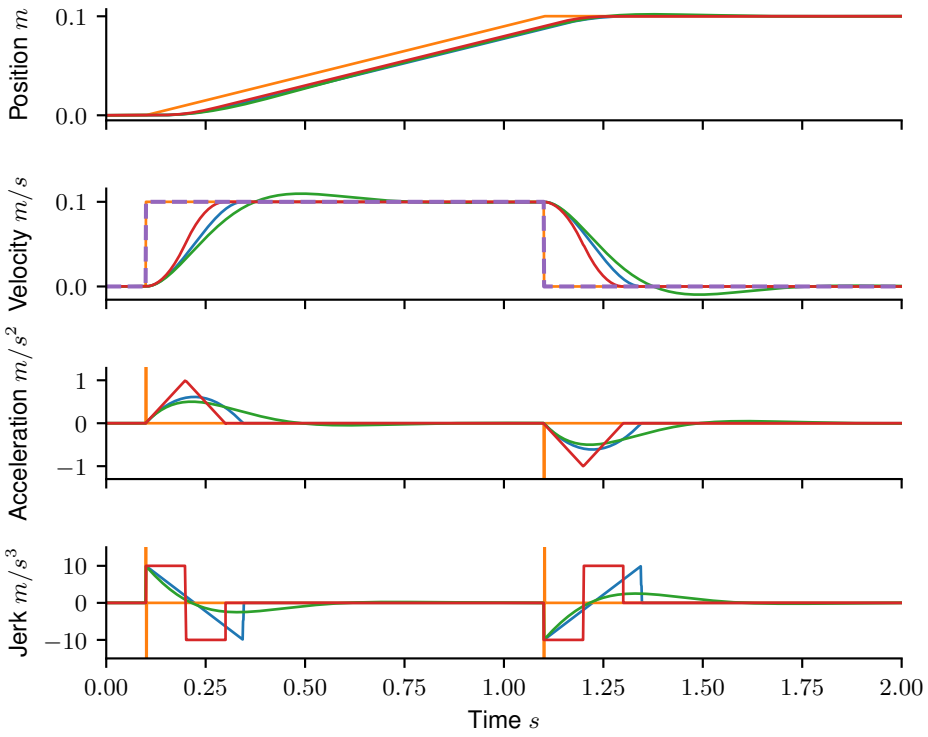

Figure 4.2: Comparison of pure integration (orange), linear filter (green), minimum time (red), and minimum jerk (blue) based reference generation methods. Showing the tracking of a small stepped user input (dashed violet).

jerk throughout the velocity ramping, causing the reference acceleration to become triangle-shaped with a higher peak acceleration than the other two methods. The minimum jerk trajectory-based method is tuned such that the peak jerk equals the constraint. The solution is, in this case, a linear ramp for jerk and a quadratic acceleration profile with a flat peak acceleration which is only 62% of the peak acceleration of the minimum time method despite only taking 15% longer time to complete. The linear filter designed as a mass-damping-spring system begins with the same jerk as the other two but has a much longer tail and lower jerk in the second half of the transition.

Figure 4.3 uses exactly the same setup for handling user input changes as in Figure 4.2 but with a ten times larger step in requested velocity. The big difference that should be noted first is that the linear filter now has a maximum jerk much larger than intended, ten times larger than the other methods at 100 $\mathrm{m\,s^{-3}}$ and a maximum acceleration more than twice as high as for the minimum jerk trajectory.

This is clear in the velocity curve, where the linear filter has a much faster rise time. It is also visible for the position, where the green filtered reference curve is far ahead of the minimum jerk and minimum time solutions. The linear filter is better than the unrestricted direct integration, but its property of proportional jerk to the velocity step size, non-symmetric profile, and overshoot only make it a little bit better, not good.

The two approaches to optimal trajectories, minimum time and minimum jerk remain within the jerk bounds and keep their shapes consistent. Both methods have an increase in acceleration which can be constrained if needed. Two properties are in favor of the minimum jerk trajectory in this thesis. Firstly, this is the shape of human motions, and the focus is on optimal shape, not optimal time. Secondly, the optimal time problem is not described by one polynomial but by piecewise polynomials. These need to be optimized. Therefore, the minimum jerk trajectory is chosen as the method of reference generation in this thesis.

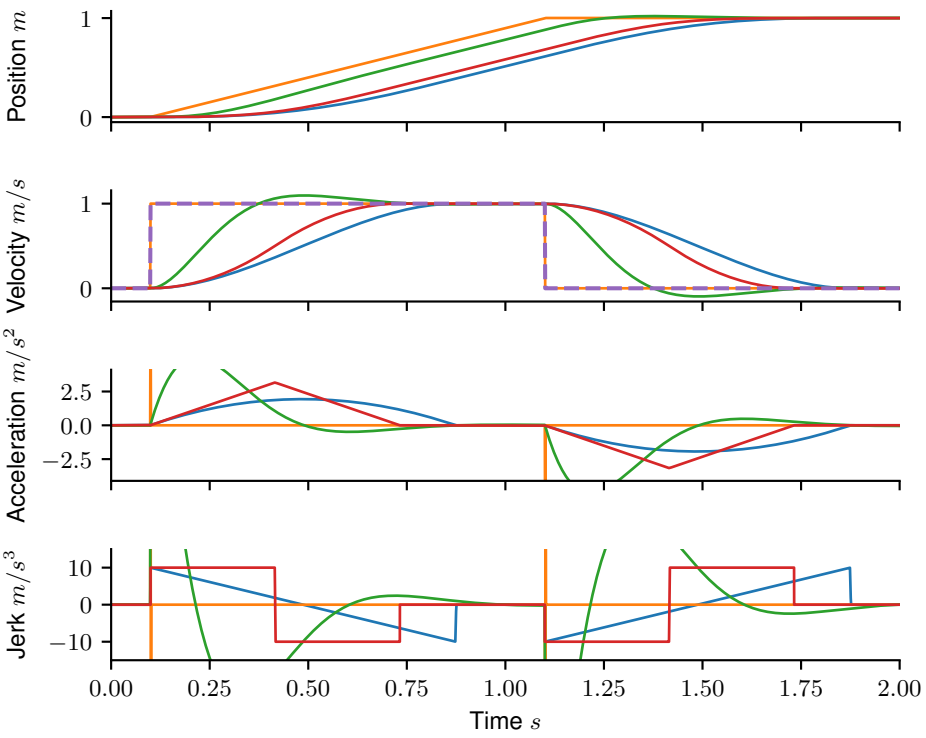

Figure 4.3: Comparison of pure integration (orange), linear filter (green), minimum time (red), and minimum jerk (blue) based reference generation methods. Showing the tracking of a large step user input (dashed violet).

Chapter 5

Implementation

5.1 Motivation

This chapter describes the methods implemented on the Blueye Pioneer underwater. It begins with the description of the core contribution of this thesis, which is Online Minimum Jerk Velocity Trajectories. It can be used for high-quality reference ramping with smooth and continuous reference states for a position, velocity, and acceleration.

The second part presents how the observers for the control system were implemented and with what assumptions.

The third part covers the control approach, forming a feedback loop using the observed states and the minimum jerk trajectory reference. The control method is linear, and the focus is on using all of the defined reference states efficiently.

5.2 Minimum Jerk Velocity Reference

The goal of the Minimum Jerk Velocity Reference method, later referred to as MJVR, is to generate a good reference state from a velocity set point given by a human pilot. This set point can have substantial and fast changes, which are hard to track and cause vibrations in the system. The proposed method addresses these problems by enforcing C^2 smoothness as a design parameter. Guaranteeing continuous acceleration and with a configurable smoothness as a maximum jerk.

This section describes the process of generating a reference with continuous acceleration and planned jerk for any random velocity input from a human pilot.

The method is based on the analytical minimum jerk trajectory for any velocity change which is also the reason for the name. The solution is also explicit with small computational load and can thus run in real time on lower power platforms.

The remaining of the section investigates the process and details of implementing online minimum jerk velocity trajectories. The process is two fold: It comprises of defining how to find and calculate the analytical minimum jerk velocity trajectory for any velocity state change over a specified time frame. The second part is defining the time given to the transition change, later referred to as converge time. Calculating the converge time is crucial for making the method a real-time online method. It is crucial as the trajectory has to be identical irrespective of at what time during the transition the trajectory is calculated.

Combining the two steps makes it possible to do online updates without the knowledge whether the velocity set point from the pilot has changed or not. An unchanged set point will regenerate the same trajectory for the remaining transition. A changed set point will result in a new trajectory reaching this new set point, while still keeping a continuous acceleration state and staying within the design jerk parameter.

The section also describes how to use the polynomial velocity to integrate a position reference which is useful for systems where position drift is undesirable. In addition, a method for limiting the maximum acceleration is also described as some systems might be limited by maximum acceleration before jerk becomes a problem.

5.2.1 Minimum Jerk Velocity Trajectory

As the jerk is the derivative of acceleration, the third time derivative of the position x and it can be written as $\dddot{x}$ using Newton's dot notation. The velocity is the first time derivative of the position is written as $\dot{x}$. The minimum jerk velocity trajectory between two velocities is the trajectory that minimizes the squared jerk over that transition. Let the transition happen between the time t_1 and t_2, and the problem can be written as follows.

$$\dot{x}(t) = \arg\min_{\dot{x}(t)} \frac{1}{2} \int_{t_1}^{t_2} \dddot{x}(t)\, dt \tag{5.1}$$

It was shown earlier in section 3.1.2 that a polynomial meets this on the form

$$\dot{x}(t) = c_1 + c_2 t + c_3 t^2 + c_4 t^3 \, . \tag{5.2}$$

This is a cubic polynomial interpolation between the current and wanted states. The trajectory is defined between the start point and endpoint of the trajectory

$$t_1 \leq t \leq t_2 \, . \tag{5.3}$$

It was also shown in the chapter 3.1.2 that the trajectory could be found by solving for the polynomial's coefficients using a system of linear equations. The number of needed equations is four as the number of unknown variables is four. The current reference velocity is known, and so is the target velocity. The target acceleration is always zero as the reference velocity should converge to the target and then remain constant. Let us assume that the current reference acceleration is known and that t_1 equals zero

$$\dot{x}(t_1 = 0) = c_1 = v_1 \, , \tag{5.4}$$

$$\ddot{x}(t_1 = 0) = c_2 = a_1 \, , \tag{5.5}$$

$$\dot{x}(t_2) = c_1 + c_2 t_2 + c_3 t_2^2 + c_4 t_2^3 = v_2 \, , \tag{5.6}$$

$$\ddot{x}(t_2) = c_2 + 2c_3 t_2 + 3c_4 t_2^2 = 0 \, . \tag{5.7}$$

Rewritten to matrix form

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & t_2 & t_2^2 & t_2^3 \\ 0 & 1 & 2t_2 & 3t_2^2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} v_1 \\ a_1 \\ v_2 \\ 0 \end{bmatrix} \tag{5.8}$$

The solution is found by solving the equation

$$Ac = b \Rightarrow c = A^{-1} b \tag{5.9}$$

Finding c is inexpensive as the matrix inversion is of low dimension. It is despite that recommended to use built-in solvers for this operation as it improves code readability, performance, and numeric stability. Most packages offering linear algebra functionality have this built-in. Python has the Numpy package,

```
numpy.linalg.solve(A, b)
```

and C++ has options like Armadillo with

```
arma::solve(A, b);
```

The problem can be further optimized by observing that this particular problem will yield $c_1 = v_1$ and $c_2 = a_1$. This reduces the problem to two by two matrix inversion. However, the gains compared to an optimized solver will probably be minimal, and the details of implementing it is left to the reader.

