Safe Planning And Control Of Autonomous Systems: Robust Predictive Algorithms

Yash Vardhan Pant

University of Pennsylvania, pant.yash@gmail.com

Follow this and additional works at: https://repository.upenn.edu/edissertations

Part of the Electrical and Electronics Commons

Recommended Citation


https://repository.upenn.edu/edissertations/3419

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/edissertations/3419
For more information, please contact repository@pobox.upenn.edu.
Safe Planning And Control Of Autonomous Systems: Robust Predictive Algorithms

Abstract
Safe autonomous operation of dynamical systems has become one of the most important research problems. Algorithms for planning and control of such systems are now nding place on production vehicles, and are fast becoming ubiquitous on the roads and air-spaces. However most algorithms for such operations, that provide guarantees, either do not scale well or rely on over-simplifying abstractions that make them impractical for real world implementations. On the other hand, the algorithms that are computationally tractable and amenable to implementation generally lack any guarantees on their behavior.

In this work, we aim to bridge the gap between provable and scalable planning and control for dynamical systems. The research covered herein can be broadly categorized into: i) multi-agent planning with temporal logic specications, and ii) robust predictive control that takes into account the performance of the perception algorithms used to process information for control.

In the rst part, we focus on multi-robot systems with complicated mission requirements, and develop a planning algorithm that can take into account a) spatial, b) temporal and c) reactive mission requirements across multiple robots. The algorithm not only guarantees continuous time satisfaction of the mission requirements, but also that the generated trajectories can be followed by the robot.

The other part develops a robust, predictive control algorithm to control the the dynamical system to follow the trajectories generated by the rst part, within some desired bounds. This relies on a contract-based framework wherein the control algorithm controls the dynamical system as well as a resource/quality trade-o in a perception-based state estimation algorithm. We show that this predictive algorithm remains feasible with respect to constraints while following a desired trajectory, and also stabilizes the dynamical system under control.

Through simulations, as well as experiments on actual robotic systems, we show
that the planning method is computationally efficient as well as scales better than
other state-of-the-art algorithms that use similar formal specifications. We also show
that the robust control algorithm provides better control performance, and is also
computationally more efficient than similar algorithms that do not leverage the resource/
quality trade-off of the perception-based state estimator

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Electrical & Systems Engineering

First Advisor
Rahul Mangharam

Subject Categories
Electrical and Electronics

This dissertation is available at ScholarlyCommons: https://repository.upenn.edu/edissertations/3419
SAFE PLANNING AND CONTROL OF AUTONOMOUS SYSTEMS: ROBUST PREDICTIVE ALGORITHMS

Yash Vardhan Pant

A DISSERTATION

in

Electrical and Systems Engineering

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2019

Rahul Mangharam, Associate Professor of Electrical and Systems Engineering
Supervisor of Dissertation

Victor Preciado, Associate Professor of Electrical and Systems Engineering
Graduate Group Chairperson

Dissertation Committee:
George Pappas, Professor of Electrical and Systems Engineering
Manfred Morari, Professor of Electrical and Systems Engineering
Georgios Fainekos, Associate Professor of Computing, Informatics and Decision Systems Engineering,
Arizona State University
Jyotirmoy Deshmukh, Assistant Professor of Computer Science,
University of Southern California
SAFE PLANNING AND CONTROL OF AUTONOMOUS SYSTEMS:
ROBUST PREDICTIVE ALGORITHMS

COPYRIGHT

2019

Yash Vardhan Pant
To Sirisha, without whom I would not have gotten into this academic business anyway, for better or worse.
Acknowledgments

My friends and family. My advisor who gave me a chance that few others, if any, were willing to. Members of the dissertation committee, who have gently pushed me to improve and have offered help and guidance on more than just the contents of this thesis. The faculty members at Penn who gave me the technical knowledge required to execute this, and the staff at Penn (especially at the ESE office) who played their part in sustaining my efforts. Shout out to Habbas, the music and musicians of Philadelphia and the breweries of Minneapolis.
ABSTRACT

SAFE PLANNING AND CONTROL OF AUTONOMOUS SYSTEMS: ROBUST PREDICTIVE ALGORITHMS

Yash Vardhan Pant
Rahul Mangharam

Safe autonomous operation of dynamical systems has become one of the most important research problems. Algorithms for planning and control of such systems are now finding place on production vehicles, and are fast becoming ubiquitous on the roads and air-spaces. However most algorithms for such operations, that provide guarantees, either do not scale well or rely on over-simplifying abstractions that make them impractical for real world implementations. On the other hand, the algorithms that are computationally tractable and amenable to implementation generally lack any guarantees on their behavior.

In this work, we aim to bridge the gap between provable and scalable planning and control for dynamical systems. The research covered herein can be broadly categorized into: i) multi-agent planning with temporal logic specifications, and ii) robust predictive control that takes into account the performance of the perception algorithms used to process information for control.

In the first part, we focus on multi-robot systems with complicated mission requirements, and develop a planning algorithm that can take into account a) spatial, b) temporal and c) reactive mission requirements across multiple robots. The algorithm not only guarantees continuous time satisfaction of the mission requirements, but also that the generated trajectories can be followed by the robot.

The other part develops a robust, predictive control algorithm to control the dynamical system to follow the trajectories generated by the first part, within some desired bounds. This relies on a contract-based framework wherein the control algorithm controls the dynamical system as well as a resource/quality trade-off in a perception-based state estimation algorithm. We show that this predictive algorithm remains feasible with respect to constraints while following a desired trajectory, and also stabilizes the dynamical system under control.

Through simulations, as well as experiments on actual robotic systems, we show that the planning method is computationally efficient as well as scales better than other state-of-the art algorithms that use similar formal specifications. We also show that the robust control algorithm provides better control performance, and is also computationally more efficient than similar algorithms that do not leverage the resource/quality trade-off of the perception-based state estimator.
# Contents

**Title**

**Acknowledgments**

**Abstract**

**Contents**

**List of Tables**

**List of Figures**

## 1 Introduction

1.1 Challenges in planning and control of autonomous systems . . . . . . 3
1.2 Contributions of this work . . . . . . . . . . . . . . . . . . . . . . . . 4
1.2.1 Connection between the two themes . . . . . . . . . . . . . . . . . 4
1.3 Organization of the document . . . . . . . . . . . . . . . . . . . . . . 5

## 2 Representing mission requirements in temporal logic

2.1 Metric Temporal Logic (MTL) . . . . . . . . . . . . . . . . . . . . . . 8
2.1.1 Robust semantics of MTL . . . . . . . . . . . . . . . . . . . . . . 9
2.2 Signal Temporal Logic (STL) . . . . . . . . . . . . . . . . . . . . . . 10
2.2.1 Robustness of STL specifications . . . . . . . . . . . . . . . . . . 11

## 3 Smooth Operator: Control with Temporal Logic Objectives

3.1 Introduction: Controlling for robustness . . . . . . . . . . . . . . . . . 13
3.1.1 The need for temporal logic . . . . . . . . . . . . . . . . . . . . . 14
3.2 Smooth approximation of MTL Robustness for Control . . . . . . . . . 15
3.2.1 Approximating the distance function . . . . . . . . . . . . . . . . 16
3.2.2 Smooth max and min . . . . . . . . . . . . . . . . . . . . . . . . 16
3.2.3 Overall approximation . . . . . . . . . . . . . . . . . . . . . . . 17
3.2.4 The need for smoothing . . . . . . . . . . . . . . . . . . . . . . 19
3.3 Approximation and control . . . . . . . . . . . . . . . . . . . . . . . . 19
3.3.1 Approximation error . . . . . . . . . . . . . . . . . . . . . . . . . 19
# Anytime Computation and Control for Autonomous Systems

## 6.1 Introduction

## 6.2 Co-design of estimation and control

## 6.3 Control with Contract-driven Estimation

### 6.3.1 System Model

### 6.3.2 Time-Triggered Sensing and Actuation

### 6.3.3 Control Performance

### 6.3.4 Discretized Dynamics

## 6.4 Robust Model Predictive Control Solution

### 6.4.1 Solution overview

## 6.5 Stochastic Model Predictive Control Solution

### 6.5.1 Solution overview

## 6.6 Contract based perception algorithms

### 6.6.1 Profiling and Creating an Anytime Contract Perception-based State Estimation Algorithm

### 6.6.2 Run-time execution of the contract-driven perception algorithm

### 6.6.3 Visual Odometry

## 6.7 Case Study: Feedback control of a hex-rotor robot

### 6.7.1 Experimental Setup

### 6.7.2 Experiment design

### 6.7.3 Experimental Results

## 6.8 Conclusion

## 6.9 Proofs of the main results

## 6.10 The Robust case

### 6.10.1 Tractable RAMPC Algorithm

### 6.10.2 RMPC Formulation

### 6.10.3 Proofs of Feasibility

## 6.11 The Stochastic case

### 6.11.1 Constraint tightening

### 6.11.2 Sketch of proof for recursive feasibility

## 6.12 More details on the experimental setup

# Robust Model Predictive Control for Constrained Non-Linear Systems

## 7.1 Introduction

## 7.2 Problem Formulation

## 7.3 Robust MPC for the feedback linearized system

### 7.3.1 State and Input Constraints for the Robust MPC

### 7.3.2 The Control Algorithm

### 7.3.3 Robust Feasibility and Stability

## 7.4 Set definitions for the RMPC

### 7.4.1 Approximating the reach set of the nonlinear system

### 7.4.2 Approximating the bounding sets for the input
List of Tables

3.1 Example 4. Runtimes (mean and standard deviation, in seconds) for Smooth Operator (SOP) and BluSTL (BIS) in modes (B) and (R), over 100 runs with random initial states and different formula horizons $N$. BluSTL(R) did not finish (see text). .................................................. 22

3.2 HVAC. Runtimes (mean ± std deviation, in seconds) SOP and BluSTL (BIS) over 100 runs with random initial states. .................................................. 25

3.3 Robustness of final trajectory, $\rho^*$, for 3 runs with different initial trajectories ($x_0$), none of which satisfy $\varphi$. .................................................. 28

4.1 Stop-and-Go motion. Mean ± standard deviation for run-times (in seconds) and robustness values from 100 runs of the reach-avoid problem. .......... 46

4.2 Free Endpoint Velocity motion. Mean ± standard deviation for runtimes (in seconds) and robustness values from 100 runs of the reach-avoid problem. 47

4.3 Mean ± standard deviation for runtimes (in seconds) and robustness values for one-shot optimization. Obtained from 50 runs of the multi-mission problem with random initial positions. ........................................ 48

4.4 Average runtime per time-step (in seconds) of shrinking horizon robustness maximization in Boolean mode. These are averaged over 5 repetitions of the experiment from the same initial point, to demonstrate the reproducibility of the experiments. .................................................. 52

4.5 Links for the videos for simulations and experiments. Here, Sim. stands for Matlab simulations, CP2.0 for experiments on the Crazyflies. Stop-go and vel. free are the two modes of operation of the trajectory generator, and B (R) is the Boolean (Robust) mode of solving the control problem. Shr. Hrz. stands for the shrinking horizon mode for online control. The reader is advised to make sure while copying the link that special characters are not ignored when pasted in the browser. ................................. 53

6.1 SVO Modes used in the experiments .......................................................... 86

6.2 Fraction of time spent in modes: Hourglass trajectory, RAMPC ........ 91

6.3 Fraction of time spent in modes: Hourglass trajectory, SAMPC .......... 91

6.4 Fraction of time spent in modes: Spiral trajectory, RAMPC ............... 91

6.5 Fraction of time spent in modes: Spiral trajectory, SAMPC ............... 91
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Multiple autonomous drone missions in an urban environment.</td>
</tr>
<tr>
<td>1.2</td>
<td>Organization of the research contributions of this work. Chapters 3 and 4 address the problem of planning and control with temporal logic objectives, while chapters 6 and 7 address the robust predictive control problem.</td>
</tr>
<tr>
<td>1.3</td>
<td>Generated trajectories and robustness boxes within which to track. The STL specification corresponds to the two drones reaching the green goal set within time interval of 6 seconds, while making sure the two drones do not enter the unsafe red set, or crash into each other. If the drones are following their trajectories within the given (blue for drone 1 and black for drone 2) boxes at the corresponding time step, then they satisfy the STL specification. Chap. 4 presents the planning method that generates these trajectories as well as the bounds within which to track them. Chap. 6 presents a control method to follow these trajectories within the given robustness boxes while taking into account the errors and delays associated with the perception-based state estimation commonly used in feedback control.</td>
</tr>
<tr>
<td>2.1</td>
<td>This illustration shows a UAS and two trajectories, ( x_1 ) (in black) and ( x_2 ) (in blue). Color in digital version.</td>
</tr>
<tr>
<td>3.1</td>
<td>Iterates of SQP for Example 3. Colors in online version.</td>
</tr>
<tr>
<td>3.2</td>
<td>Robustness approximation error against formula horizon, evaluated on 1000 randomly generated trajectories for Example 4. Unless noted, the systems are 2D. Color in online version.</td>
</tr>
<tr>
<td>3.3</td>
<td>The first 4 trajectories are for Example 4. The last trajectory, SOP(R, unicycle), is from Example 5. Colors in online version.</td>
</tr>
<tr>
<td>3.4</td>
<td>Zone temperatures. The green rectangle shows the comfortable temperature limit of 22-28 C, applicable during time steps 10-19 (when the building is occupied). Color in online version.</td>
</tr>
</tbody>
</table>
3.5 Trajectories obtained via SQP on smooth robustness, with three different initial trajectories acting as initial solutions for the SQP. Note, all 3 trajectories satisfy \( \varphi \). Here, \( p_j^i \) refers to the positions of the \( i^{th} \) trajectory for the \( j^{th} \) quadrotor. A real-time playback of trajectories can be seen in [https://youtu.be/FU3Rg1Jb7Fw](https://youtu.be/FU3Rg1Jb7Fw).

4.1 (Top) Five Crazyflie 2.0 quadrotors executing a reach-avoid mission. (Bottom) A screenshot of a simulation with 16 quadrotors. In both cases, the quadrotors have to satisfy a mission given in STL.

4.2 The control architecture. Given a mission specification in STL, the high-level control optimization (centralized) generates a sequence of waypoints. These waypoints are sent over to the drones, and through a hierarchical control on-board control architecture, the resulting trajectories are tracked near perfectly, with the continuous time behavior of the system satisfying the STL specification.

4.3 World frame and rotation angles (left) and quadrotor frame with angular rates (right). \( g \) is gravitational force. Figure adapted from Fig. 2 in Mueller et al. [2015].

4.4 Planar splines connecting position waypoints \( p_0, p_1 \) and \( p_2 \). \( q^0 \) is the continuous spline (positions, velocities and accelerations) connecting \( p_0 \) and \( p_1 \) and \( q_1 \) is the spline from \( p_1 \) to \( p_2 \). \( q(kdt') \) is the \( k^{th} \) sample of \( q^0 \), with sampled time \( dt' \). Note, unlike the stop-go case, non-zero end point velocities mean that the resulting motion is not simply a line connecting the way points.

4.5 The upper and lower bounds on \( p_f \) due to the acceleration and velocity constraints. Shown as a function of \( v_0 \) for \( t = 0,0.1,\ldots,T_f = 1 \). The shaded region shows the feasible values of \( p_f \) as a function of \( v_0 \).

4.6 The functions \( K_1 \) to \( K_4 \) for \( T_f = 1 \).

4.7 The environment for the multi-mission problem.

4.8 The desired and actual trajectories for a Crazyflie 2.0 flying the reach-avoid problem with closed-loop control. The unsafe set is in red and the goal set is in green (Color in online version).

4.9 The desired (dashed green) and actual (blue) positions and velocities along the 3 axes (color in online version). The near-perfect tracking shows the dynamical feasibility of the selected waypoints.
5.1 The Fly-by-Logic tool-chain. Through a MATLAB-based graphical interface (figure 5.2), the user defines the workspace and the multi UAS mission. This mission is interpreted as an STL specification (of the form in equation 5.1), the parameters of which are passed from the interface to the Fly-by-Logic C++ library. Through interfacing with off-the-shelf optimization tools, trajectories that satisfy the mission are generated for each UAS and visualized through the user interface. The way-points that generate these trajectories can also be sent to a Robot Operating Systems (ROS) implementation of trajectory following control to be deployed on board actual robots (e.g. bit.ly/varve18).

5.2 The user interface and the planned trajectories for a two UAS patrolling mission (see example 6). Real-time playback can be seen at http://bit.ly/fblguiexmpl.

5.3 Trajectories for 3 UAS tasked with reaching the green-colored goal set within 2 minutes, while avoiding the all black-colored obstacles. A pairwise separation requirement of $\geq 2$ meters is enforced for all UAS.

5.4 Workspace for the PHL airport example. A 10x speedup of the planned trajectories are shown in https://youtu.be/vBYRFfnLwu8.

6.1 Contract-driven controller and estimator. The co-design allows the controller to set a contract for the perception-based state estimator, in addition to controlling the dynamical system.

6.2 Effect of delay, error values on control performance of a PID tracker.

6.3 Autonomous hexrotor with downward-facing camera flying over synthetic features.

6.4 Contract-driven estimator and controller. With knowledge of the estimator’s performance through offline profiling, the controller both actuates the dynamical system and sets contracts for the estimator at run-time in order to maximize control performance while guaranteeing that constraints on the system are always satisfied.

6.5 Time-triggered sensing and actuation. The figure shows the varying execution time for the estimator and the blue area shows the execution time for the controller, which is small.

6.6 Illustration of the building blocks used to compose the Contract Object Detector and their representation as real-time tasks. For a given $(\delta, \epsilon)$ contract, knob settings are chosen at run-time resulting in a schedule to execute these sequential components, or tasks, to respect the contract.

6.7 Profiled delay-error curve for the object detection tool chain run at different parameter settings.
6.8 The profiling process to characterize the performance of SVO in terms of estimation error, computation time and power consumption. Sensor and ground truth data is logged from flights of the hexrotor, and then played back and processed offline to generate the error-delay curve (shown in Fig. 6.9) for SVO. The code snippet shows how little modification is needed to the SVO code base to be able to profile its timing characteristics. Through this offline profiling process, we avoid the need of performing separate flights for each knob setting of SVO.

6.9 (Color online) Error-delay curve for the SVO algorithm running on the Odroid-U3 with different settings of maximum number of features ($\#C$) to detect and track. The vertical line shows the cut-off for maximum delay and the SVO settings that are allowable (upto $\#C = 200$) for closed loop control of a hexrotor at 20Hz. No value of $\#C$ is used above this as it results in the delay approaching the sampling period of the controller.

6.10 The control and computational components on-board the hex-rotor.

6.11 The two reference trajectories, the spiral is in dashed red and the hourglass is in solid black (color in online version). The figure on the right shows the trajectories projected on the x,y plane. Note, the spiral starts on the outside and ends inwards while the hourglass trajectory starts and ends at (0,0,1).

6.12 Performance, hourglass trajectory. The vertical axis has the average control performance (eq. 6.8) over the flights for the labeled settings, with lower values implying better control performance. The horizontal axis shows the computation power (in Joules) consumed by SVO to perform the state estimation task. The figure shows how our methods (RAMPC/SAMPC) leveraging the co-design have both better control performance while consuming less computation power than the baseline method.

6.13 Performance, spiral trajectory. The vertical axis has the average control performance (eq. (6.8)) over the flights for the labeled settings, with lower values implying better control performance. The horizontal axis shows the computation power (in Joules) consumed by SVO to perform the state estimation task. Similar to the case for the hourglass trajectory, our methods outperform the baseline.

6.14 (Color online) Reference positions (dashed red) and actual positions (blue) of the hex-rotor flying the hourglass trajectory while being controlled by the SAMPC ($\alpha = 0$).

6.15 (Color online) Reference positions (dashed red) and actual positions (blue) of the hex-rotor flying the spiral trajectory while being controlled by the RAMPC ($\alpha = 0.1$).
6.16 SVO Mode and control cost over time for the spiral trajectory flown with SAMPC at $\alpha = 0.001$.

6.17 Cumulative distribution of profiled execution times for visual odometry running on the Odroid-U3 for varying maximum number of corners from the SVO algorithm.

7.1 The outer-approximated reach sets for $x_{k+j}$, computed at time steps $k, k+1$, used to compute $E_{k+j|k}$, $V_{k+j|k}$.

7.2 Local and global inner approximations of input constraints for running example, with $X_{k+j|k} = [-\pi/4, 0] \times [-0.9666, -0.6283]$ for some $k, j$ and $U = [-2.75, 2.75]$. Color in online version.

7.3 The error sets $E_{max}$ and $E$ computed over an arbitrary $X_{k+j|k}$. Also shown are realizations of $\hat{e} := T(\hat{x}) - T(x)$ for randomly chosen $x \in X$. Color in online version.

7.4 The states and their estimates of the feedback linearized and non-linear running example. Recall that $z_1 = x_1$ therefore to reduce clutter, we only plot the first state only for the feedback linearized system. Color in online version.

7.5 Inputs $v$ and $u$ and their bounds for the running example. Color in online version.

7.6 Flexible joint manipulator. Figure adapted from Seidi et al. [2012].

7.7 The states and their estimates of the feedback linearized and non-linear manipulator. Recall that $z_1 = x_1$ and $z_2 = x_2$, therefore to reduce clutter, we only plot first two states only for the feedback linearized system. Color in online version.

7.8 Inputs $v$ and $u$ and their bounds for the manipulator example. Color in online version.
Chapter 1

Introduction

With autonomous systems no longer restricted to the sterile environments, the problem of their safe planning, i.e. generating trajectories that satisfy given mission requirements, and control, i.e. actuating the dynamical system to follow the desired trajectory, is now one of utmost importance. From self-driving cars, to underwater robots, to multi-rotor drones, autonomous robotic systems are finding widespread applications with complex operating requirements, creating new safety concerns along with them.

Example 1.1. Fig. 1.1 shows a scenario where multiple quadrotors have to fly a variety of missions in a common air space, including package delivery, surveillance,
and infrastructure monitoring. Drone A is tasked with delivering a package, which it has to do within 15 minutes and then return to base in the following 10 minutes. Drone B is tasked with periodic surveillance and data collection of the wildlife in the park, while Drone C is tasked with collecting sensor data from equipment on top of the white-and-blue building. All of these missions have complex spatial requirements (e.g. avoid flying over the buildings highlighted in red, perform surveillance or monitoring of particular areas and maintain safe distance from each other), temporal requirements (e.g., a deadline to deliver package, periodicity of visiting the areas to be monitored) and reactive requirements (like collision avoidance).

The scenario outlined in the above example is now close to being a reality. In the United States of America, the National Aeronautics and Space Association (NASA) and the Federal Aviation Authority (FAA) have been studying the regulation of the airspace when multiple fleets of autonomous drones share the same airspace, outlined in the Concept of Operations document (ConOps) [Federal Aviation Authority 2018]. While their focus is on the infrastructure and management of the airspace, the Unmanned Aerial Systems (UAS) Traffic Management (UTM) ConOps makes it clear that the safety (separation from other aircraft, terrain, and other hazards) is a responsibility of the drone fleet operators. The ConOps also outline a potential airspace reservation based system for operation where operators reserve a volume of the airspace for a given time interval to operate in. This approach is no doubt conservative, and will not scale as the airspace gets more crowded.

In fact, the problem of planning for multiple aerial vehicles in the same airspace is not an entirely new one. Air Traffic Control (ATC) operations worldwide guide manned commercial aircrafts from their source to destination while maintaining safety. ATC relies on human controllers to guide aircrafts along pre-designed routes while scheduling/directing operations in a way that maintains separation between aircrafts. Given the number of UAS expected to operate in urban airspaces in the near future, as well as the short time frames of the missions and communication constraints, it is impossible to have humans operators for UTM operations. This motivates the need to have an mission planning system similar to ATC, but for UTM operations.

In UTM operations, each UAS will not be simply going from a source to a destination but may in fact be performing a complicated set of tasks, often working together with multiple other UAS, e.g. in disaster management [Homola et al. 2018]. This necessitates that the UTM framework be capable to taking into account complicated mission requirements beyond point to point navigation, as well as be able to handle multiple UAS sharing the same airspace at the same time.