The result is a polynomial that can be evaluated at any point from $t_1 = 0$ to t_2, which also is the minimum jerk trajectory for said state transition. Figure 5.1 shows the resulting plots for the velocity trajectory and its derivatives.

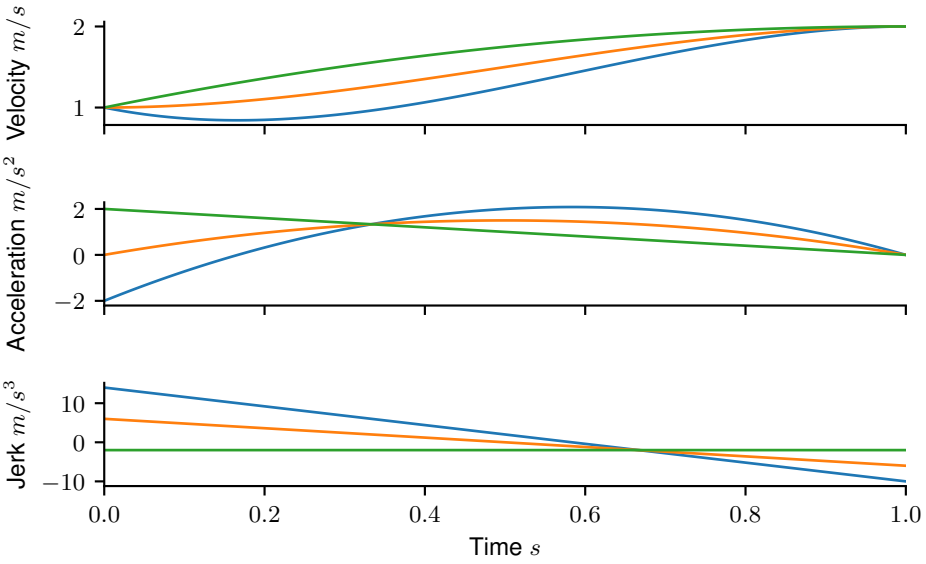

Figure 5.1: Minimum jerk velocity trajectories between two velocities for three different initial acceleration states $a_1 = -2, 0, 2$.

5.2.2 Converge Time

The section above showed how to find and solve the minimum jerk trajectory for a given set of initial and final conditions assuming a fixed time. The constant time assumption is problematic for online applications for two reasons. The first is that the acceleration and jerk will grow very large for significant transitions, as shown in Figure 5.2. Additionally, minor changes will be too slow for acceptable response times if the converge time is tuned for the maximum possible transition. The result

is that this approach only gives a good trajectory for one transition step size. All others will be either be too slow for the user, or too aggressive for the system. Last but not least, a constant time approach will give varying responsiveness of the system for a user who is changing the target velocity in real-time.

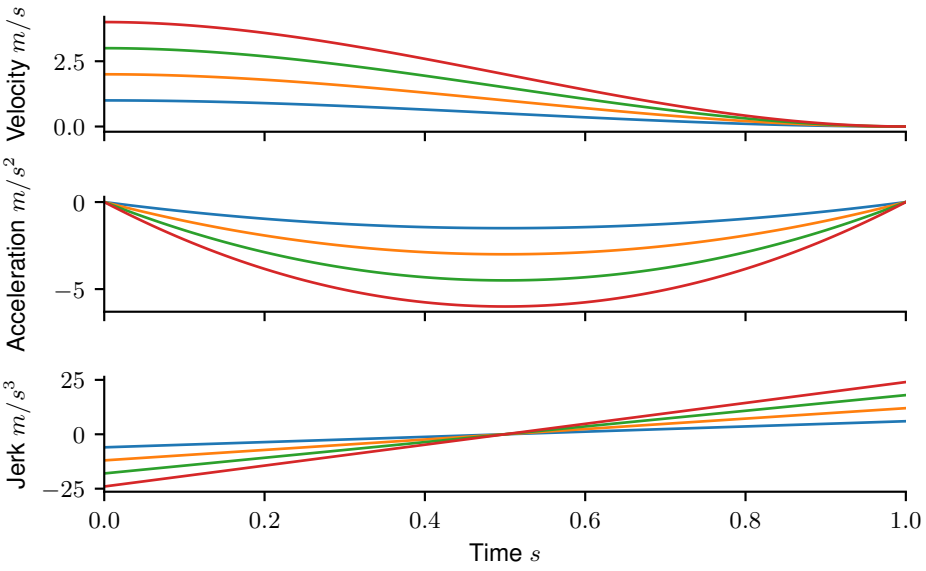

Figure 5.2: Minimum jerk velocity trajectories with constant converge time between a set of linearly-spaced initial velocities. Acceleration is zero for both the initial and final states.

The second problem with constant time is that the trajectory can not be updated in a real-time process. Figure 5.3 shows the result of updating a trajectory along the way with a new initial state but the same target state. The expected behavior is that the trajectory should be the same if it is recalculated at any point along the original trajectory. This is not the case with a constant time assumption as the target always will stay equally far away in time. Figure 5.3 shows the effect of updating with a constant converge time of one second. Each update is performed at a marked dot.

It turns out that both the above problems are solved by not assuming constant converge time. Instead, calculate the converge-time based on the current state. The trajectories found using this calculated time have to fulfill the following two properties:

1. Consistent finishing time for recalculated trajectories.

2. Predictable acceleration response for any velocity change.

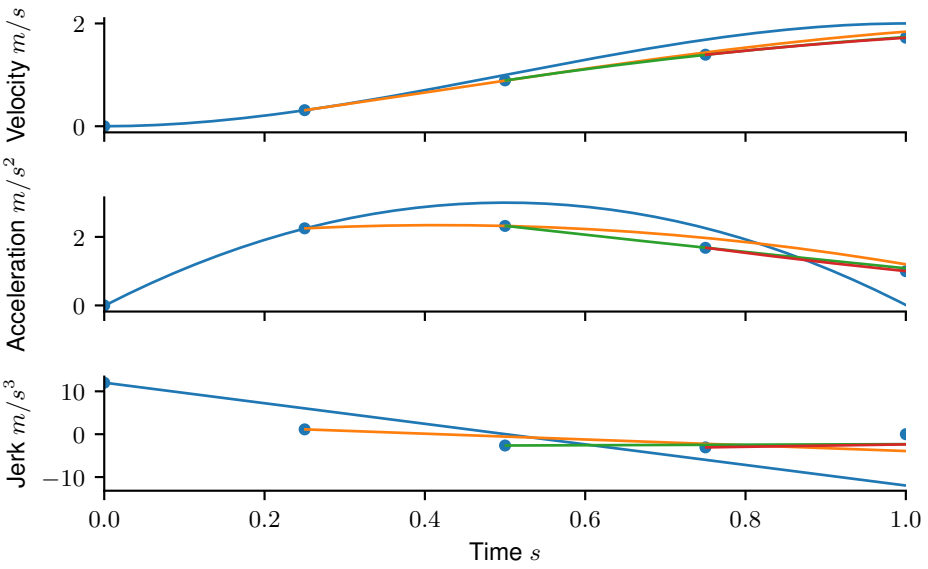

Figure 5.3: Minimum jerk velocity trajectory recalculated multiple times during a transition towards a constant target. The setup assumes constant times, which also is the culprit for the warped result.

Many different methods were tested to solve this problem, and it was eventually found that a straightforward metric can be used for solving both points, plan for a constant end jerk. This does not mean that jerk is added to the trajectory calculation step but that we pick a converge time with a specific final jerk value.

The end jerk for any state transition with known initial and final velocity and acceleration can be expressed as an inverse quadratic function. t_2 is the time for arriving at the final state and t_1 is the time for the start of the transition.

$$j_{end} = \left| \frac{6(v_2 - v_1)}{(t_2 - t_1)^2} + \frac{2(a_2 - a_1)}{t_2 - t_1} \right| \tag{5.10}$$

The equation can be simplified by assuming $t_1 = 0$ which leaves t_2 as the duration of the transition. a_2 can also be removed as it is set as zero in the problem statement. The remaining equation is as follows

$$j_{end} = \left| \frac{6(v_2 - v_1)}{t_2^2} - \frac{2a_1}{t_2} \right| . \tag{5.11}$$

j_{end} is then treated as a design parameter which means that it is possible to solve for t_2 as v_1 and a_1 are the current wanted state and v_2 is the wanted final state. The

easiest solution is to first split the absolute value operation into two cases and then reorganize the two cases on standard quadratic form.

$$j_{end}t_2^2 + 2a_1t_2 - 6(v_2 - v_1) = 0 \tag{5.12}$$

$$-j_{end}t_2^2 + 2a_1t_2 - 6(v_2 - v_1) = 0 \tag{5.13}$$

These two equations usually returns four distinct roots. All are mathematically correct, but two will be complex solutions and can be discarded as we know our converge time will be a real number. The other two roots are real, and we always want the largest one. Negative solutions are correct from a polynomial standpoint but not valid as the trajectory always is predicted into the future. To summarize, the largest real root is the correct solution from the set of 4 possible roots.