Finally, also of consideration is the control of autonomous systems when they rely on perception based algorithms for localization and sensing the environment around them. A mission/flight plan designed by the UTM needs to be such that the UAS can follow it closely, but also be robust enough that small deviations from the planned trajectories do not result in unsafe situations. Similarly, the control of autonomous systems needs to be robust to localization and other sensor errors while following the
planned trajectories to within a given tracking error bound.

We aim to take into account these requirements and develop planning and control methods for safe operation of autonomous systems.

1.1 Challenges in planning and control of autonomous systems

In order to deal with complicated missions requirements, as those outlined in Example 1.1, most existing approaches either lack the expressiveness to handle such requirements e.g. [Ma et al. 2016], [van den Berg and Overmars 2015], rely on simplifying assumptions that result in conservative or infeasible behavior [Aksaray et al. 2016], or do not take into account the explicit timing requirements [Saha et al. 2014]. In addition to these limitations, many of the planning methods are computationally intractable (and hence do not scale well or work in real-time), and provide guarantees only on a simplified abstraction of the system behavior [Aksaray et al. 2016]. This leads to the additional problem that plans/trajectories generated by the planning methods may in practice be impossible for the real dynamical system to follow [Kantaros and Zavlanos 2018]. A detailed review of existing approaches is presented in Sec. 8.2.

Despite robot planning and control being a well studied in literature [Elbanhawi and Simic 2014], [Yang et al. 2016], dealing with the complicated requirements, e.g. those of Example 1.1 pose a fundamental problem due to:

1. *Explicit temporal constraints*: Asking a dynamical system to satisfy a particular task in some given interval of time adds challenges that are not well studied in literature outside of temporal logic based planning/control (see chap. 8 for more details). Most multi/single-robot planning algorithms either ignore time bounds or aim to achieve the minimum time to completion [Guo and Parker 2002], which may not be well suited to a particular application.

2. *Multi-agent co-operation across tasks*: Planning for multiple agents when they have to perform tasks in a dependent manner adds another layer of complexity. Common approaches to solving these problems either impose task priorities [De-wangan et al. 2017] or rely on other heuristics to simplify the problem, possibly at the cost of sacrificing performance.

3. *Reactive constraints*: Handling certain events caused by changes in the environment or the system in a systematic if-then framework, along with temporal requirements, is a complicated problem that has not been much studied so far. Approaches to handing reactive requirements do not take temporal requirements into account or do not scale very well [Kress-Gazit et al. 2009].
4. *Disconnect between planing and control*: Classical planning methods like $A^*$ rely on a discrete representation of the workspace and result in jagged trajectories that later need to be smoothed out for an actual dynamical system to follow them. As the safety guarantees are on the original trajectory only, the smoothed trajectory may be unsafe.

5. *Robust control with perception based estimation*: The problem of control for trajectory following under uncertainty, while well studied, rarely takes into account the impact of the computation time or energy taken to generate a state estimate from a perception (e.g. vision, lidar-based) driven estimator. This can, in practice result in poor control performance and reduced operating time for a robot.

The research presented here aims to develop computationally tractable algorithms that also provide performance guarantees in order to address the above issues.

### 1.2 Contributions of this work

The research carried out, in order to deal with some of the challenges outlined above, can be divided into two categories.

1. **Multi-agent planning with Signal temporal logic (STL) objectives.** This part focuses on generating trajectories for autonomous robots such that they satisfy objectives specified using STL.

2. **Robust predictive control with anytime perception.** The second part focuses on predictive control algorithms to follow the trajectories generated by our planning methods while staying within a predefined *robustness tube* around the desired trajectory. We explicitly take into account the time and energy consumption of perception-based estimation algorithms that give us state estimation as feedback, and show that we do not always need the best quality state estimate to perform the control task. By varying the quality and resource consumption of the state estimator online, the control schemes we develop result in improved control performance while being efficient with regards to the energy consumed by the computation.

#### 1.2.1 Connection between the two themes

In Chapter 4, we present a method that generates trajectories for fleets of multi-rotor drones such that they satisfy STL specifications. Along with the trajectories, the *robustness* (see Sec. 2.1.1) value associated with them specify time-varying, box-like bounds within which the state of the robotic system must lie in order to satisfy the STL specification. Fig. 1.3 shows a visualization of this. The blue and black boxes
Figure 1.2: Organization of the research contributions of this work. Chapters 3 and 4 address the problem of planning and control with temporal logic objectives, while chapters 6 and 7 address the robust predictive control problem.

show the time-varying sets in which the position of the two quadrotors should be within at the corresponding time step, in order to satisfy the STL specification.

This problem of following the trajectory within given bounds is explored in chap. 6, where we develop a method that does this while also taking into account the behavior of the perception-based estimator that supplies the state-feedback for the controller. The robust adaptive model predictive control algorithm (RAMPC) presented in that chapter takes as reference the trajectories to track as well as the constraints within which to track them (see fig. 1.3) from the planning method. The planning method on the other hand takes into account a characterization of the region within which the controller can track well (see def. 4.1).

1.3 Organization of the document

This document will cover work in each of two themes outlined in Fig. 1.2, as well as the connection between them. Chapter 2 presents a brief introduction to the types of temporal logic the methods presented here are applicable to. Chapter 3 covers the problem of control of dynamical systems with Metric Temporal Logic (MTL) objectives. It also introduces a smooth robustness metric associated with the MTL
specifications. Chapter 4 covers Fly-by-Logic, a method to generate trajectories for multiple multi-rotor UAS such that they satisfy a given STL specification. Chapter 5 presents a user-interface for multi-rotor UAS fleet planning and the briefly covers the underlying tool-chain.

Chapter 6 presents an approach to design a Robust Adaptive Model Predictive Control (RAMPC) algorithm that we use for trajectory tracking. The RAMPC algorithm ensures that the dynamical system remains feasible with respect to state and input constraints despite estimation errors and computation delays due to perception-based estimation algorithms. It also leverages a co-design between the computation and control to provide good control performance as well as computation energy consumption compared classical MPC approaches that do not. Chapter 7 presents a feedback linearization-based Robust Model Predictive Control method that can allow the co-design framework to be directly extended to non-linear (control-affine) systems.

In addition to these, chapter 8 presents the existing research that is relevant to the material covered in this document.
Figure 1.3: Generated trajectories and robustness boxes within which to track. The STL specification corresponds to the two drones reaching the green goal set within time interval of 6 seconds, while making sure the two drones do not enter the unsafe red set, or crash into each other. If the drones are following their trajectories within the given (blue for drone 1 and black for drone 2) boxes at the corresponding time step, then they satisfy the STL specification. Chap. 4 presents the planning method that generates these trajectories as well as the bounds within which to track them. Chap. 6 presents a control method to follow these trajectories within the given robustness boxes while taking into account the errors and delays associated with the perception-based state estimation commonly used in feedback control.
Chapter 2

Representing mission requirements in temporal logic

In this document, the problem of multi-agent planning is studied through the lens of planning with missions given in temporal logic form. This chapter covers the basics of a class of temporal logic that we are interested in, as well as the concept of robustness that is central to the methods presented in the following chapters.

Consider a discrete-time dynamical system \( H \) given by

\[
x_{t+1} = f(x_t, u_t)
\]

(2.1)

where \( x \in X \subset \mathbb{R}^n \) is the state of the system and \( u \in U \subset \mathbb{R}^m \) is its control input. The system’s initial state \( x_0 \) takes value from some initial set \( X_0 \subset \mathbb{R}^n \). Given an initial state \( x_0 \) and a finite control input sequence \( u = (u_0, \ldots, u_{T-1}), u_t \in U \), a trajectory of the system is the unique sequence of states \( x = (x_0, \ldots, x_T) \) s.t. for all \( t, x_t \) is in \( X \) and obeys (2.1). All temporal intervals that appear here are implicitly discrete-time, e.g. \( [a, b] \) means \( [a, b] \cap \mathbb{N} \). The set \( \{0, 1, \ldots, T\} \subset \mathbb{N} \) will be abbreviated as \( T \). For an interval \( I \subset \mathbb{N} \), let \( t + I = \{t + a \mid a \in I\} \). The set of subsets of a set \( S \) is denoted \( \mathcal{P}(S) \). The signal space \( X^T \) is the set of all signals \( x : T \rightarrow X \). The max operator is written \( \sqcup \) and min is written \( \sqcap \).

2.1 Metric Temporal Logic (MTL)

The controller of \( H \) is designed to make the closed loop system (2.1) satisfy a specification expressed in MTL [Ouaknine and Worrell 2008]. Formally, let \( AP \) be a set of atomic propositions, which can be thought of as point-wise constraints on the state of the system. An MTL formula \( \varphi \) is built recursively from the atomic propositions using the following grammar:

\[
\varphi ::= T|p|\neg \varphi_1 \land \varphi_2 | \varphi_1 U_{I} \varphi_2
\]
where \( I \subset \mathbb{R} \) is a time interval. Here, \( \top \) is the Boolean True, \( p \) is an atomic proposition, \( \neg \) and \( \wedge \) are the Boolean negation and AND operators, respectively, and \( U \) is the Until temporal operator. Informally, \( \varphi_1 U \varphi_2 \) means that \( \varphi_1 \) must hold \textit{until} \( \varphi_2 \) holds, and that the hand-over from \( \varphi_1 \) to \( \varphi_2 \) must happen sometime during the interval \( I \). The disjunction (\( \lor \)), implication (\( \implies \)), Always (\( \Box \)) and Eventually (\( \Diamond \)) operators can be defined using the above operators.

Formally, the \textit{pointwise semantics} of an MTL formula define what it means for a system trajectory \( x \) to satisfy the formula \( \varphi \). Let \( O : AP \to P(X) \) be an observation map for the atomic propositions. The boolean truth value of a formula \( \varphi \) w.r.t. the trajectory \( x \) at time \( t \) is defined recursively.

\[ (x, t) \models \top \iff \top \]
\[ \forall p \in AP, (x, t) \models_{O} p \iff x_t \in O(p) \]
\[ (x, t) \models_{O} \neg \varphi \iff \neg(x, t) \models_{O} \varphi \]
\[ (x, t) \models_{O} \varphi_1 \wedge \varphi_2 \iff (x, t) \models_{O} \varphi_1 \wedge (x, t) \models_{O} \varphi_2 \]
\[ (x, t) \models_{O} \varphi_1 U \varphi_2 \iff \exists t' \in t + I. (x, t') \models_{O} \varphi_2 \wedge \forall t'' \in (t, t'), (x, t'') \models_{O} \varphi_1 \]

As \( O \) is fixed here, it is dropped from the notation. We say \( x \) satisfies \( \varphi \) if \( (x, 0) \models \varphi \). All formulas that appear here have bounded temporal intervals: \( 0 \leq \inf I < \sup I < +\infty \). To evaluate whether such a formula \( \varphi \) holds on a given trajectory, only a finite-length prefix of that trajectory is needed. Its length can be upper-bounded by the \textit{horizon} of \( \varphi \), \( hrz(\varphi) \in \mathbb{N} \), calculable as shown in [Dokhanchi et al.] 2014. For example, the horizon of \( \Box_{[0,2]}(\Diamond_{[2,4]}p) \) is 2+4=6.

\[ \text{2.1.1 Robust semantics of MTL} \]

Designing a controller that satisfies the MTL formula \( \varphi \) is not always enough. In a dynamic environment, where the system must react to new unforeseen events, it is useful to have a margin of maneuverability. That is, it is useful to control the system such that we \textit{maximize} our degree of satisfaction of the formula. When unforeseen events occur, the system can react to them without violating the formula. This degree of satisfaction can be formally defined and computed using the robust semantics of MTL. Given a point \( x \in X \) and a set \( A \subset X \), \( \text{dist}(x, A) := \inf_{a \in \overline{A}} |x - a|_2 \) is the minimum Euclidian distance from \( x \) to the closure \( \overline{A} \) of \( A \).

\[ \text{Definition 2.2 (Robustness [Fainekos and Pappas 2009])} \]

The \textit{robustness} of \( \varphi \) relative \(^1\)Strictly, a controller s.t. the closed-loop behavior satisfies the formula.
to \( x \) at time \( t \) is recursively defined as

\[
\rho_\top(x, t) = +\infty
\]

\[
\forall p \in AP, \rho_p(x, t) = \begin{cases} \text{dist}(x_t, X \setminus \mathcal{O}(p)), & \text{if } x_t \in \mathcal{O}(p) \\ -\text{dist}(x_t, \mathcal{O}(p)), & \text{if } x_t \notin \mathcal{O}(p) \end{cases}
\]

\[
\rho_{\neg \varphi}(x, t) = -\rho_\varphi(x, t)
\]

\[
\rho_{\varphi_1 \land \varphi_2}(x, t) = \rho_{\varphi_1}(x, t) \sqcap \rho_{\varphi_2}(x, t)
\]

\[
\rho_{\varphi_1 \mathcal{U} I \varphi_2}(x, t) = \sqcup_{t' \in t + T I}(\left[ \rho_{\varphi_2}(x, t') \right] \sqcap \left[ \sqcap_{t'' \in [t, t')} \rho_{\varphi_1}(x, t'') \right])
\]

When \( t = 0 \), we write \( \rho_\varphi(x) \) instead of \( \rho_\varphi(x, 0) \).