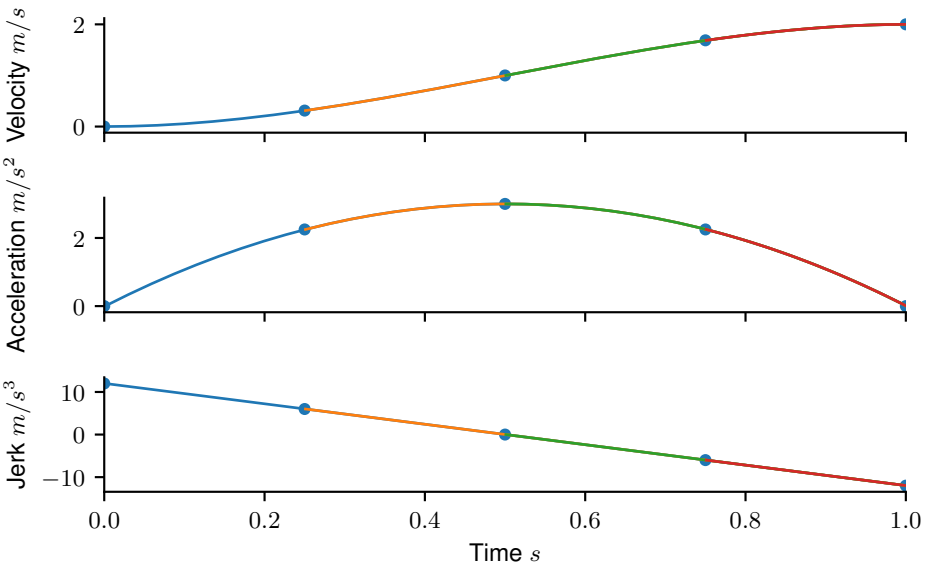

Figure 5.4: Minimum jerk velocity trajectory recalculated multiple times during a transition towards a constant target. The remaining converge time is calculated from the states with a planned end jerk.

Figure 5.4 shows that the trajectory now can be recalculated at any time during the transition with consistent results. This means that the first property is fulfilled. The end jerk method of finding the remaining converge time is consistent with calculating the trajectory from the initial and final states.

Figure 5.5 shows the reference trajectory for different magnitudes of change in velocity. Acceleration is zero for both the initial and final states. Note how the

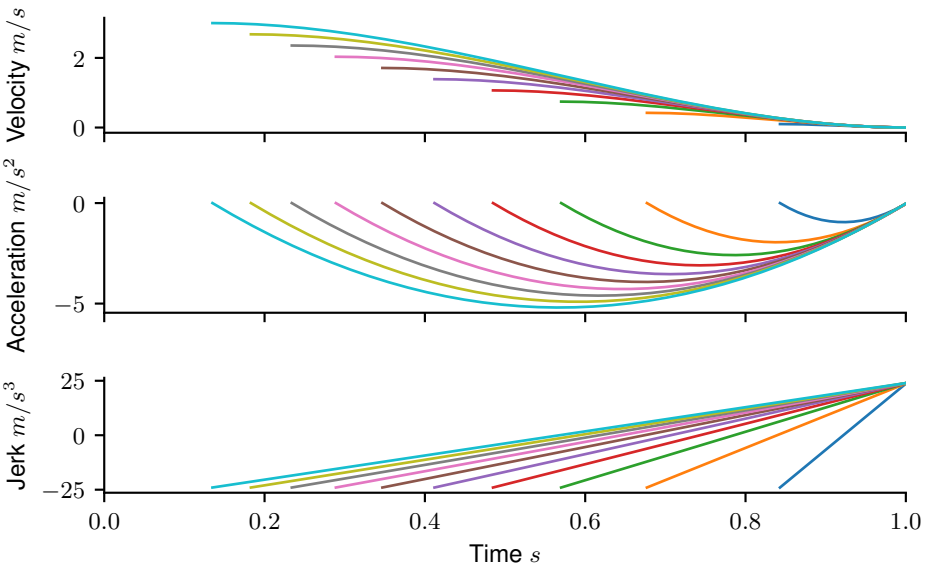

Figure 5.5: Minimum jerk velocity trajectories with converge time calculated based on planned end jerk, plotted for a set of linearly spaced initial velocities and zero acceleration in both the initial and final states.

maximum acceleration grows with the size of the change but still keeps a consistent end dynamic. This comes from the end jerk planning, which means that all trajectories will accelerate and decelerate in the same way regardless of how large the velocity change is. The acceleration envelope of a small velocity change will always be smaller than for a larger velocity change, helping with the predictability of the motion. The maximum velocity bounds the maximum acceleration, so the end jerk must be tuned so that acceleration from negative maximum to positive maximum does not saturate the system actuators.

5.2.3 Online Minimum Jerk Velocity Trajectory

The previous sections laid the groundwork for an online minimum jerk trajectory planning approach for reference generation. Assembling these pieces into one functional unit requires some assumptions and constraints

- Continuous velocity and acceleration
- The target velocity is piecewise constant and random
- Assume that the controller always can follow the the reference state

These assumptions make it possible to efficiently update the reference state at every time step. Continuous velocity and acceleration mean that the planned trajectory always begins at the current reference state. The target is assumed to be piece-wise constant. Therefore, it is possible to always define the end acceleration state as zero. The transition time can be found using the above-presented method of time estimation. The trajectory is then calculated using the estimated time, the current reference state, and the constant target velocity. The next step's reference state is then found by evaluating the polynomial at one time step dt ahead in time. The edge case of dt being larger than the estimated time is handled by checking for this case and setting the reference state equal to the wanted state.

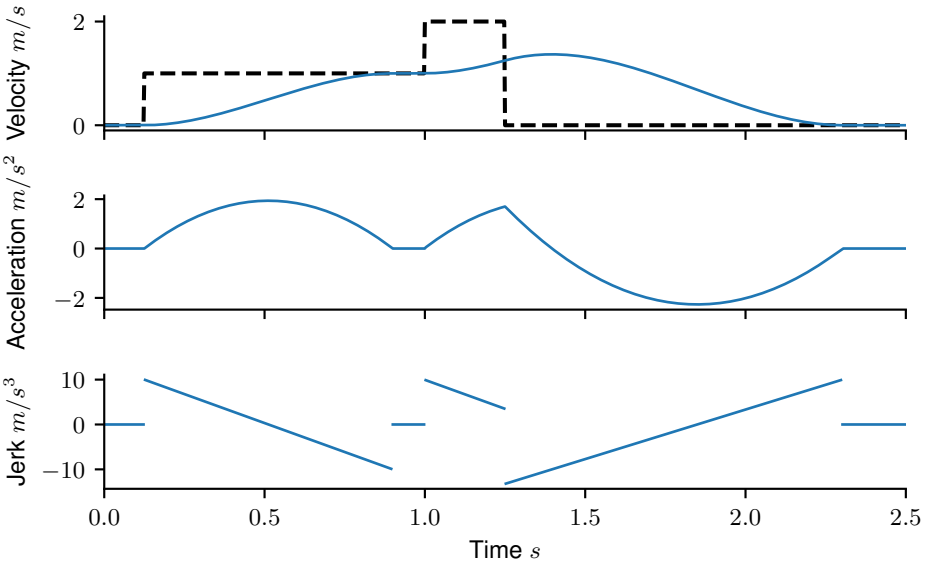

Figure 5.6: Online Velocity Minimum Jerk Trajectory for discontinuous user input.

The result is shown in Figure 5.6 for which the reference state has been computed with dt = 0.01 s and the planned end jerk being 10 $\mathrm{m\,s^{-3}}$.

5.2.4 Position Reference

It is beneficial for the control feedback to have a reference position to counter any slow biases that otherwise would build up a large drift over time. An example of this is keeping a constant depth for the Blueye Pioneer and its small positive buoyancy. This would cause the drone to slowly float up if it did not have a position reference to track using the depth measurements.

The solution is straightforward, initialize the position reference as the current depth when starting the controller and then integrate the minimum jerk velocity for each update call. Integration is done on the polynomial and becomes

$$x(t_s) = x(t_1) + \int_{t_1}^{t_s} \dot{x}(t)\, dt, \qquad t_1 \leq t_s \leq t_2 \tag{5.14}$$

$$x(t_s) = x(t_1) + \int_{t_1}^{t_s} c_1 + c_2 t + c_3 t^2 + c_4 t^3\, dt \tag{5.15}$$

t_1 is the current time that can be picked as 0 for simplicity, which gives us the following solution

$$x(t_s) = x(0) + c_1 t_s + c_2 \frac{t_s^2}{2} + c_3 \frac{t_s^3}{3} + c_4 \frac{t_s^4}{4}\,. \tag{5.16}$$

Figure 5.7 is an example of how a rectangular velocity input is followed by the minimum jerk velocity and is integrated into a position reference. Note that the velocity trajectory only is recalculated at the defined nodes but that the reference state is defined at any point in time on the piece-wise trajectories. This addition to the reference means that the controller has three smooth reference states to work with, acceleration, velocity, and position.

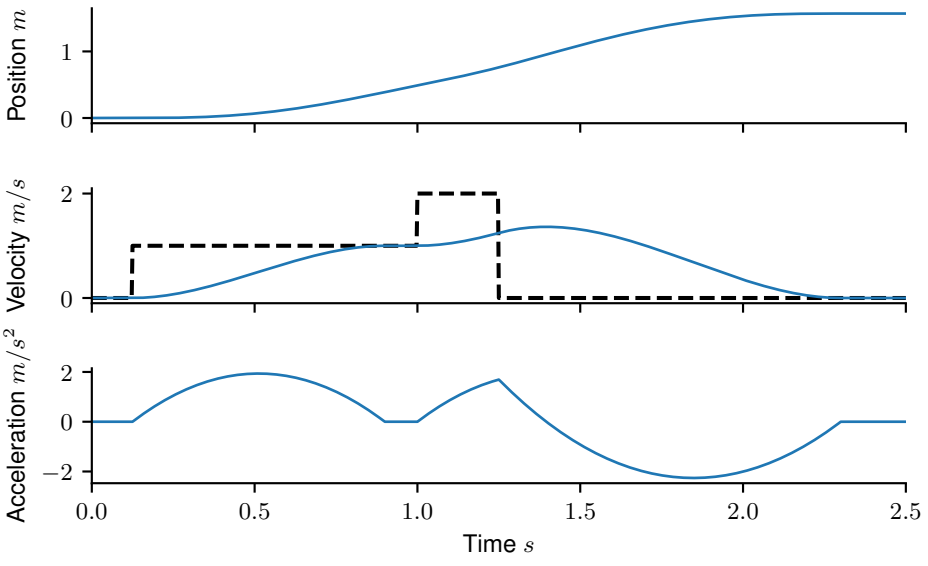

Figure 5.7: Position reference integrated from a piece-wise planned minimum jerk velocity.

5.2.5 Clamped Acceleration

The solution for minimum jerk velocity reference has so far assumed unbounded acceleration and that the acceleration can be achieved. This is not always the case. It is, in fact, common for the acceleration of the system to be a limiting factor. The Blueye Pioneer is not acceleration-limited by its thrusters but rather by the human response time, so this is not a problem that had to be solved for the product, but there is an easy solution if needed.

Recall that the trajectory planning is done online and on every control loop update. This means that any constraints only have to be evaluated until the next update and recalculation of trajectories. The acceleration reference can thus be clamped to the system's limits for each trajectory update. The velocity change will be a linear ramp until the next trajectory update.