The robustness is a real-valued function of \( x \) with the following important property.

**Theorem 2.1.** [Fainekos and Pappas 2009] For any \( x \in X^T \) and MTL formula \( \varphi \), if \( \rho_\varphi(x, t) < 0 \) then \( x \) violates the spec \( \varphi \) at time \( t \), and if \( \rho_\varphi(x, t) > 0 \) then \( x \) satisfies \( \varphi \) at \( t \). The case \( \rho_\varphi(x, t) = 0 \) is inconclusive.

Thus, we can compute control inputs by maximizing the robustness over the set of finite input sequences of a certain length. The obtained sequence \( u^* \) is valid if \( \rho_\varphi(x, t) \) is positive, where \( x \) and \( u^* \) obey (2.1). The larger \( \rho_\varphi(x, t) \), the more robust is the behavior of the system: intuitively, \( x \) can be disturbed and \( \rho_\varphi \) might decrease but not go negative.

### 2.2 Signal Temporal Logic (STL)

In chapter [4] we present a method for multi-rotor drone fleet planning with mission specifications expressed in Signal Temporal Logic (STL) [Maler and Nickovic 2004, Donzé and Maler 2010b]. Similar to MTL, STL is a logic that allows the succinct and unambiguous specification of a wide variety of desired system behaviors over time, such as “The quadrotor reaches the goal within 10 time units while always avoiding obstacles” and “While the quadrotor is in Zone 1, it must obey that zone’s altitude constraints”.

Formally, let \( M = \{ \mu_1, \ldots, \mu_L \} \) be a set of real-valued functions of the state \( \mu_k : X \to \mathbb{R} \). For each \( \mu_k \) define the predicate \( p_k := \mu_k(x) \geq 0 \). Set \( AP := \{ p_1, \ldots, p_L \} \). Thus each predicate defines a set, namely \( p_k \) defines \( \{ x \in X \mid f_k(x) \geq 0 \} \). Similar to MTL, let \( I \subset \mathbb{R} \) denote a non-singleton interval, \( \top \) the Boolean True, \( p \) a predicate, \( \neg \) and \( \land \) the Boolean negation and AND operators, respectively, and \( \mathcal{U} \) the Until temporal operator. An STL formula \( \varphi \) is built recursively from the predicates using the following grammar:

\[
\varphi ::= \top | p | \neg \varphi | \varphi_1 \land \varphi_2 | \varphi_1 \mathcal{U} \varphi_2
\]
Figure 2.1: This illustration shows a UAS and two trajectories, $x_1$ (in black) and $x_2$ (in blue). Color in digital version.

### 2.2.1 Robustness of STL specifications

STL follows the same grammar as that of MTL and has similarly defined robust semantics as well (see 2.1.1). The notable difference is that the $\text{dist}$ operator in MTL for each predicate $p_k$ is simply replaced by $\mu_k$ in STL, where $\mu_k$ is the real-valued function that defines the predicate $p_k$ as described above. This means that the only difference with respect to definition 2.2 is that $\rho_p(x,t) = \mu(x_t)$, where $x_t$ is the value of signal $x$ at time $t$. The rest of the robust semantics of STL follow the construction of definition 2.2.

The STL specifications we consider in the rest of this document satisfy the following assumption:

**Assumption 2.1.** The function $\mu$ that defines an atomic proposition in STL is continuously differentiable, or $\mu \in C^1$.

The following shows an example of a specification in STL and its associated robustness:

**Example 1.** Consider a safety specification of the form:

$$\varphi_{\text{safe}} = \Box_{[0,T]} \neg(p \in \text{Unsafe}_1) \land \Box_{[0,T]} \neg(p \in \text{Unsafe}_2)$$  \hspace{1cm} (2.2)

This states the position of the UAS $p$ should, in the time interval $[0,T]$, never be inside the region given by Unsafe$_1$ and it should also never be inside Unsafe$_2$. Fig.
shows these regions. Consider the trajectory \( x_1 \) (in black), shown from time 0 to \( T \) seconds. As can be seen, the UAS does indeed avoid the unsafe regions and satisfies the specification, which by Theorem 2.1 implies that the robustness of this trajectory \( x \) with respect to the specification \( \varphi_{safe} \), \( \rho_{\varphi_{safe}}(x) \) is positive.

In order to further understand this robustness value, let us first compute it. Let \( p = [p_x, p_y] \) be the position of the drone in 2-d. The proposition \( p \in Unsafe_1 \) can be written in more detail in STL as \((p_x \leq -1) \land (-p_x \leq 2) \land (p_y \leq 2) \land (-p_y \leq -1)\). This comes from the representation of the set as a bounded axis-aligned polyhedron in \( \mathbb{R}^2 \). Following the robustness semantics of definition 2.2 that states the robustness \( \rho_{\varphi_1 \land \varphi_2} = \min(\rho_{\varphi_1}, \rho_{\varphi_2}) \), the robustness of \( p \in Unsafe_1 \), evaluated at a single point in the trajectory, can be computed as \( \rho_{Unsafe_1}(p) = \min(-1 - p_x, 2 + p_x, 2 - p_y, -1 + p_y) \). As an example, consider the point \([-1.5, 0.75]\) marked by 1 in fig. 2.1. The robustness of this point w.r.t proposition \( p \in Unsafe_1 \) is \( \rho_{Unsafe_1} = \min(0.5, 0.5, 1.25, -0.25) = -0.25 \). This negative robustness implies that the point \([-1.5, 0.75]\) does not satisfy the proposition \( p \in Unsafe_1 \), which is as seen in the figure.

Since a part of the safety specification \( \varphi_{safe} \) asks for \( \neg(p \in Unsafe_1) \), the robustness of this proposition is simply the negative of the robustness of the proposition \( p \in Unsafe_1 \) (again see def. 2.3), or 0.25.

To then evaluate the robustness of \( \square_{[0,T]} \neg(p \in Unsafe_1) \), following def. 2.2, we need to compute the minimum of the robustness of the proposition \( \neg(p \in Unsafe_1) \) over all points from time 0 to \( T \) in trajectory \( x_1 \), i.e. \( \min_{t \in [0,T]}(-\min(-1 - p_x(t), 2 + p_x(t), 2 - p_y(t), -1 + p_y(t))) \). For trajectory \( x_1 \), this minimum is achieved by the point marked by 1, hence the robustness of trajectory \( x_1 \) w.r.t the specification \( \square_{[0,T]} \neg(p \in Unsafe_1) \) is 0.25. Similarly we can compute the robustness of the specification \( \square_{[0,T]} \neg(p \in Unsafe_2) \). The robustness of the safety specification \( \varphi_{safe} \) is then (using \( \rho_{\varphi_1 \land \varphi_2} = \min(\rho_{\varphi_1}, \rho_{\varphi_2}) \)) given by minimum of the robustness of \( \square_{[0,T]} \neg(p \in Unsafe_1) \) and \( \square_{[0,T]} \neg(p \in Unsafe_2) \). For the trajectory \( x_1 \), this value is achieved by the point marked by 1, and is hence 0.25.

This value of 0.25 implies that each point in the trajectory \( x_1 \) could be moved by at most 0.25m along any axis and still the trajectory would satisfy the specification \( \varphi_{safe} \). Again, focusing on the point marked by 1 helps explain this. If we move 1 along the \( y \)-axis by upto 0.25m, 1 still does not enter the set \( Unsafe_1 \). By moving it 0.25m in the \( y \)-axis would bring it to the boundary of the unsafe set, and higher values would push it into the unsafe set, violating the requirement that the trajectory never enters this set.
Chapter 3

Smooth Operator: Control with Temporal Logic Objectives

3.1 Introduction: Controlling for robustness

The errors in a cyber-physical control system like an automated air traffic controller can affect both the cyber components (e.g., software bugs) and physical components (e.g., sensor failures and attacks) of a system. Under certain error models, like a bounded disturbance on a sensor reading, a system can be designed to be robust to that source of error. In general, however, unforeseen and unmodeled issues will occur and the controller has to deal with them at runtime. To help deal with unforeseen problems at runtime, the system’s controller must make decisions that not only satisfy the system’s requirements (like a maximum response time to an event), but satisfy them robustly. Intuitively, the requirements are robustly satisfied if a disturbance to the system does not cause it to violate them. This can give a margin of maneuverability to the system during which it addresses the unforeseen problem. Since these problems are, by definition, unforeseen and unmodeled and only detected by their effect on the output, the notion of robustness must be computable using only the output behavior of the system.

**Example 2.** Air-Traffic Control (ATC) coordinates landing arrivals at an airport. ATCs have very complex rules to ensure that all airplanes, of different sizes and speeds, approach the airport and land safely, with sufficient margin to other airplanes to accommodate emergencies. Sample rules for the Chicago O’hare airport include (A) When an aircraft enters any of 3 designated zones, it must stay between that zone’s altitude floor and ceiling, and (B) If the airspace is too busy, an aircraft must remain in either holding zones 6 or 7, until some maximum amount of time expires.

How do we ensure that the ATC system satisfies these complex rules robustly?
3.1.1 The need for temporal logic

The above requirements go beyond traditional control objectives like stability, tracking, quadratic cost optimization and reach-while-avoid for which we have well-developed theory. While these requirements can be directly encoded from natural language into a Mixed Integer Program (MIP) by encoding every possibility at each (discrete) time point with integer variables, such a direct encoding can easily involve an exorbitant number of variables. For complex requirements, with many variables involved, this encoding process can also be error-prone and checking that it corresponds to the designer’s intent is near impossible. On the other hand, such control requirements are easily and succinctly expressed in Metric Temporal Logic (MTL) [QuaPWN and Worrell 2008]. MTL is a formal language for expressing reactive requirements with constraints on their time of occurrence and sequencing, such as those of the ATC (see chapter 2). The advantage of first expressing the requirements in MTL is that MTL formulas are more succinct and legible, and less error-prone, than the corresponding directly-encoded MIP. In this sense, MTL bridges the gap between the ease of use of natural language and the rigour of mathematical formulation. For example, ATC rule (A) can be formalized with the following MTL formula (□ means ‘Always’, q is an aircraft and $q_z$ is its altitude).

$$□(q \in \text{Zone1} \implies q_z \leq \text{Ceiling1} \land q_z \geq \text{Floor1})$$

Rule (B) can be formalized as follows.

$$□(\text{Busy} \implies \diamondsuit_{[t_1,t_2]} (q \in \text{Holding-6} \lor q \in \text{Holding-7}) U_{[0,\text{MaxHolding}]} \neg \text{Busy})$$

This says that Always (□), if airport is Busy, then sometime $t_1$ to $t_2$ seconds later ($\diamondsuit_{[t_1,t_2]}$), the plane goes into one of two Holding areas. It stays there Until the airport is not (¬) busy, which must happen before duration MaxHolding elapses.

Given an MTL specification $\varphi$ and a system execution $x$, the robustness $\rho_\varphi(x)$ of the spec relative to $x$ measures two things: its sign tells whether $x$ satisfies the spec ($\rho_\varphi(x) > 0$) or violates it ($\rho_\varphi(x) < 0$). Its magnitude $|\rho_\varphi(x)|$ measures how robustly the spec is satisfied or violated. Namely, any perturbation to $x$ of size less than $|\rho_\varphi(x)|$ will not cause its truth value to change relative to $\varphi$. Thus, we are interested in developing a control algorithm that can maximize the robustness over all possible control actions to determine the next control input.