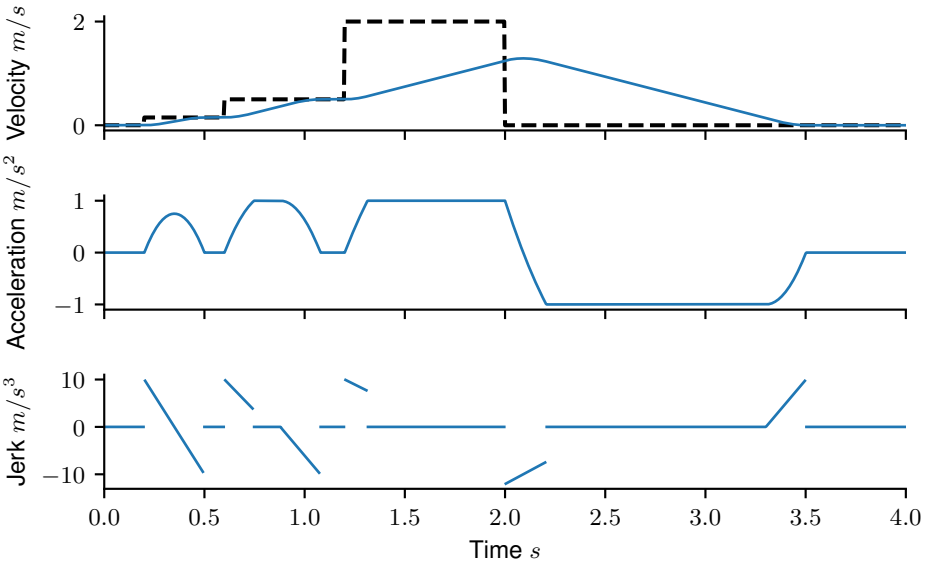

Figure 5.8: Minimum jerk velocity trajectories with clamped acceleration.

This method is shown in Figure 5.8, the acceleration limit is obeyed, and the velocity targets are reached. There is one problem, the acceleration ramps are not symmetric for growing and fading. The behavior or shape is different and thus not fully repeatable and predictable. The solution is to plan the clamping of the acceleration, note how the fading phase always begins with zero jerk and goes to the planned end jerk. Adding a planning target of zero jerk with the same snap, i.e., change of jerk, as for the fade phase yields the behavior as shown in Figure 5.9.

The acceleration profile is now symmetric and thus more predictable when tuning the controller later on.

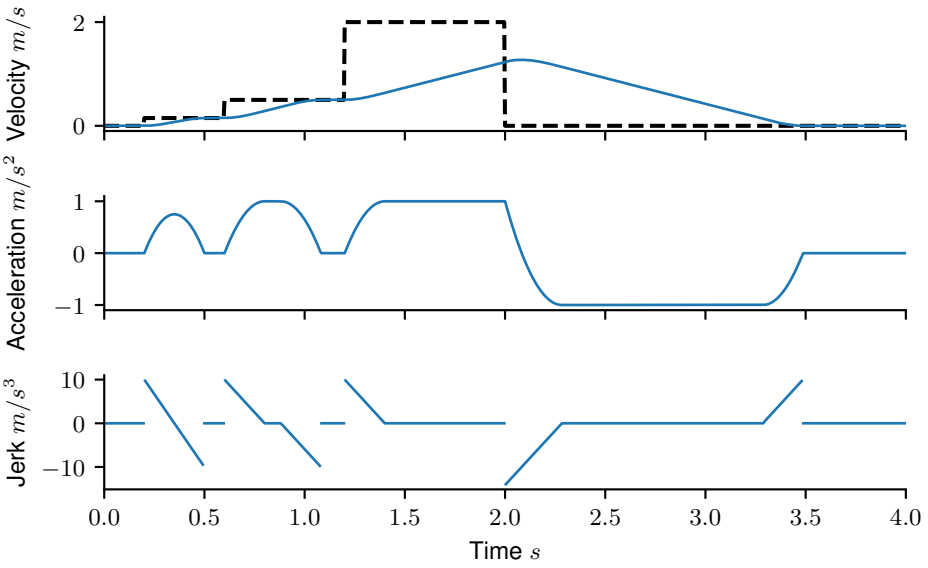

Figure 5.9: Minimum jerk velocity trajectories with planned clamping of the acceleration.

The planned acceleration clamping is found by observing that the acceleration trajectory is quadratic with its extrema at the end. Define the trajectory end as $t_2 = 0$ and solve for the remaining time.

$$t_1 = \sqrt{\frac{a_{max} - a_1 \operatorname{sgn}(v_2 - v_1)}{s}} \tag{5.17}$$

$$a_t = (a_{max} - st^2) \operatorname{sgn}(v_2 - v_1) \tag{5.18}$$

$$t_1 \leq t \leq t_2 = 0 \tag{5.19}$$

The snap, s, should be the same as for the tangent acceleration trajectory and can thus be calculated by

$$s = \frac{j_{end}^2}{4a_{max}} \,. \tag{5.20}$$

The reference trajectory uses the clamped trajectory towards maximum acceleration if the planned acceleration from the unconstrained trajectory is larger than the clamped trajectory. The logic for each planning step is thus to plan both alternatives and then pick the one that plans for the lowest acceleration in the direction of the planned velocity change.

5.3 Observers

An observer is needed for a full state estimation from the available sensors. The needed states consist of both position and velocity for each controlled dimension. The control scope for this work was to stabilize the depth and heading of the drone. The two dimensions are orthogonal to each other and can be seen as independent. This means that the observer can be broken up into two separate and independent parts, one for depth and one for heading.

5.3.1 Depth Observer

The depth dynamics of the drone are assumed to be independent of all other drone movements and can thus be modeled as a one-dimensional kinematic system. A kinematic only observer will be robust against non-linear hydrodynamics, tether drag, and collisions and was therefore preferred over including a dynamics model in the observer. This reduces the problem to a one-dimensional linear system that can be optimally solved with a Kalman filter. Let x be a column vector with the estimated state for position and velocity, and the Kalman filter becomes the following:

$$x_{k+1} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} x_k \tag{5.21}$$

$$z_k = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k \tag{5.22}$$

$$R_2 = 0.05 \tag{5.23}$$

The process noise, R_1, was tuned manually for a good trade-off between low noise in the velocity estimate and fast response for lower latency in the control loop. The pressure sensor was sampled at its maximum sampling frequency to improve the velocity estimates. The drone control system and observer ran at an update rate of $100Hz$, and the pressure sensor readings were sampled and held whenever the sensor could not match that frequency.

5.3.2 Heading Observer

The Blueye Pioneer is equipped with a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetic flux sensor. The naive approach for heading estimation

assumes that the drone always has a stable attitude and that there is no bias or drift in the gyroscope readings. A heading estimate could, in that case, be made by simply integrating the body frame gyroscope heading angular velocity over time. Body frame integration is not good enough for two reasons. The drone does not always maintain an upright attitude due to its small size, and the angular velocity measurements contain a bias that will build up over time.

The first problem is solved by estimating the full rotational state in world frame. The body frame angular velocity observations are used as a predictive step that can be corrected by comparing the acceleration vector against the earth's gravity. It is common in the ROV space to do this estimation in Euler degrees due to them being easy to understand and directly usable in higher-level control loops [7], and [4]. Euler angles are easy for humans to understand. However, they have multiple problematic properties, such as being highly non-linear and having states of undefined behavior, so-called gimbal lock. The attitude estimation space was instead chosen to be defined as a quaternion, with the observer proposed by Madgwick [19]. This observer uses a quaternion for its state definition, which can be robustly rotated by the angular velocity measurements and corrected using the gravity measurement to define the up axis.

The second problem of correct heading can be solved by comparing the expected magnetic field vector rotated to the sensor frame with the measured magnetic flux. A corrective step can then be made such that the estimated heading converges to be correctly defined relative to the true north. This method works well as long as no magnetic disturbances are present. Environmental magnetic disturbances from underwater metallic constructions and ships are unfortunately common and will sometimes significantly impact the estimate. The impact of this error is marginal for the navigational task where the heading in the UI will be wrong, but the pilot can rely on the video feed instead. However, the impact on control is more severe where the error can cause the drone to violently turn sideways even though the user wanted to travel straight forward. The solution was to decouple the heading estimation for the control loop and user interaction. The control loop can accept some absolute drift as this can be corrected by the human operator, while uncontrolled turning due to magnetic disturbances is unacceptable. Absolute drift means that the drone will very slowly turn right or left if no stick input is given for long periods. It was possible to reduce the drift of the estimate to less than one degree per minute with automatic calibration routines, even after long dives. That

drift is slow enough that a human pilot does not even recognize it as an error of the system and compensates for it without thinking.

The separate observer established for navigation with a geographical heading was estimated using the magnetic flux readings from the IMU. The pilot could then independently decide if the compass estimate is trustworthy and adapt depending on the situation. This means that the heading observation problems are split into two problems, one for navigation and one for control. The navigation observer uses the magnetic flux data, which usually provides a true north but is sensitive to magnetic disturbances. The control observer only uses acceleration and angular velocity measurements for its estimate and is thus robust against anything magnetic but with the downside of not knowing where the true north is and drifting over time. The quaternion representation is unfortunately hard for humans to grasp and the estimate was thus first converted to the understandable Euler angles representation [2] before being reported to the user.

$$\hat{x}_{\texttt{navigation}} = \begin{bmatrix} \texttt{pitch} \\ \texttt{roll} \\ \texttt{yaw} \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0 q_1 + q_2 q_3),\ 1 - 2(q_1^2 + q_2^2)) \\ \text{asin}(2(q_0 q_2 - q_3 q_1)) \\ \text{atan2}(2(q_0 q_3 + q_1 q_2),\ 1 - 2(q_2^2 + q_3^2)) \end{bmatrix} \tag{5.24}$$

The final puzzle piece for the heading observer is to provide an angular velocity for the dampening term of the heading control loop. This state is directly measured by the IMU sensor with low noise in body frame and could be transformed into earth frame with the already estimated attitude of the drone. However, this was not done. Instead, the heading angular velocity was kept in the body frame as the actuator for heading control is in the body frame, and that is the frame where dampening of the drone's heading is applied. Therefore, the angular velocity is simply the measured angular velocity from the gyroscope with a bit of exponential low pass filtering for lower noise. While a quaternion state can be used for full state feedback [12] this is not needed for heading only stabilization. A earth frame heading estimate is instead extracted from the quaternion state using the same Euler angles-based method for navigation. The heading angular velocity estimate is used as-is.

$$\hat{x}_{\texttt{control}} = \begin{bmatrix} \texttt{yaw}_{\texttt{earth}} \\ \dot{\texttt{yaw}}_{\texttt{body}} \end{bmatrix} \tag{5.25}$$

5.4 Controller

The drone is assumed to have independent degrees of freedom, which reduces the control problem of this thesis to two separate one-dimensional problems. These problems can be seen as linear except for the quadratic drag produced by moving through water. The maximum possible force produced by the thrusters is also so high that saturation is uncommon. The dynamics are also heavily damped due to being performed in water. However, this thesis is not about exploring advanced controller structures but focuses on the importance of high-quality and smooth reference states. Both the depth and heading controllers are designed and implemented as linear controllers because of these reasons. Recall that the control input u can be stated on vector form with the full state reference as follows

$$u = L_r r - L\hat{x} \tag{5.26}$$

$$= [L_r^P, L_r^D, L_r^A] \begin{bmatrix} r \\ \dot{r} \\ \ddot{r} \end{bmatrix} + [L^P, L^D] \begin{bmatrix} \hat{x} \\ \dot{\hat{x}} \end{bmatrix} \tag{5.27}$$

This is the basis for both the depth and heading controllers, and on which they then can build on top with needed edge case and non-linearity handling.