Unfortunately, the robustness function $\rho_\varphi$ is hard to work with. In particular, it is non-convex and non-differentiable, which makes its online optimization a challenge - indeed, most existing approaches treat it as a black box and apply heuristics to its optimization (see Section 8.1). These heuristics provide little to no guarantees, have too many user-set parameters, and don’t have rigorous termination criteria. On the other hand, gradient descent optimization algorithms typically offer convergence guarantees to the function’s (local) minima, have known convergence rates for certain
function classes, usually have a fewer number of parameters to be set, and important issues like step-size selection are rigorously addressed.

**Contributions.** This chapter presents smooth (infinitely differentiable) approximations to the robustness function of arbitrary MTL formulae. The smooth approximation is proven to always be within a user-defined error of the true robustness, and this is illustrated experimentally. This allows running powerful and rigorous off-the-shelf gradient descent optimizers. We leverage this to maximize the smooth robustness for control of a system to robustly satisfy its MTL specification. Through multiple examples, the proposed control method is shown to be faster and to yield more robust trajectories than various current heuristics and MIP-based approaches. The results are demonstrated on a case study for an autonomous ATC for two quad-rotors, where the MIP-based approach fails to yield a satisfying controller. While this work does not tackle the non-convexity of MTL robustness issue directly, having an inexpensive gradient optimizer makes it possible to run an efficient multi-start optimization, increasing the chances of approaching the global optimum.

### 3.2 Smooth approximation of MTL Robustness for Control

Let $\varphi$ be an MTL formula with horizon $N$. The goal of the present work is to solve the following problem $P_\rho$.

$$
\begin{align*}
P_\rho : \max_u & \quad \rho_\varphi(x) - \gamma \sum_{k=0}^{N-1} l(x_{k+1}, u_k) \\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k), \forall k = 0, \ldots, N - 1 \\
& x_k \in X, \forall k = 0, \ldots, N \\
& u_k \in U, \forall k = 0, \ldots, N - 1 \\
& \delta \rho_\varphi(x) \geq \delta \epsilon_{\text{min}}
\end{align*}
$$

Here, $u = (u_0, \ldots, u_{N-1})$, $l(x_{k+1}, u_k)$ is a control cost, e.g. the LQR cost $x_k'Qx_k + u_k'Rx_k$, and $\gamma \geq 0$ is a trade-off weight. The scalar $\epsilon_{\text{min}} \geq 0$ is a desired minimum robustness. If $\delta = 0$, then this constraint is effectively removed, while $\delta = 1$ enforces the constraint. Because $\rho_\varphi$ uses the non-differentiable functions $\text{dist}$, max and min (see definition 2.2), it is itself non-differentiable. The next three sub-sections introduce smooth approximations to each of these functions.

**Assumption 3.1.** The function $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ that represents the system dynamics in (3.1b) is such that its first and second derivatives exist and are continuous, or $f \in C^2$. 

---

15
3.2.1 Approximating the distance function

The distance function $\text{dist}(\cdot, U)$ is in $L_2(\mathbb{R}^n)$, so it can be approximated arbitrarily well using a Meyer wavelet expansion [DeVore 1998]. Specifically, the 1-D Meyer wavelet function is given in the frequency domain by $(i = \sqrt{-1})$:

$$\hat{\psi}(\omega) = \frac{1}{\sqrt{2\pi}} \begin{cases} \sin\left(\frac{3|\omega|}{2\pi} - 1\right) \epsilon^{i|\omega|/2}, & 2\pi/3 \leq |\omega| \leq 4\pi/3 \\ \cos\left(\frac{3|\omega|}{4\pi} - 1\right) \epsilon^{i|\omega|/2}, & 4\pi/3 \leq |\omega| \leq 8\pi/3 \\ 0, & \text{otherwise} \end{cases}$$

where $\nu(x) = 0$ if $x \leq 0$, $1$ if $x \geq 1$, and equals $x$ if $0 \leq x \leq 1$. The time-domain expression for this wavelet is given in [Vermehren Valenzuela and de Oliveira 2015] and is infinitely differentiable. An $n$-D wavelet can be obtained using the tensor product construction [DeVore 1998]. Let $E$ be the set of vertices of the unit hypercube $[0, 1]^n$. For every $e = (e_1, e_2, \ldots, e_n) \in E$ and $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, define $\Psi^e : \mathbb{R}^n \to \mathbb{R}$ by $\Psi^e(x) = \psi^{e_1}(x_1) \cdots \psi^{e_n}(x_n)$. Given $k \in \mathbb{Z}$ and $j \in \mathbb{Z}^n$, a dyadic cube in $\mathbb{R}^n$ is a set of the form $I = 2^{-k}(j + [0, 1]^n)$. Let $D$ be the set of all dyadic cubes in $\mathbb{R}^n$ obtained by varying $k$ over $\mathbb{Z}$ and $j$ over $\mathbb{Z}^n$. Then $\{\Psi^e, e \in E, I \in D\}$ is an orthonormal basis for $L_2(\mathbb{R}^n)$ (because the Meyer wavelet itself is orthonormal). Then every function in $L_2(\mathbb{R}^n)$ has an expansion

$$f(x) = \sum_{I \in D} \sum_{e \in E} c_I^e \Psi^e_I(x), \quad c_I^e := \langle f, \Psi^e_I \rangle$$

with $\langle h, g \rangle := \int_{\mathbb{R}^n} h(x)g(x)dx$. The desired approximation is obtained by truncating this expansion after a finite number of terms, i.e., by using a finite set $D' \subseteq D$

$$\text{dist}(x, U) \approx \text{dist}_\varepsilon(x, U) := \sum_{I \in D'} \sum_{e \in E} c_I^e \Psi^e_I(x) \quad (3.2)$$

where $\varepsilon$ is the approximation error magnitude. Using more coefficients yields a better approximation. The coefficients $c_I^e := \langle \text{dist}(\cdot, U), \Psi^e_I \rangle$ are calculated offline and stored in a lookup table for online usage.

3.2.2 Smooth max and min

The following standard smooth approximations of $m$-ary max and min are used. Let $k \geq 1$.

$$\widetilde{\max}_k(a_1, \ldots, a_m) := \frac{1}{k} \ln(e^{ka_1} + \cdots + e^{ka_m}) \quad (3.3)$$

$$\widetilde{\min}_k(a_1, \ldots, a_m) := -\widetilde{\max}(-a_1, \ldots, -a_m) \quad (3.4)$$

Suppose $k = 1$ and that $a_1$ is the largest number. Then $e^{a_1}$ is even larger than the other $e^{a_i}$’s, and dominates the sum. Thus $\widetilde{\max}_1(a) \approx \ln e^{a_1} = a_1 = \max(a)$. If $a_1$ is
not significantly larger than the rest, the sum is not well-approximated by $e^{a_1}$ alone. To counter this, the scaling factor $k$ is used: it amplifies the differences between the numbers. It holds that for any set of $m$ reals,

$$0 \leq \max_k(a_1, \ldots, a_m) - \max(a_1, \ldots, a_m) \leq \ln(m)/k \quad (3.5)$$

$$0 \leq \min(a_1, \ldots, a_m) - \min_k(a_1, \ldots, a_m) \leq \ln(m)/k \quad (3.6)$$

with the maximum error is achieved when all the $a_i$'s are equal. Indeed, assume $a_1$ is the largest number, then $\max_k(a) - a_1 \leq k^{-1} \ln \left( \sum \frac{e^{a_i}}{e^{k a_1}} \right) \leq \ln m/k$.

### 3.2.3 Overall approximation

Putting the pieces together yields the approximation error for the robustness of any MTL formula.

**Theorem 3.1.** Consider an MTL formula $\varphi$ and reals $\varepsilon > 0$ and $k \geq 1$. Define the smooth robustness $\tilde{\rho}_\varphi$, obtained by substituting $\text{dist}_\varepsilon$ for $\text{dist}$, $\max_k$ for $\max$, and $\min_k$ for $\min$, in Def. 2.2. Then for any trajectory $x$, it holds that

$$|\rho_\varphi(x, t) - \tilde{\rho}_\varphi(x, t)| \leq \delta_\varphi$$

where $\delta_\varphi$ is (a) independent of the evaluation time $t$, and (b) goes to 0 as $\varepsilon \to 0$ and $k \to \infty$.

**Proof.** We will prove a stronger result that implies the theorem. When $x$ or $t$ are clear from the context, we will drop them from the notation.

The proof is by structural induction on $\varphi$, and works by carefully characterizing the approximation error.

Case $\varphi = p \in \text{AP}$. $\rho_\varphi(x, t)$ is given by either $\text{dist}_\varepsilon(x_t, O(p))$ or $-\text{dist}_\varepsilon(x_t, O(p))$, and $\tilde{\rho}_\varphi(x, t)$ is given by either $\text{dist}_\varepsilon(x_t, O(p))$ or $-\text{dist}_\varepsilon(x_t, O(p))$, respectively. Either way, $|\tilde{\rho}_\varphi(x, t) - \rho_\varphi(x, t)| \leq \varepsilon$. Indeed, $\varepsilon$ satisfies the conditions on $\delta_\varphi$.

Case $\varphi = \neg \varphi_1$. $|\rho_\varphi(x, t) - \tilde{\rho}_\varphi(x, t)| = |\rho_\varphi(x, t) + \tilde{\rho}_\varphi(x, t)| \leq \delta_\varphi_1$, and $\delta_\varphi_1$ satisfies (a)-(b) by the induction hypothesis.

Case $\varphi = \varphi_1 \lor \varphi_2$. If the same sub-formula $\varphi_i$ achieves the max for both $\rho_\varphi(x, t) \cup \rho_\varphi_2(x, t)$ and $\tilde{\rho}_\varphi(x, t) \cup \tilde{\rho}_\varphi_2(x, t)$, then by induction hypothesis we immediately obtain $|\rho_\varphi(x, t) - \tilde{\rho}(x, t)| \leq \delta_\varphi_i$.

Otherwise if, say, $\rho_\varphi = \rho_\varphi_1$ and $\tilde{\rho}_\varphi = \tilde{\rho}_\varphi_2$ then

$$\rho_\varphi_1 - \delta_\varphi_1 \leq \tilde{\rho}_\varphi_1 \leq \tilde{\rho}_\varphi_2 \implies \rho_\varphi_1 - \tilde{\rho}_\varphi_2 \leq \delta_\varphi_1$$

Also

$$\tilde{\rho}_\varphi_2 \leq \rho_\varphi_2 + \delta_\varphi_2 \leq \rho_\varphi_1 + \delta_\varphi_2 \implies -\delta_\varphi_2 \leq \rho_\varphi_1 - \tilde{\rho}_\varphi_2$$

Therefore

$$-(\delta_\varphi_1 \cup \delta_\varphi_2) \leq \rho_\varphi_1 - \tilde{\rho}_\varphi_2 \leq \delta_\varphi_1 \cup \delta_\varphi_2$$
Similarly, if $\rho_\varphi = \rho_{\varphi_2}$ and $\tilde{\rho}_\varphi = \tilde{\rho}_{\varphi_1}$, we have $|\rho_{\varphi_2} - \tilde{\rho}_{\varphi_1}| \leq \delta_{\varphi_1} \sqcup \delta_{\varphi_2}$. So in all cases,

$$|\rho_{\varphi_1} \sqcup \rho_{\varphi_2} - \tilde{\rho}_{\varphi_1} \sqcup \tilde{\rho}_{\varphi_2}| \leq \delta_{\varphi_1} \sqcup \delta_{\varphi_2}$$

Therefore by the triangle inequality and $(3.5)$

$$|\rho_{\varphi_1} \sqcup \rho_{\varphi_2} - \tilde{\max}_k(\tilde{\rho}_{\varphi_1}, \tilde{\rho}_{\varphi_2})| \leq \delta_{\varphi_1} \sqcup \delta_{\varphi_2} + \ln(2)/k = \delta_{\varphi}$$

Clearly, $\delta_{\varphi}$ satisfied (a)-(b).

The case $\varphi_1 \wedge \varphi_2$ is treated similarly.

$\varphi = \varphi_1 \cup \varphi_2$. Before proving this case, we will need the following lemma, which is provable by induction on $n$:

**Lemma 1.** If $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_n$ or $\varphi = \varphi_1 \lor \ldots \lor \varphi_n$, $n \geq 2$, then $|\rho_{\varphi} - \tilde{\rho}_{\varphi}| \leq \sqcup_{1 \leq i \leq n} \delta_{\varphi_i} + \ln(n)/k$.

We now proceed with the proof of the last case. Recall that $\rho_{\varphi_1 \cup \varphi_2}(x, t) = \sqcup_{t' \in t + T_{\varphi_2}} (\rho_{\varphi_2}(x, t')) \sqcap \sqcap_{t'' \in [t, t')} \rho_{\varphi_1}(x, t'')$. Starting with the innermost sub expression $\rho_{\psi} := \sqcap_{t'' \in [t, t')} \rho_{\varphi_1}(x, t'')$, we have, by Lemma 1

$$|\rho_{\psi} - \tilde{\rho}_{\psi}| \leq \sqcup_{t'' \in [t, t')} \delta_{\varphi_1} + \ln(t' - t)/k$$ (3.7)

where $\delta_{\varphi_1}$ is the bound for approximating $\rho_{\varphi_1}(x, t'')$. But $\delta_{\varphi}$ does not depend on the time at which the formula is evaluated. Therefore the bound in (3.7) becomes

$$|\rho_{\psi} - \tilde{\rho}_{\psi}| \leq \delta_{\varphi_1} + \ln(t' - t)/k$$ (3.8)

To avoid introducing a dependence on time, we further upper-bound by

$$|\rho_{\psi} - \tilde{\rho}_{\psi}| \leq \delta_{\varphi_1} + \ln(hrz(\varphi))/k := \delta_{\psi}$$

where, recall, $hrz(\varphi)$ is the horizon of $\varphi$ (see Section 2.1).

Continuing with the sub-expression $\rho_{\alpha} = \rho_{\varphi_2}(x, t') \sqcap \rho_{\psi}$, by the induction hypothesis it holds that $|\rho_{\alpha} - \tilde{\rho}_{\alpha}| \leq \delta_{\varphi_2} \sqcup \delta_{\psi} + \ln(2)/k := \delta_{\alpha}$. Finally, the top-most max operator introduces the total error

$$|\rho_{\varphi} - \tilde{\rho}_{\varphi}| \leq \delta_{\alpha} + \ln(|I|)/k$$

$$= \delta_{\varphi_2} \sqcup \delta_{\psi} + \ln(2)/k + \ln(|I|)/k$$

$$= \delta_{\varphi_2} \sqcup (\delta_{\varphi_1} + \ln(hrz(\varphi))/k) + \ln(2|I|)/k$$

$$= \delta_{\varphi}$$ (3.9)

The first inequality obtains from the fact that $\delta_{\alpha}$ is independent of evaluation time and Lemma 1. The bound $\delta_{\varphi}$ obeys (a)-(b). This concludes the proof.
3.2.4 The need for smoothing

The application of gradient descent methods requires a differentiable objective function. Our objective function, \( \rho_\varphi \), is non-differentiable, because it uses the distance, max, and min functions, all of which are non-differentiable. One may note that these functions are all differentiable almost everywhere (a.e.) on their domain. That is, the set of points in their domain where they are non-differentiable has measure 0 in \( \mathbb{R}^n \). Therefore, by measure additivity, the composite function \( \rho_\varphi \) is itself differentiable a.e. Thus, one may be tempted to ‘ignore’ the singularities (points of non-differentiability), and apply gradient descent to \( \rho_\varphi \) anyway. The rationale for doing so is that sets of measure 0 are unlikely to be visited by gradient descent, and thus don’t matter. However, as we show in the next example, the lines of singularity (along which the objective is non-differentiable) can be precisely the lines along which the objective increases the fastest. See also Cortes [2008]. Thus they are consistently visited by gradient descent, after which it fails to converge because of the lack of a gradient.

Example 3. A simple example illustrates how gradient descent gets stuck at singularities. We use the optimization algorithm Sequential Quadratic Programming (SQP) [Polak 1997] to maximize the robustness of \( \varphi = \neg(x \in U) \), where \( U = [-1,1]^2 \) is the unsafe red square in Fig. 3.1. In this case, \( \rho_\varphi \) is simply \( \text{dist}(x_0, U) \), the distance of the first trajectory point to the set. The search space is \([-2.5,2.5]^2 \) (big grey square in Fig. 3.1). The most robust points are the corners of the grey square, such as \( x^* = [2.5,2.5] \) (green ‘+’ in figure), being furthest from the unsafe set. We initialize the SQP at \( x_0 = [0,0] \). SQP generates iterates (blue circles) on the line of singularity connecting \([1,1]\) to \( x^* \) and ultimately gets stuck at \( x = [1,1] \). That’s because along the line, the gradient does not exist and attempts by SQP to approximate it numerically fail, prompting it to generate smaller and smaller step-sizes for the approximation. Ultimately, SQP aborts due to the step-size being too small, and concludes it is at a local minimum.

3.3 Approximation and control

We implemented the smooth approximation to the semantics of MTL, and tested it on several examples.

3.3.1 Approximation error

We evaluated the robustness \( \rho_\varphi \) and its approximation \( \tilde{\rho}_\varphi \) for five formulae. The horizon \( N \) of each formula is varied, and at each value of \( N \) we generate 1000 trajectories of system \( x_{k+1} = x_k + u_k \) with input and state saturation, by feeding it random input sequences. Fig. 3.2 shows the Root Mean Square (RMSE) of the approximation, \( \sqrt{(1/1000)\sum_x (\rho_\varphi(x) - \tilde{\rho}_\varphi(x))^2} \), and variance bars around it. As seen, the approximation errors and their variances are small, showing the accuracy and stability of
the smooth approximation. Note that while the RMSE increased with the system dimension (4th formula in Fig. 3.2), it was observed that the relative error remained very small i.e. the increase in error is explained by an increase in the robustness’s magnitude.

3.3.2 Robustness maximization for control

Problem $P_\rho$ given in (3.1) is solved by replacing the true robustness $\rho_\varphi$ by its smooth approximation $\tilde{\rho}_\varphi$, and setting $\epsilon_{\min}$ to the value of the smooth approximation error. We thus obtain Problem $P_{\tilde{\rho}}$. This approach is labeled Smooth Operator (SOP).

Optimization solver. Problem $P_{\tilde{\rho}}$ is solved using Sequential Quadratic Programming (SQP). SQP solves constrained non-linear optimization problems by creating a sequence of quadratic approximations to the problem and solving these approximate problems. SQP enjoys various convergence-to-(local)-optima properties [Polak 1997, Section 2.9]. For example, for SQP to converge to a strict local minimum (a minimum that is strictly smaller than any objective function value in an open neighborhood around it), it suffices that 1) all constraint functions be twice Lipschitz continuously differentiable. In our case, this includes function $f$ in (3.1a), and the problems we solve satisfy this requirement. And, 2) at points in the search space that
lie on the boundary of the inequality-feasible set there exists a search direction towards the interior of the feasible set that does not violate the equality constraints [Polak 1997, Assumption 2.9.1]. This is also true for $\tilde{\rho}$ since the equality constraints come from the dynamics and are always enforced for any $u$.

**Solver initialization.** To initialize SQP when solving $P_\tilde{\rho}$ (i.e., to give it a starting value for $u$), we can either solve an inexpensive feasibility linear program with constraints (3.1b)-(3.1d), or generate a random input sequence respecting $u_t \in U$. The resulting initial trajectory could violate the specification (as it does in every example we study here) and it is only required to satisfy the dynamics and state constraints.

**Comparisons to BluSTL.** The tool BluSTL implements the MILP approach of Raman et al. [2014] and is used in the experiments. It has two modes of operation: mode (B) or *Boolean*, which aims at satisfying the specification without maximizing its robustness, and mode (R) or *Robust*, which attempts to maximize robustness. The proposed SOP method optimizes robustness and so naturally runs in mode (R). SOP emulates mode (B) by terminating the optimization as soon as $\tilde{\rho}_{\varphi} \geq \epsilon_{\text{Meyer}}$, which implies $\rho_{\varphi} \geq 0$. $\epsilon_{\text{Meyer}}$ can be computed explicitly using the approach in the online report Pant et al. [2017a].

![Figure 3.2: Robustness approximation error against formula horizon, evaluated on 1000 randomly generated trajectories for Example 4. Unless noted, the systems are 2D. Color in online version.](image-url)
Table 3.1: Runtimes (mean and standard deviation, in seconds) for Smooth Operator (SOP) and BluSTL (BlS) in modes (B) and (R), over 100 runs with random initial states and different formula horizons $N$. BluSTL(R) did not finish (see text).

<table>
<thead>
<tr>
<th>$N$</th>
<th>BlS(B)</th>
<th>SOP(B)</th>
<th>BlS(R)</th>
<th>SOP(R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.96 ± 0.82</td>
<td>0.31 ± 0.13</td>
<td>NA</td>
<td>3.30 ± 1.25</td>
</tr>
<tr>
<td>30</td>
<td>1.37 ± 1.72</td>
<td>0.33 ± 0.25</td>
<td>NA</td>
<td>5.85 ± 2.74</td>
</tr>
<tr>
<td>40</td>
<td>3.86 ± 5.10</td>
<td>0.60 ± 0.29</td>
<td>NA</td>
<td>12.36 ± 6.04</td>
</tr>
<tr>
<td>50</td>
<td>4.36 ± 12.97</td>
<td>0.74 ± 0.30</td>
<td>NA</td>
<td>30.05 ± 18.23</td>
</tr>
<tr>
<td>100</td>
<td>16.77 ± 27.84</td>
<td>1.21 ± 0.25</td>
<td>NA</td>
<td>69.70 ± 13.16</td>
</tr>
<tr>
<td>200</td>
<td>53.88 ± 14.18</td>
<td>4.19 ± 1.18</td>
<td>NA</td>
<td>126.11 ± 20.43</td>
</tr>
</tbody>
</table>

Example 4. The linear system $x_{k+1} = x_k + u_k$ is controlled to satisfy the specification

$$\varphi = \Box_{[0,20]}(x \in \text{Unsafe}) \land \Diamond_{[0,20]}(x \in \text{Terminal})$$

with the sets Unsafe $= [-1,1]^2$ and Terminal $= [2,2.5]^2$. The state space is $X = [-2.5,2.5]^2$, $U = [-0.5,0.5]^2$. Unless otherwise indicated, $\gamma = \delta = 0$ in Eq. (3.1) to focus on robustness maximization in this illustrative example. Experiments were run on a quad-core Intel i5 3.2GHz processor with 24GB RAM, running MATLAB 2016b.

**Results.** Fig. 3.3 shows the trajectories of length $N = 20$ obtained by SOP and BluSTL in modes (B) and (R), starting from the same initial point $x_0 = [-2,-2]'$. Both BluSTL(B) and SOP(B) produce satisfying trajectories. The trajectory from SOP(B) ends in the middle of the terminal set, resulting in a higher robustness than mode (B), as expected. In mode (R), BluSTL could not finish a single instance of robustness maximization within 100 hours on both the above machine and on a more powerful 8 core Intel Xeon machine with 60GB RAM, leading us to believe that the corresponding MILP was not tractable.

SOP(R, $\gamma=0.1$) takes into account a control cost $l(x_k, u_k) = ||x_k||^2_2$ that penalizes longer trajectories. The resulting trajectory is shorter but has a lower robustness than SOP(R, $\gamma = 0$), (0.236 vs 0.247).

For further evaluation, we ran 100 instances of the problem, varying the trajectory’s initial state in $[-2.5,-1.5] \times [-2.5,2.5]$. We also varied the formula horizon $N$ (and hence the size of the problem) from 20 to 200 time steps. Table 3.1 shows the execution times.

**Analysis.** As seen in Table 3.1, SOP is consistently faster than BluSTL in Boolean mode, and displays smaller variances in runtimes. Note also that the problem solved here is very similar to the one used in Saha and Julius [2016], which uses another MILP-based method. While the underlying dynamics differ and their numbers are reported on a more powerful machine, SOP numbers compare favourably with those in Saha and Julius [2016].