There is one detail in this setup that is not commonly used when designing a control system, and that is the acceleration reference, $\ddot{r}$. An acceleration reference is usually not defined for the control problem as it is either Dirac pulses for the cases of naive integration or a tri-state, bang-bang-like negative max, zero, or positive max for a linear ramp. None of those cases are constructive to include as reference states to the control problem and will probably do more harm than good. The smooth and continuous acceleration curve of the minimum jerk velocity trajectories is, on the other hand, feasible and possible to achieve. Incorporating the smooth acceleration as part of the reference allows tuning the response and minimizing the tracking error of a reference change.

Linear Quadratic Control for gain calculation was explored, but the amount of required testing and tuning was similar, and the problem is small enough to tune to wanted properties manually. The system is also relatively stable due to the damping provided by the water, so missteps in the tuning process had no real consequences.

5.4.1 Depth Controller

The depth controller's task on the drone is to keep a stable and constant depth, especially with a static reference. The primary disturbance on the drone for this control axis is the slightly positive buoyancy of the drone. Buoyancy is constant and easy to compensate for by introducing a slow integrator for the position error term.

$$e_k = \sum_{i=0}^{k} (r_i - \hat{x}_i)\, dt \tag{5.28}$$

An important note on this integrator term is that it is specifically introduced for counteracting the buoyancy and not any dynamics when moving. The integration is therefore paused during requested motion. The correct value is thus kept for when the drone stops again. The error term can then be added to the linear controller and the total is used as control input.

$$u_k = L_r r_k - L\hat{x}_k + L_I e_k \tag{5.29}$$

The introduction of an integrator introduces the risk of integrator windup. The typical case for the underwater drone is that the pilot runs it down into the seafloor but keeps giving the input of diving deeper. In this case, the position reference will keep going down while the drone is stuck on the seafloor. This will cause very high control inputs, and the integrator can quickly go towards some large unrealistic value. The unrealistic integrator state is especially problematic as it can take a long time to get rid of and make the drone unusable for that time as it will try to run into the bottom again and again. Three guards were put in place against this problem. First, a constraint on the maximum allowed position error clamps the position reference to a realistic set of values. Secondly, clamping of the integral does not allow it to grow bigger than some number, which is lower than a fully saturated thruster. Finally, the introduction of an anti-wind-up function.

The anti-wind-up function works by doing a sanity check on the position error integral e_k compared to the current position error $\tilde{p}_k$. It is only seen as a problem if the absolute position error is larger than some threshold. The error integral e_k is, in that case, reduced towards zero by iteratively multiplying with $D \in (0, 1)$.

$$\tilde{p}_k = r_k - \hat{x}_k \tag{5.30}$$

$$e_{k+1} = \begin{cases} De_k, & \text{if } e_k\tilde{p}_k < 0 \text{ and } |\tilde{p}_k| > threshold \\ e_k + \tilde{p}_{k+1}dt, & \text{otherwise} \end{cases} \tag{5.31}$$

The test checks that the integral error component works in the same direction as the position error. They will have different signs if that is not the case, and it is easy to check if the signs are the same by multiplying and a check on the sign of the result. Adding a threshold check for the error ensures that the controller behaves like a linear controller during normal operation.

5.4.2 Heading Controller

The heading controller works quite well on the above default setup apart from one problem due to thruster deadband. The deadband becomes a problem when the drone holds a static heading and only experiences small perturbations. The observed effect is that the drones start to turn slightly because the calculated input signal is lower than the smallest achievable thrust. The drone will then suddenly jerk back once the error is large enough for the thrusters to run due to the thrusters being very aggressive at startup. Reducing the feedback gains reduced the problem but also reduced the drone's ability to accurately follow a moving reference as well as counteracting disturbances. The controller gain was, because of this, divided into two different setups, one for static hold and one for moving the drone. This avoids the negative impact of tuning for reduced deadband problems while retaining the quick response when moving.

5.4.3 Accelereation Feedback

The reference acceleration state is used as a feedforward argument in the proposed implementation. It is possible to estimate the acceleration states for the controlled dimension, and it is thus tempting to introduce feedback for the acceleration state. This was evaluated and decided against for two main reasons. The first is noise. The acceleration estimate is noisier as it is a derivative of a random signal and noisy control signals produce vibrations and higher battery consumption. Filtering of the acceleration state is possible but introduces a time delay that reduces the effectiveness of the feedback. The depth acceleration estimate could have been improved dramatically by including the accelerometer in the estimate. The large bias of the sensor made that non-trivial, and it was thus left outside the scope

of this thesis. The second reason is that the deadband on the thrusters used on the drone caused non-smooth motion, which would be enlarged into oscillation with acceleration feedback. This was especially true for the heading estimate. To summarize, it is possible to introduce feedback on acceleration but not on the target platform with sensors and actuators.

Chapter 6

Results

This section presents both simulations and real-world tests for using the reference ramping method developed in this thesis. Two cases are compared, direct usage of the user velocity as the set point which is compared against the usage of Online Minimum Jerk Velocity Trajectories for reference generation. Simulations are first presented as they are easier to design for and reproduce accurately to show specific parts of the dynamics. Subsequently, real-world tests are then presented to show that the simulation results stand in the real world and that the method works exceptionally well on Blueye Pioneer.

6.1 Simulations

The goal of the simulations is to show the advantage of using high-quality ramping as a part of the feedback loop. Reference ramping is a commonly overlooked part of the feedback loop in both university courses and scientific papers. The following steps have been taken to emphasize the effect of the reference method.

- Assume a known linear system model.
- Use the same controller and controller tuning.
- Assume a perfectly known state.
- Reuse the same process noise.

Reusing the same process noise for every method in the simulations clarifies what affects and does not affect the controller. The results will be identical when

the reference is equal. This is clear at the beginning and end of these simulations, as these sections are chosen to be a simple static hold. A significant contributor to the performance of a feedback system is the quality of the state estimates employed on the drone. The state estimates quality depends on the quality of the sensor readings and the ability of the observer to distinguish signal from noise. However, the goal of these simulations is not to provide an accurate representation of the Blueye underwater drone but rather to clearly show the effects of reference ramping. The observer and any simulation of measurement noise are therefore omitted, and the complete state is assumed to be perfectly known. The same logic is used for the choice of identical controllers for the shown cases. Keeping the controllers identical makes it clearer to distinguish between the effects of the reference and the controller. Therefore, the simulations will show identical results for the parts where the reference is identical. The dynamics are finally assumed to be linear and known such that a suitable controller easily can be made, especially the feedforward part can be precisely calculated.

6.1.1 Velocity Step

Figure 6.1 shows the most straightforward simulation for showing the difference between no input ramping and minimum jerk based ramping. The user input is a target velocity, and it starts at zero, meaning no motion, then takes a step to $0.3m/s$ keeps that level for $3s$, and returns to zero meters per second after that. The user input is the red dashed line in the velocity graph.

The naive reference method uses the user input directly and integrates it to acquire a suitable position reference for each discrete time step. The naive position reference is also a red dashed line. The green line represents the result for this simulation, and it fulfills the velocity target quickly but overshoots both the velocity and position reference.

The minimum jerk trajectory-based reference is planned as a smooth trajectory to the user requested velocity and then back to zero when the user reference changes. The position reference is integrated over time, and both are marked as dashed orange lines.

The naive solution has a faster rise time at the expense of more overshoot. The minimum jerk is slower than the naive method, but this is due to the planned rise time, and the system follows the reference almost perfectly. See 6.1.3 for an elaborated look into trajectory rise time. The big difference in rising time is evident

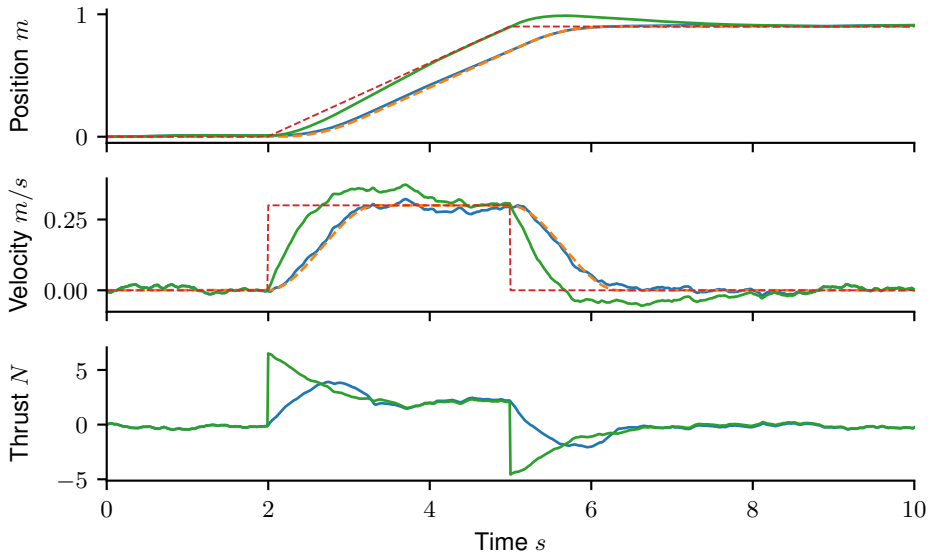

Figure 6.1: Rectangular step velocity target, with (blue and dashed orange) and without (green and dashed red) reference ramping.

in the thrust subplot, where the naive reference results in a sizeable discontinuous step in thrust for a significant initial acceleration. This causes a high peak thrust as well as a jerky behavior. Compare that with the minimum jerk approach, where the peak is lower and comes much later after a smooth increase in thrust. Both trajectories end up at the same position despite the minimum jerk trajectory reference lag.

	Direct Velocity	Minimum Jerk Trajectory
Peak Thrust	8.91	4.88
Position Error	2.16	0.14
Thrust	6554	3669
Jerk	1643461	11728

Table 6.1: The peak input, i.e., the maximum force used. And the square sum of; position error, control inputs, and jerk for the transition. Lower is better for all metrics.

Table 6.1 presents some key metrics from the two simulations, and there are significant reductions on all metrics when using the smoother reference. The peak thrust is reduced by a factor of two, which means that a weaker system with lower peak output could fulfill the trajectory without clipping.

The square sum of the position error is an entire 15 times lower, which can be attributed to the large overshoots by the naive method. The minimum jerk trajec-

tory based case has in comparison no overshoot caused by the transition, and the position error is almost entirely due to the presence of process noise.

The significant reduction in the square sum of thrust might be the most important result of this simulation. This metric directly impacts the drone's power consumption and thus its battery life. A reduction by two for small movements like shown will significantly increase battery life for detailed observation and documentation missions.

The last metric, the square sum of jerk, is an incredible 140 times larger for the naive direct usage of the input velocity compared to the smoother minimum jerk trajectory. This is entirely due to the discontinuous thrust jumps when the input reference changes.

6.1.2 Acceleration Feedforward

Figure 6.2 shows the effect of different gains for the acceleration feedforward component. The coloring of the graph is related to the amount of acceleration feedforward gain, with blue being zero and yellow double the correct amount. The gain is in steps of 20% points. The perfectly tuned case is black, whereas the generated reference is the red dashed line. All cases are simulated with the same process noise. The only thing that is changed between these 11 simulations is thus the acceleration feedforward gain. This should be clear when viewing the simulations' beginning and end, where all cases are identical. Looking at the effect of the feedforward term tells us several things. First off, a perfectly tuned feedforward yields a resulting trajectory that is very close to the wanted plan, with no overshoot in either position or velocity. The lack of overshoot is especially evident in the thrust graph, where it shows as a clear transition to equal the drag of the planned velocity. Compared with the no-feedforward case, the blue line lags in reaction and then overshoots the position reference for a long time before converging to the reference. The over-tuned case with twice the correct gain has the opposite behavior. A too-aggressive reaction initially goes too fast and then brakes too much, causing that case not to reach the final position reference as planned. Note how all simulated cases which are linearly spaced in gain also are linearly spaced in their results. This proved to be very helpful when tuning the real system; any change in gain would have a proportional change in the results.

The vital part of this test is a more general one, and that is that all the tested feedforward gains were stable and converged to the reference. Hence there exists

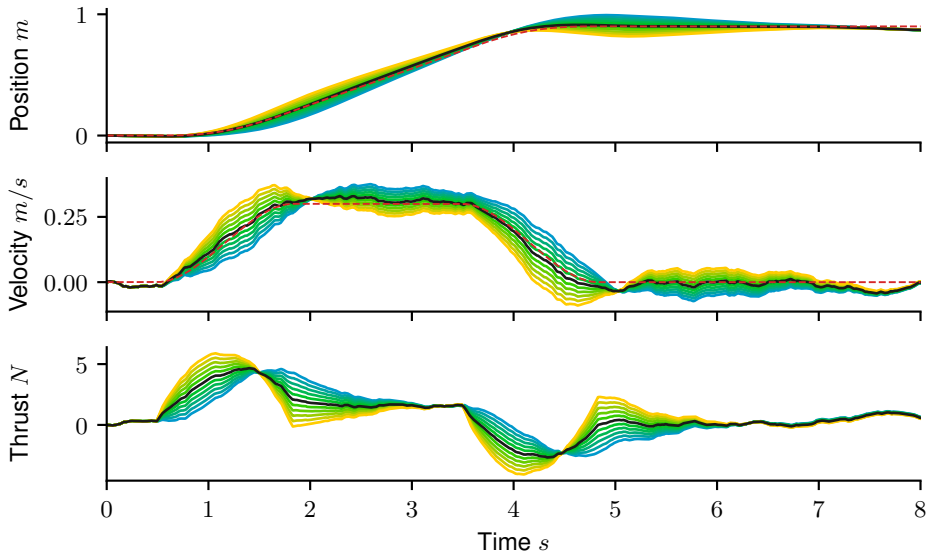

Figure 6.2: Impact by acceleration feedforward, 0%, blue to 200%, orange in steps of 20%. The black line represent a perfect tuning of acceleration feedforward.

little risk in slowly introducing some of this gain. Another observation should be done about the magnitude of the gain, 80% or even as little as 60% of the correct gain is still a significant help in getting a better reaction from the controlled system. It might even be of an advantage to slightly under-tune this feedforward component. The minimum square sum of thrust for this particular case was the simulation with 75% of the correct gain.

Another takeaway of this simulation is that it is easy to tune the acceleration feedforward manually. The tuning should begin with zero acceleration feedforward, which means the blue line in Figure 6.2. The controller should first be tuned for the stable static and constant velocity cases. The feedforward can then be added once those two cases are satisfactorily tuned. The most straightforward approach is to increase the acceleration feedforward until the trajectory overshoots the velocity during the acceleration phase and then reduce it until the result is slightly behind the reference.

6.1.3 Rise Time

A central aspect of feedback design is the metric of rise time, overshoot, and settling time. They are related to the controller's ability to follow a quick change

in the reference. Rise time has multiple definitions depending on application, the defintion used here is the time required for the controlled system to go from 0% to 90% of the target state. Figure 6.3 shows a step input in user reference speed from 0 to 0.3 $\mathrm{m\,s^{-1}}$ and the result when using the same controller as in Figure 6.1 and with no process noise. The difference between the simulations stems only from how the reference is handled. The dark green line that stands out in shape from the rest results from naive handling of the reference and direct usage of the user input. The other simulations use the minimum jerk trajectory reference ramping presented in this thesis. The ramping cases differ by the planned maximum jerk which range from 0.5 $\mathrm{m\,s^{-3}}$, light green to 10 $\mathrm{m\,s^{-3}}$, orange in steps of 0.5 $\mathrm{m\,s^{-3}}$. The blue line marks the trajectory with equal rise time to naive implementation, and it uses a maximum planning jerk of 3 $\mathrm{m\,s^{-3}}$.

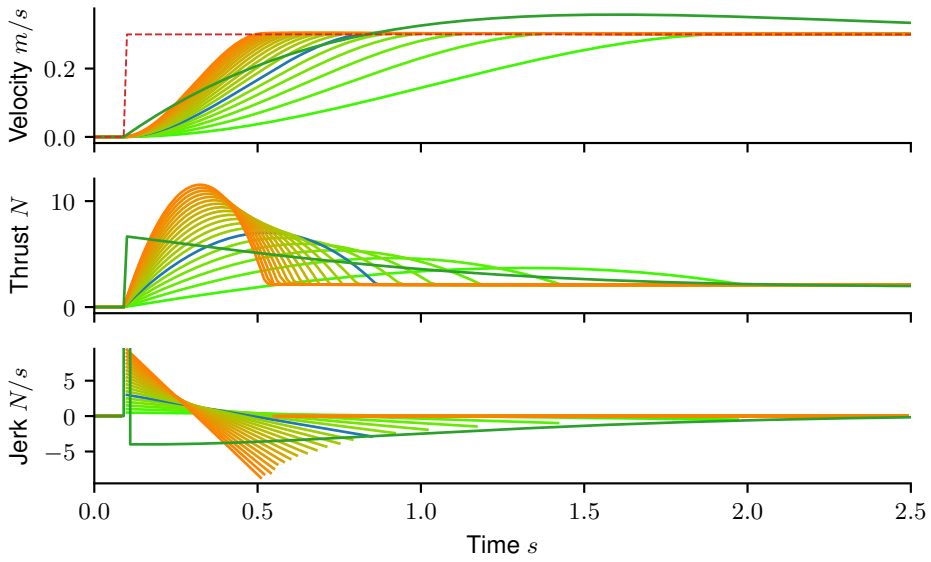

Figure 6.3: Changing the rise time by changing the planned maximum jerk. Light green, low maximum jerk, orange high maximum jerk. Dark green trajectory is the system response with no velocity ramping. Blue is the minimum jerk trajectory with equal rise time to the naive trajectory.

Remember that the controller feedback stays the same for all simulations and that the only difference is in the reference state. It is thus possible to define metrics such as rise time as part of the reference generation instead of the feedback loop when using acceleration feedforward. Also, pay attention to the lack of overshoot. This happens because the position state of the reference also follows a realistic

trajectory which considers limited jerk. The naive solution assumes that the controller can instantly push the velocity state to the reference velocity and lets the position reference go full speed ahead. The problem is that the true state lags behind the reference since there is a limit to how sensitive a controller can be made. This lag must be caught up to later, so there is overshoot in the naive response. The minimum jerk trajectory ramped reference with acceleration feedforward does not lag. The reference considers the jerk limited acceleration and uses the known acceleration as a feedforward component.

Mark a few interesting observations. Firstly, the minimum jerk trajectory reference can achieve the same rise time with the same peak input thrust as the naive reference but with complete removal of problems with overshoot. The rise time can be shorter when using acceleration feedforward, assuming the input signal is achievable and the acceleration can be properly modeled and calculated. The peak input will go up considerably with a shorter rise time, and so does the difficulty in modeling the needed acceleration correctly.

6.1.4 Pseudo Random Velocity Target

The last simulation is done on pseudo-random mock data to show the result better when using a human input. The user target input signal was generated by logging the actual human input. This data could then be as the wanted velocity in the simulation. There are both smooth parts typical of experienced pilots and rapid full forward and breaking typical for novice drivers.

Figure 6.4 shows the different results when using a naive non ramped reference (dashed red) and a minimum jerk trajectory based reference (dashed orange). Table 6.2 presents some metrics for the two simulations with data on peak input, the square sum of the input thrust, and the square sum of the jerk in the control signal. The most significant difference is found in the jerk metric, where there is a 40-fold decrease by introducing the minimum jerk trajectory based velocity ramping. This is caused by the step-like user input, which sometimes changes rapidly and causes a very high jerk in the non-ramped system. The position error of the trajectory is reduced by an order of magnitude. When stopping, the almost complete overshoot removal may be the most significant addition to the improved sense of stability and control. The used thrust is halved when applying the smooth ramping, which will result in significant battery life improvements.