In (R) mode, across 100 experiments, SOP has an average $\rho_\varphi = 0.247$ with a standard deviation less than 0.005. This gets very close to the upper bound on robustness,
Figure 3.3: The first 4 trajectories are for Example 4. The last trajectory, SOP(R, unicycle), is from Example 5. Colors in online version.

which is 0.25. This bound is achieved by a trajectory reaching (in < 20 time steps) the center of the Terminal set while remaining more than 0.25 distant from Unsafe.

Example 5 (Nonlinear system). Since SQP can handle non-linear (twice differentiable) constraints, Smooth Operator can also deal with non-linear dynamics whereas the MILP-based methods have to linearize the dynamics to solve the system. The following example shows SOP applied in a one-shot manner to the unicycle dynamics $(\dot{x}_t = v_t \cos(\theta_t), \dot{y}_t = v_t \sin(\theta_t), \dot{\theta}_t = u_t)$ discretized at 10Hz. For the specification of Ex. 4, the resulting trajectory of length 20 steps obtained by SOP(R) is shown in Fig. 3.3, starting from an initial state of $[-2, -2, 0]$. The resulting robustness is 0.248, which is close to the global optimum of 0.25. This shows that SOP can indeed handle non-linear dynamics without the need for explicit linearization as long as the systems satisfy assumption 3.1.

3.4 Case studies

This section focuses on evaluating the efficiency of Smooth Operator (SOP) by testing it on two systems and comparing to existing approaches.
• SOP in (B)oolean and (R)obust modes.
• BluSTL in modes (B) and (R).
• R-SQP, which uses SQP to optimize the exact non-smoothed robustness $\rho_\varphi$.
• SA, which uses Simulated Annealing to optimize $\rho_\varphi$.

For both case studies, the wavelet approximation to the distance function is computed off-line. The control problem \[3.1\] is solved as an open-loop, single-shot, finite-horizon constrained optimization. This is then used in a shrinking horizon scheme in Sec 3.4.1.

The code to reproduce these results can be found at https://github.com/yashpant/SmoothOperator0. Future versions of the code will focus on re-usability of the code.

3.4.1 HVAC Control of a building for comfort

The first example is the Heating, Ventilation and Air Conditioning (HVAC) control of a 4-state model of a single zone in a building. Such a model is commonly used in literature for evaluation of predictive control algorithms Jain et al. [2017]. The control problem is similar to the example used in Raman et al. [2014]. The control horizon is a 24 hour period. The objective is to bring the zone temperature to a comfortable range, $[22, 28]$ Celsius, when the zone is occupied during the hours 10-to-19. The specification is:

$$\varphi = \Box_{[10, 19]}(\text{ZoneTemp} \in [22, 28]) \quad (3.10)$$

Note: For this particular specification, the maximum robustness is 3, achievable by setting the room temperature at 25C during the interval [10, 19]. Thus the problem can be solved by minimizing the cost $\sum_{10 \leq k < 19} (x_{4k} - 25)^2$ with linear constraints, which is a problem-specific approach. SOP, which is a general purpose technique, results in a robustness which is just 0.02% smaller than the global optimum. Section 3.4.2 shows a specification that cannot be trivially turned into a quadratic program.

System dynamics. The single-zone model, discretized at a sampling rate of 1 hour (which is common in building temperature control) is of the form:

$$x_{k+1} = Ax_k + Bu_k + B_d d_k \quad (3.11)$$

Here, $A$, $B$ and $B_d$ matrices are from the hamlab ISE model Van Schijndel [2005]. $x \in \mathbb{R}^4$ is the state of the model, the 4th element of which is the zone temperature, the others are auxiliary temperatures corresponding to other physical properties of the zone. The input to the system, $u \in \mathbb{R}^1$, is the heating/cooling energy. $b_d \in \mathbb{R}^3$ are disturbances (due to occupancy, outside temperature, solar radiation) assumed known a priori. The control problem we solve is of the form in \[3.1\], with $\gamma$ and $\delta$ both set to zero (correspondingly, no cost for control in BluSTL), and $X = [0, 50]^4$, $U = [-1000, 2000]$. 

24
Table 3.2: HVAC. Runtimes (mean ± std deviation, in seconds) SOP and BluSTL (BLS) over 100 runs with random initial states.

<table>
<thead>
<tr>
<th></th>
<th>BLS (B)</th>
<th>BLS (R)</th>
<th>SOP (B)</th>
<th>SOP (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.041 ± 0.002</td>
<td>0.622 ± 0.118</td>
<td><strong>0.014 ± 0.002</strong></td>
<td>0.316 ± 0.015</td>
</tr>
</tbody>
</table>

Results. For comparison across all methods, we run 100 instances of the problem, starting from random initial states $x_0 \in [20, 21]^4$. SA, R-SQP and SOP are initialized with the same initial input sequences $u$. The final trajectories after optimization are shown in Fig. 3.4 for $x_0 = [21, 21, 21, 21]'$. To reduce clutter, trajectories from SA and R-SQP in mode (B) are not shown.

Analysis. In Boolean mode, SOP, BluSTL, and SA all find satisfying trajectories across all 100 instances, while R-SQP does not find one for any run and always exits at a local minimum. Execution times for SOP and BluSTL are shown in Table 3.2 while the runtimes for SA (B) are $3.7 ± 2.3s$. R-SQP has run-times in the order of minutes.

In Robust mode SOP, BLS and SA result in trajectories that satisfy $\varphi$, with an average robustness of 2.99, 3.0 and 2.88 respectively. On the other hand, R-SQP often returns violating trajectories (average $\rho_{\varphi} = -0.1492$). Runtimes for SOP and BLS in Robust mode are shown in Table 3.2 SA(R) takes $8.56 \pm 0.31s$ on average.

Shrinking horizon implementation

SOP can also be applied online in a shrinking horizon fashion similar to Raman et al. [2014]. The control horizon in Eq. (3.1) equals the formula horizon $N$. For each time step $k = 0, \ldots, N$, problem (3.1) is solved while constraining the previously applied inputs and states (for times steps $< k$) to their actual values. In this scheme, the length of the optimization remains $N$, but the number of free variables keeps on shrinking as $k$ increases. For initializing SOP at each time step, the sequence of inputs computed at time $k − 1$ is used as a feasible solution for the optimization at time $k$. We implemented this scheme for the HVAC control problem with additional unknown disturbances in $d_k$ term of Eq. (3.11). These disturbances (in $\mathbb{R}^3$) are uniform random variables centred around the known $d_k$ with an interval of 10% of element wise magnitude of $d_k$. This can be thought of as prediction errors of upto $±10\%$ in the disturbances like solar radiation and outside temperature which make up $d_k$ in Eq. (3.11). Over 100 runs with random initial states as before, the online application of SOP (in robust mode) resulted in an average robustness value of 2.91. In terms of execution time, the first iteration takes times of the order of those in table 3.2 and subsequent iterations take a fraction of that time (average for one instance 0.0151s). This is because we re-use the input sequence at time $k − 1$ as an initial guess for the solver at time $k$. Since at the initial time step we have achieved near global robustness maxima, the subsequent SQP optimizations terminate much faster while the formulation takes into account change in the state due to disturbance values.
by making small changes to the input sequence being computed at time $k > 0$. The high value of average robustness and the small execution time per iteration show the applicability of SOP as an online closed loop control method.

3.4.2 Autonomous ATC for quad-rotors

Air Traffic Control (ATC) offers many opportunities for automation to allow safer and more efficient landing patterns. The constraints of ATC are complex and contain many safety rules [Li and Ryerson 2016]. In this example we formalize a subset of such rules, similar to those in example 2, for an autonomous ATC for quad-rotors in MTL. We demonstrate how the smoothed robustness is used to generate control strategies for safely and robustly manoeuvring two quad-rotors in an enclosed airspace with an obstacle.

The specification. The specification for the autonomous ATC with two quad-
rotors is:

\[
\varphi = \Diamond_{[0,N-1]}(q_1 \in \text{Terminal}) \land \Diamond_{[0,N-1]}(q_2 \in \text{Terminal}) \land \\
\Box_{[0,N-1]}(q_1 \in \text{Zone}_1 \implies z_1 \in [1,5]) \land \\
\Box_{[0,N-1]}(q_2 \in \text{Zone}_1 \implies z_2 \in [1,5]) \land \\
\Box_{[0,N-1]}(q_1 \in \text{Zone}_2 \implies z_1 \in [0,3]) \land \\
\Box_{[0,N-1]}(q_2 \in \text{Zone}_2 \implies z_2 \in [0,3]) \land \\
\Box_{[0,N-1]}(\neg (q_1 \in \text{Unsafe})) \land \Box_{[0,N-1]}(\neg (q_2 \in \text{Unsafe})) \land \\
\Box_{[0,N-1]}(||q_1 - q_2||_2^2 \geq d_{\text{min}}^2) \tag{3.12a}
\]

Here \(q_1\) and \(q_2\) refer to the position of the two quad-rotors in \((x,y,z)\)-space, and \(z_1\) and \(z_2\) refer to their altitude. The specification says that, within a horizon of \(N\) steps, both quad-rotors should: a) Eventually visit the terminal zone (e.g. to refuel or drop package), b) Follow altitude rules in two zones, \(\text{Zone}_1\) and \(\text{Zone}_2\) which have different altitude floors and ceilings, c) Avoid the Unsafe set, and d) always maintain a safe distance between each other (\(d_{\text{min}}\)).

Note that turning the specification into constraints for the control problem is no longer simple. This is due to the \(\Diamond\) operator, which would require a MILP formulation to be accounted for. In addition, the minimum separation and altitude rules for the two zones cannot be turned into convex constraints for the optimization. As will be seen below, our approach allows us to keep the non-convexity in the cost function, and have convex (linear) constraints on the optimization problem.

**System dynamics.** The airspace and associated sets for the specification \(\varphi\) are hyper-rectangles in \(\mathbb{R}^3\) (visualized in Fig. 3.5), except the altitude floor and ceiling limit, which is in \(\mathbb{R}^1\). In simulation, \(d_{\text{min}}\) is set to 0.2 m.

The quad-rotor dynamics are obtained via linearization around hover, and discretization at 5-Hz. Similar models have been used for control of real quad-rotors with success \cite{Pant2015a}. For simulation, we set the mass of either quad-rotor to be 0.5 kg. The corresponding linearized and discretized quad-rotor dynamics are given in \cite{Pant2017a}. The state for a quad-rotor \(x \in \mathbb{R}^6\) consists of the velocities and positions in the \(x, y, z\) co-ordinates respectively. The inputs to the system are the desired roll angle \(\theta\), pitch angle \(\phi\) and thrust \(T\).

**The control problem.** For the autonomous ATC problem for two quad-rotors, we solve (3.1) with \(\hat{\rho}\) in the objective instead of \(\rho\). Note, we set \(\gamma = 0\) here, following logically from existing ATC rules (see sec. 3.1), which do not have an air-craft specific cost for fuel, or distance traveled. Because of this, we can also set \(\delta = 0\) and simply maximize (smooth) robustness (subject to system dynamics and constraints) to get trajectories that satisfy \(\varphi\). For the control problem \((P_{\hat{\rho}})\), \(X\) and \(U\) represent the bounds on the states (Airspace and velocity limits) and inputs respectively, for both quad-rotors. \(f\) represents the linearized dynamics applied to two quad-rotors, and \(N = 21\). The initial state for the first quad-rotor is \([0,0,0,2,2,2]'\) and for the second, \([0,0,0,2, -2,2]'\).

**Results.** For each approach (except BluSTL), we ran three optimizations, start-
ing from three different trajectories to initialize the optimization. This can be thought of as a multi-start optimization, and these initial trajectories can be obtained in practice by a fast trajectory generator. All three initial trajectories have negative robustness, i.e. they violate \( \varphi \). In this case study, we only aim to maximize robustness, i.e. operate in the robust mode. BluSTL, in either boolean or robust mode could not find a solution for this problem (ran over 100 hours without terminating) and so is excluded from the rest of this comparison. This suggests that having a complex specification like the one in this problem, non-trivial dynamics/horizon length results in a MILP that is intractable to solve. We believe that this example highlights a fundamental limitation of MILP based approaches.