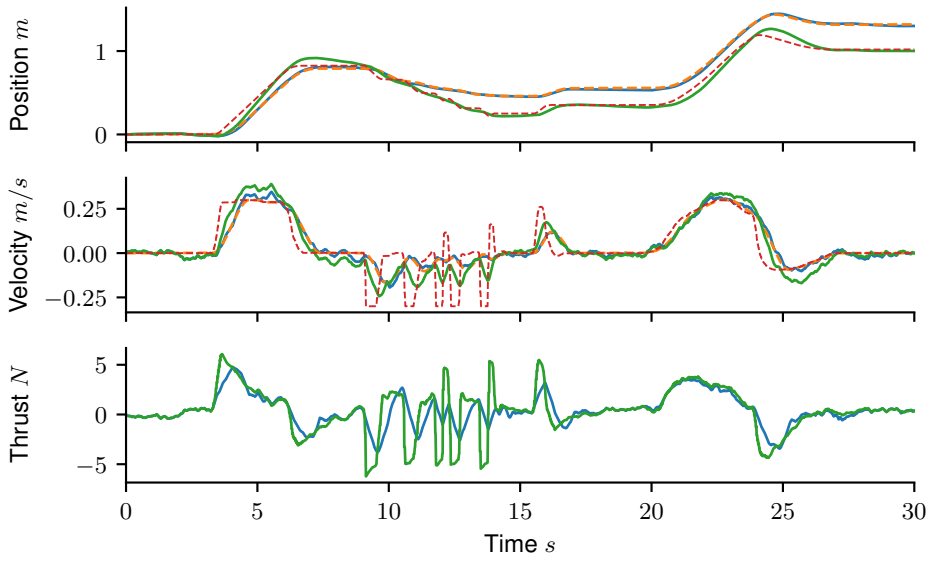

Figure 6.4: Typical pseudo-random non-linear human input. With a comparison of naive and minimum jerk trajectory references. The naive reference is marked as a red dashed line, and its result is marked in green. The minimum jerk reference is the orange dashed line with the result marked in blue.

	Direct Velocity	Minimum Jerk Velocity
Peak Thrust	6.24	4.41
Position Error	5.57	0.51
Thrust	14814	6987
Jerk	2389915	62596

Table 6.2: Performance metrics for pseudo-random input. Peak input, i.e., the maximum force used. The square sum of; position error, control inputs, and jerk for the transition. Lower is better for all metrics.

6.2 Experiments

Experiments were continuously performed on the Blueye Pioneer platform during the work with this thesis. Bugs and understanding of how the system reacts to the different approaches were iteratively solved and incorporated into the design. Below is a small comparison between the original direct integration method used on the drone and the new minimum jerk trajectory based reference. The implementation effort was focused on releasing a stable and well-tested code to customers. The setup was not optimized for quantitative comparisons between the methods, so no comparable numbers are available. The following tests were done with the same

controller gains and only the reference generation method where changed between the tests.

6.2.1 Auto Depth

Figure 6.5 shows the resulting trajectory for a step input from the user using the original direct velocity method. It shows similar characteristics to the simulations with a rapid change in thrust at the beginning of the transition. The velocity then lags behind the reference causing a lag and an overshoot in the position. Figure 6.6 shows a similar depth transition but using the minimum jerk based reference. The most significant difference is in the amount of jerk in the system, which is lower due to the smoother change in thrust caused by the smoother reference. The overshoot in position is also smaller.

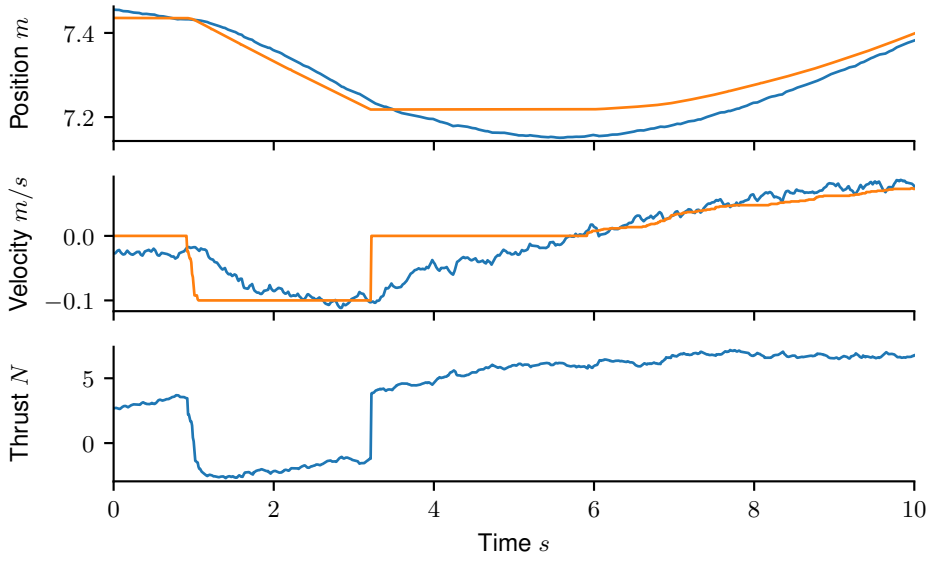

Figure 6.5: Step input for depth using direct velocity reference, orange line. The estimated depth, velocity, and force are in blue.

Some qualitative observations of the results are based on the reported feeling of drone usage. The system's most noticeable impact of the reduced jerk was visible in the covariant dynamics of pitch, cruise velocity, and depth acceleration. Changing the depth when driving the drone forward will cause a pitching motion due to the asked velocity and the acceleration. The high jerk of the original implementation caused large oscillations in the pitching axis in these situations and made it

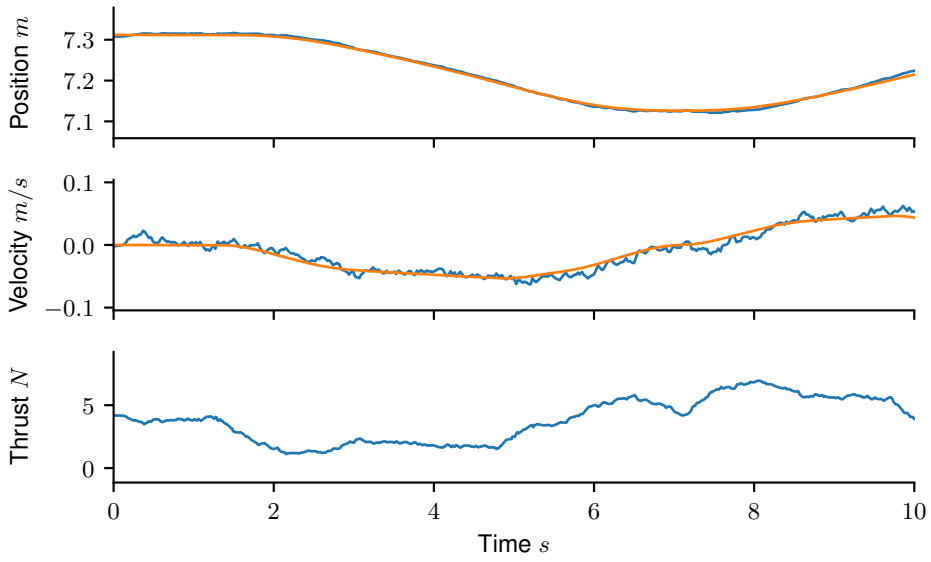

Figure 6.6: Step input for depth using a minimum jerk trajectory based reference shown in orange. Resulting position, velocity, and used thrust in blue.

hard to follow the seafloor. The reduced jerk of the online minimum jerk velocity reference significantly reduced the oscillations and made seafloor navigation and general tracking of objects under motion much more manageable. The reduced overshoot also significantly improved stability and control as the drone stops accurately every time.

6.2.2 Auto Heading

Figure 6.7 and Figure 6.8 show the result for driving with direct integration and minimum jerk trajectory reference respectively. The most important difference is the lower and less aggressive peak moment used by the minimum jerk method. Comparing the two trajectories also exposes one of the design problems of the drone. The thrusters exhibit a relatively large deadband which means that they can not produce small forces. This is generally a smaller problem for the depth control as it is biased from the small buoyancy of the drone. The heading is close to unbiased, though, and the uneven engagement of the thrusters causes effects as present at about one second in Figure 6.8. The proper response is shown at three and nine seconds in the same figure.

Again, the qualitative results based on usage and feedback from colleges and customers take up several advantages with the smoother approach. The reduced jerk at large input changes by the user produces a smoother motion, improving video quality and ease of use. The usable video length is extended as navigation changes are much smoother than the direct velocity method. The improved ease of use comes from the original high jerk motion sometimes being too fast and missed video targets when diving. The minimum jerk reference also stops faster and without overshoot while still being smooth in its motion. Early tests used jerk bounds on the control feedback signal instead of the reference. This did improve the situation by mitigating the high initial jerk. The downside of this approach was that the control systems' ability to counteract disturbances was reduced, which was problematic when operating in conditions with currents and other disturbances. The minimum jerk trajectory reference method can sustain disturbance rejection while maintaining navigational smoothness. It was even possible to increase the feedback gains when using the improved reference and thus improve the controller's strength.

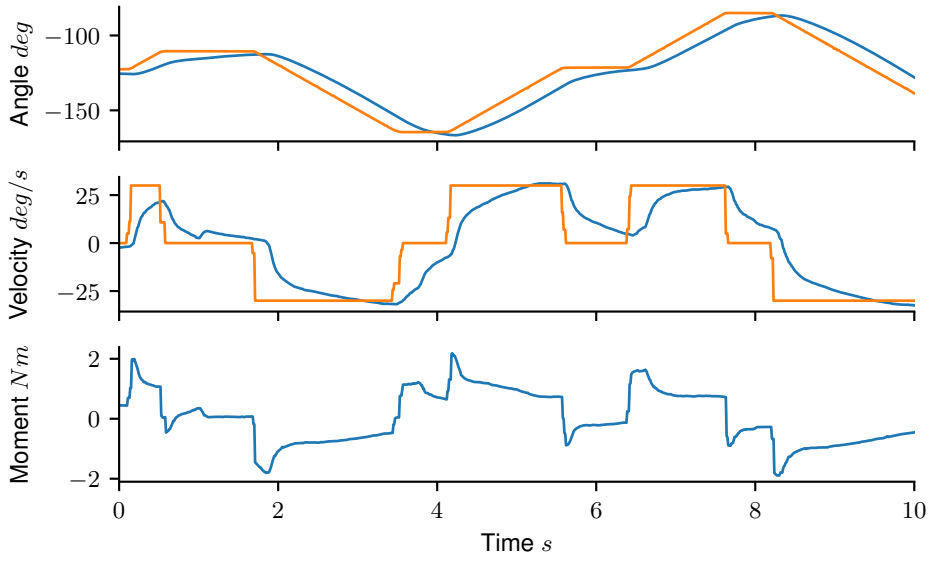

Figure 6.7: Step input for heading using direct integration of the reference, orange line. The estimated heading, angular velocity, and moment are in blue.

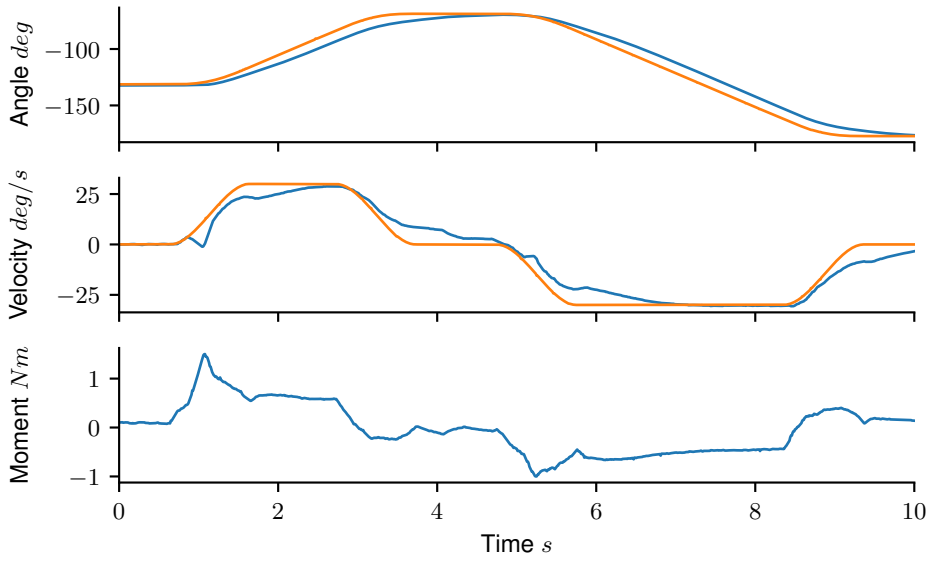

Figure 6.8: Step input for heading using a minimum jerk trajectory based reference shown in orange. The resulting heading, angular velocity, and used moment are in blue.

6.2.3 Current Usage

The final result of developing and implementing Online Minimum Jerk Velocity Reference for the Blueye Pioneer is deployed on all operational drones. The smooth and easy control of the drone is stated as one of the product's most prominent strengths. Several customers have given feedback and expressed how impressed they were with the operation of the drone and the feel of control that was described as well balanced for its size. The reduced control peaks and jerk in the thrusters have increased the drone's battery time by about 10 to 20% to the original implementation. The same method is also used for controlling the pitch speed of the drone's camera tilt system, yielding smooth camera motions for the pitch axis.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The main goal of this thesis was to improve the perception of stability when remotely operating the Blueye Pioneer underwater drone. Two metrics were chosen for measuring perceived stability; motion smoothness and motion overshot. The proposed method both restricts beginners to possible motion trajectories and maintains high responsiveness for advanced users.

The presented work fills a gap in the literature about the handling of humans in the loop for underwater vehicles. Most work in this field covers either the actual control loop or planning algorithms for autonomous applications where humans are excluded from operation altogether. Humans are generally expected to adapt to the system if they are part of the control loop. This work takes a different approach and tries to take more of the load away from the human pilot, even under dynamic driving to make it easier to focus on the task at hand.

The final implementation is a smooth ramping of the velocity target given by the human operator before it is used as a reference in the control system. The ramping is performed using a recursive implementation of the minimum jerk trajectory for velocity transitions, here called online minimum jerk velocity trajectories. It runs in real-time and produces a smooth and continuous velocity and an acceleration reference state usable for the feedback controller. The acceleration reference is determined to be very helpful in reducing lag, rise time, and overshoot of the final system by integrating it as a feedforward term in the control loop. The reference position is easily integrated from the smooth reference velocity.

The general takeaway of this thesis is that it is essential to take higher-order derivatives such as acceleration and jerk into account when generating a reference state. Minimum-jerk-based reference ramping improves human-in-the-loop systems such as the Blueye Pioneer, even when the response time is fast in highly dynamic systems. The smoother reference also reduces peak control input, lowering battery consumption and system wear. It also practically eliminates overshoot which increases the feeling of system stability. Another benefit is decoupling the system's response to human input and external disturbances. The disturbance rejection can thus be aggressive while the human input response can be tuned to a desired smoothness.

7.2 Future Work

The original task of improving the control feeling of the Blueye Pioneer is considered achieved with the Online Minimum Jerk Velocity Reference. However, in writing this thesis, several new areas for future work have opened up.

7.2.1 Formalized Acceleration Feedforward

The addition of a feedforward component to the control system based on a continuous acceleration reference was successfully simulated, implemented, and tested. However, it only used manual tuning or direct calculation on simple linear systems. The next logical step in studying the use and effects of an acceleration reference is to provide a more generalized approach for including this acceleration reference. An interesting approach would be to formalize it in the context of Linear Quadratic Control to prove whether it enhances the optimal control problem for time-varying reference signals. The same framework could then be used to define the optimal feedforward gain.

7.2.2 Position as User Input

To use minimum jerk trajectory generation with a position as the user input was explored but not implemented or adequately evaluated. However, it was found that an online iterative solution similar to the velocity case exists with the remaining

converge time being given by

$$\pm j_{end}t^3 + 3a_e t^2 + 24v_e t + 60p_e = 0 \tag{7.1}$$

The correct root out of six possible solutions must be selected and then evaluated against possible constraints. The situation with constraints becomes tricky, and the efforts were therefore not included in this study. A possible alternative approach is to handle it similar to the eight segment minimum time problems but instead with three phases: acceleration, constant velocity, and deceleration. This would be an exciting topic to study further as it potentially could reduce the minimization problem compared to the minimum time problem. In addition, it could introduce a slightly different metric which might be advantageous in some situations.

7.2.3 Optimization

The method is lightweight enough for running on the Blueye Pioneer, but there should be plenty of room for optimization. The most obvious one is not recalculating the trajectory from scratch at every step. A straightforward approach is only to do it when needed due to user input changes. A more interesting approach would be to formulate it as an iterative filter or similar. Again not strictly needed for the target system but an interesting extension for better understanding and the possibility for deployment on units with less computational power.

7.2.4 Extend to MIMO

The current implementation only handles heading and depth as two decoupled Single Input Single Output (SISO) systems. The drone needs support for feedback control of the horizontal plane if position measurements are added in the future. The next step would be to handle 6 DOF navigation of underwater drones. How would a jerk-aware smoothing be applied in such Multiple Inputs Multiple Outputs (MIMO) systems?

7.2.5 Quadratic Drag Compensation

Linear dynamics well describe the Blueye Pioneer for low velocities. However, higher velocities introduce non-linearities such as quadratic drag. It is possible to compensate for these effects with an integral on the velocity error. There are

probably performance gains to be made by dealing with these dynamics directly within a non-linear controller.

Bibliography

[1] Pierre-Jean Barre, Richard Bearee, Pierre Borne, and Eric Dumetz. Influence of a jerk controlled movement law on the vibratory behaviour of high-dynamics systems. *Journal of Intelligent and Robotic Systems*, 2005.

[2] Jose-Luis Blanco. A tutorial on se(3) transformation parameterizations and on-manifold optimization. Technical report, University of Malaga, 2013.

[3] Tweet Boogie and HowCastArtsRec. How to pop — club dancing, 2011.

[4] Fredrik Dukan. *ROV Motion Control System*. PhD thesis, NTNU, 2014.

[5] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steve Worley. *Texturing and Modeling, Third Edition: A Procedural Approach*. Morgan Kaufmann, 2002.

[6] K. Erkorkmaz and Y. Altintas. High speed cnc system design. part i: jerk limited trajectory generation and quintic spline interpolation. *International Journal of Machine Tools and Manufacture*, 2011.

[7] T. I. Fossen. *Guidance, Navigation and Control of Marine Craft*. John, 2010.

[8] Mahdi Ghazaei, Anders Robertsson, and Rolf Johansson. Online minimum-jerk trajectory generation. *Proc. IMA Conf. Mathematics of Robotics*, 2015.

[9] T. Glad and L. Ljung. *Reglerteori Flervariabla och olinjära metoder*. Studentlitteratur, 1997.

[10] R. Haschke, E. Weitnauer, and H. Ritter. On-line planning of time-optimal, jerk-limited trajectories. *Inteligent Robots and Systems*, 2008.

[11] Eirik Hexeberg Henriksen. Rov control system for positioning of subsea modules. Master's thesis, NTNU, 2014.

[12] Hans-Christian Becker Jensen and Rafal Wisniewski. Quaternion feedback control for rigid-body spacecraft. Technical report, Aalborg Universitet, 2001.

[13] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[14] John Kessenich. *The OpenGL® Shading Language*. The Khronos Group inc, 1.4 edition, 2009.

[15] Kim Doang Nguyen, I-Ming Chen, and Teck-Chew Ng. Planning algorithms for s-curve trajectories. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pages 1–6, 2007.

[16] K. Kottenhoff and J. Sundström. Samband mellan körstil och åkkomfort – förbättringspotentialen inom kollektivtrafiken. Technical report, VTI, 2011.

[17] Jonathan Cacace Lentin Joseph. *Mastering ROS for Robotics Programming*. Packt Publishing Ltd, 2018.

[18] S. Macfarlane and E.A. Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on Robotics and Automation*, 2003.

[19] Sebastian O.H. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. Technical report, University of Bristol, 2010.

[20] F. Manning and CR. Millman. *Frankie Manning: Ambassador of Lindy Hop*. Temple University Press, 2007.

[21] MW. Mueller and R. D'Andrea. A model predictive controller for quadrocopter state interception. *European Control Conference*, 2013.

[22] MW. Mueller, R. D'Andrea, and M. Hehn. A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification. *International Conference on Intelligent Robots and Systems*, 2013.

[23] Flash T. Hogan N. The coordination of arm movements: An experimentally confirmed mathematical model. *The Journal of Neuroscience*, 5(7):1688–1703, 1985.

[24] A. Piazzi and A. Visioli. Global minimum-jerk trajectory planning of robot manipulators. *IEEE Transactions on Industrial Electronics*, 2000.

[25] RockThatSwing. Rtsf 2017 - lisa & fabien, 2017.

[26] O. Salvador and D. Angolini. *Embedded Linux Development with Yocto Project*. Packt Publishing Ltd, 2014.

[27] Peter M. Thompson. Snap, crackle, and pop. Technical report, Systems Technology Inc, 2011.

[28] Trinamic. Motion control ramping, 2017.

[29] Eric W. Weisstein. "smooth function." from mathworld–a wolfram web resource., 2020.