Fig. 3.5 shows the three trajectories obtained after applying SOP, all of which satisfy the specification \( \varphi \). To avoid visual clutter, we do not show the trajectories obtained from the other methods on the figure. Instead, we summarize the results in Table 3.3 which shows the true robustness of the three initial trajectories, and the true robustness for the trajectories obtained via the three methods, SOP, SA, and R-SQP. Unlike previous examples, we did not explicitly compute the gradient of the robustness for \( \varphi \). Because of this run-times are much slower as MATLAB has to numerically compute the gradient using finite-differences, resulting in overheads that were not incurred in the other examples. Despite this, SOP takes the order of 30 minutes for the optimization, while SA and R-SQP take over 4 hours to do so. Including explicit gradients should result in a significant speed up as was observed for the other examples.

**Analysis.** It is seen that SOP and R-SQP satisfy \( \varphi \) for all instances, while SA satisfies it only once. Note that in all three cases, R-SQP results in trajectories with the same robustness value, which is less than the robustness value achieved in SOP. We conjecture that this is because R-SQP is getting stuck at local minima at points of non-differentiability of the objective (see Ex. 2 in Pant et al. [2017a]). On further investigation, we also noticed that the robustness value achieved is due to the segment of the \( \varphi \) corresponding to \( \hat{\varphi}_{[0,N]}(q_2 \in \text{Terminal}) \). R-SQP does not drive the trajectory (for quad-rotor 2) deeper inside the set Terminal, unlike the proposed approach, SOP, even though the minimum separation property is far from being violated. This lends credence to our hypothesis of SQP terminating on a local minima, which is the flag MATLAB’s optimization gives.

<table>
<thead>
<tr>
<th>Run</th>
<th>( \rho(x_0) )</th>
<th>SOP: ( \rho^<em>\hat{\rho}^</em> )</th>
<th>SA: ( \rho^* )</th>
<th>R-SQP: ( \rho^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.8803</td>
<td><strong>0.3689</strong> [0.4107]</td>
<td>-0.2424</td>
<td>0.1798</td>
</tr>
<tr>
<td>2</td>
<td>-0.7832</td>
<td><strong>0.3688</strong> [0.4106]</td>
<td>-0.5861</td>
<td>0.1798</td>
</tr>
<tr>
<td>3</td>
<td>-0.0399</td>
<td><strong>0.3689</strong> [0.4107]</td>
<td>0.0854</td>
<td>0.1798</td>
</tr>
</tbody>
</table>
3.5 Discussion and Conclusions

We present a method to obtain smooth (infinity differentiable) approximations to the robustness of MTL formulae, with bounded and asymptotically decaying approximation error. Empirically, we show that the approximation error is indeed small for a variety of commonly used MTL formulae. Through several examples, we show how we leverage the smoothness property of the approximation for solving a control problem by maximizing the smooth robustness, using SQP, an off-the-shelf gradient descent optimization technique. A similar approach can also be used for falsification by minimizing the smooth robustness over a set of possible initial states for a closed loop system. We compare our technique (SOP) to other approaches for robustness maximization for control of two dynamical systems, with state and input constraints, and show how our approach consistently outperforms the other methods. While for
most examples, we solve the control problem in a single-shot, finite horizon manner,
in general, for a real-time implementation, the problem can be solved in an online
manner as in Sec. 3.4.1. Future work will include a C implementation of SOP, which
will allow us to experiment on real platforms, like the aforementioned quad-rotors,
and also expand.
Chapter 4

Fly-by-Logic: Control of Multi-Drone Fleets with Temporal Logic Objectives

4.1 Introduction

In this chapter, we present a method that overcomes some of the limitations of multi-robot planning and control, outlined in Chapter 1. We focus on multi-rotor drones as our dynamical system, and develop a planning method that allows for a wide variety of multi-drone missions that consist of a combination of the following objectives:

1. **Spatial objectives**, e.g. geofenced no fly zones, or delivery zones,
2. **Temporal objectives**, e.g. a time window to monitor wireless signal strengths in an area,
3. **Reactive objectives**, e.g. in case of a drone failure, another drone picks up its mission.

These mission, or behavioral objectives are specified using Signal Temporal Logic (STL), which allows us to express a comprehensive list of objectives for the robots to satisfy.

4.1.1 Contributions

This chapter presents a control framework for mission planning and execution for fleets of multi-rotor UAS, given a STL specification.

1. **Continuous-time STL satisfaction**: We develop a control optimization that selects waypoints by maximizing the robustness of the STL mission. A solution to the optimization is guaranteed to satisfy the mission in continuous-time, so
Figure 4.1: (Top) Five Crazyflie 2.0 quadrotors executing a reach-avoid mission. (Bottom) A screenshot of a simulation with 16 quadrotors. In both cases, the quadrotors have to satisfy a mission given in STL.

trajectory sampling does not jeopardize correctness, while the optimization only works with a few waypoints.

2. **Dynamic feasibility of trajectories**: We demonstrate that the trajectories generated by our controller respect pre-set velocity and acceleration constraints, and so can be well-tracked by lower-level controllers.

3. **Real-time control**: We demonstrate our controller’s suitability for online control by implementing it on real quadrotors and executing a reach-avoid mission.

4. **Performance and scalability**: We demonstrate our controller’s speed and performance on real quadrotors and in simulation.
4.2 Preliminaries and notations used

Consider a continuous-time dynamical system $H$ and its uniformly discretized version

$$\dot{x}_c(t) = f_c(x_c(t), u(t)), \quad x^+ = f(x, u) \quad (4.1)$$

where $x \in X \subset \mathbb{R}^n$ is the current state of the system, $x^+$ is the next state, $u \in U \subset \mathbb{R}^m$ is its control input and $f : X \times U \rightarrow X$ is differentiable in both arguments. The system’s initial state $x_0$ takes values from some initial set $X_0 \subset \mathbb{R}^n$. In this work we deal with trajectories of the same duration (e.g. 5 seconds) but sampled at different rates, so we introduce notation to make the sampling period explicit.

Let $dt \in \mathbb{R}^+\setminus\{0\}$ be a sampling period and $T \in \mathbb{R}^+$ be a trajectory duration. We write $[0 : dt : T] = (0, dt, 2dt, \ldots, (H-1)dt)$ for the sampled time interval s.t. $(H-1)dt = T$ (we assume $T$ is divisible by $H-1$). Given an initial state $x_0$ and a finite control input sequence $u = (u_0, u_1, \ldots, u_{H-2})$, $u_t \in U$, $t_k \in [0 : dt : T]$, a trajectory of the system is the unique sequence of states $x = (x_0, x_1, \ldots, x_{H-1})$ s.t. for all $t \in [0 : dt : T]$, $x_t$ is in $X$ and $x_{t+1} = f(x_t, u_t)$. We also denote such a trajectory by $x[dt]$. Given a time domain $T = [0 : dt : T]$, the signal space $X^T$ is the set of all signals $x : T \rightarrow X$.

For an interval $I \subset \mathbb{R}^+$ and $t \in \mathbb{R}^+$, set $t + I = \{ t + a \mid a \in I \}$. The max operator is written $\sqcup$ and min is written $\sqcap$.

4.3 Control Using a Smooth Approximation of STL Robustness

The goal of this work is to find a provably correct control scheme for fleets of quadrotors, which makes them meet a control objective $\varphi$ expressed in temporal logic. So let $\epsilon > 0$ be a desired minimum robustness. We solve the following problem.

$$P : \max_{u \in U^{N-1}} \rho_\varphi(x) \quad (4.2a)$$

s.t. $x_{k+1} = f(x_k, u_k), \; \forall k = 0, \ldots, N-1$ \hspace{1cm} (4.2b)

$x_k \in X, u_k \in U \; \forall k = 0, \ldots, N$ \hspace{1cm} (4.2c)

$\rho_\varphi(x) \geq \epsilon$ \hspace{1cm} (4.2d)

Because $\rho_\varphi$, the robustness of the STL specification $\varphi$ (see section 2.2.1), uses the non-differentiable functions max and min (see Def. 2.2), it is itself non-differentiable as a function of the trajectory and the control inputs.

Chapter 3 approximates the non-differentiable objective $\rho_\varphi$ by a smooth (infinitely differentiable) function $\tilde{\rho}_\varphi$ and solves the resulting optimization problem $\tilde{P}$ using Sequential Quadratic Programming. The approximate smooth robustness for STL is obtained by using smooth approximations of min and max in Def. 2.2 (also see assumption 2.1). In this chapter, we also use the smoothed robustness $\tilde{\rho}_\varphi$ and solve $\tilde{P}$ instead of $P$. The lower bound on robustness (3.1e) is used to ensure that if $\tilde{\rho}_\varphi \geq \epsilon$
then $\rho_\varphi \geq 0$. In the previous chapter it was shown that an $\epsilon$ can be computed such that $|\rho_\varphi - \tilde{\rho}_\varphi| \leq \epsilon$.

Despite the improved runtime of Smooth Operator (chapter 3), experiments in the previous chapter also show that it is not possible to solve $\tilde{P}$ in real-time using the full quadrotor dynamics. Therefore, in this work, we develop a control architecture that is guaranteed to produce a correct and dynamically feasible quadrotor trajectory. By ‘correct’, we mean that the \textit{continuous-time, non-sampled} trajectory satisfies the formula $\varphi$, and by ‘dynamically feasible’, we mean that it can be implemented by the quadrotor dynamics. This trajectory is then tracked by a lower-level MPC tracker. The control architecture and algorithms are the subject of the next section.

\textbf{Note:} For the method presented in this chapter, the discrete-time state update equations\textsuperscript{[4.2b]} represent the kinematics associated with minimizing the jerk for multi-rotor dynamics Mueller et al.\textsuperscript{[2015]}. The state $x$ and input $u$ are defined accordingly. More details are in section 4.4.2.

### 4.4 Quadrotor Planning and Control Architecture

Fig. 4.2 shows the control architecture used here, and its components are detailed in what follows. The overall idea is that we want the \textit{continuous-time} trajectory $y_c : [0, T] \to X$ of the quadrotor to satisfy the STL mission $\varphi$, but can only compute a discrete-time trajectory $x : [0 : dt : T] \to X$ sampled at a low rate $1/dt$. So we do two things, illustrated in Fig.4.2:

A) to guarantee continuous-time satisfaction from discrete-time satisfaction, we ensure that a discrete-time high-rate trajectory $q : [0 : dt' : T] \to X$ satisfies a suitably \textit{stricter} version $\varphi_s$ of $\varphi$. This is detailed in Section 4.5.

B) To compute, in real-time, a sufficiently high-rate discrete-time $q[dt']$ that satisfies $\varphi_s$, we perform a (smooth) robustness maximization over a low-rate sequence of waypoints $x$ with sampling period $dt >> dt'$. In the experiments (Sections 4.6 and 4.7) we used $dt = 1s$ and $dt' = 50ms$. The optimization problem is such that the optimal low-rate trajectory $x : [0 : dt : T] \to X$ and the desired high-rate $q$ are related analytically: $q = L(x)$ for a known $L : \mathbb{R}^{(T/dt)} \to \mathbb{R}^{(T/dt')}$. So the robustness optimization maximizes $\tilde{\rho}_{\varphi_s}(L(x[dt]))$, automatically yielding $q[dt']$. Moreover, we must add constraints to ensure that $q$ is dynamically feasible, i.e., can be implemented by the quadrotor dynamics. Thus, qualitatively, the optimization problem we solve is

$$\max_{x[dt]} \tilde{\rho}_{\varphi_s}(L(x[dt]))$$

s.t. $L(x[dt])$ obeys quadrotor dynamics and is feasible

- $x$ and $L(x[dt])$ are in the allowed air space
- $\tilde{\rho}_{\varphi_s}(L(x[dt])) \geq \varepsilon$  \hspace{1cm} (4.3)
Figure 4.2: The control architecture. Given a mission specification in STL, the high-level control optimization (centralized) generates a sequence of waypoints. These waypoints are sent over to the drones, and through a hierarchical control on-board control architecture, the resulting trajectories are tracked near perfectly, with the continuous time behavior of the system satisfying the STL specification.

The mathematical formulation of the above problem, including the trajectory generator \( L \), is given in Section 4.4.2. But first, we end this section by a brief description of the position and attitude controllers that take the high-rate \( q[dt'] \) and provide motor forces to the rotors, and a description of the quadrotor dynamics. The state of the quadrotor consists of its 3D position \( p \) and 3D linear velocity \( v = \dot{p} \). A more detailed version of the quad-rotor dynamics is in Section 4.4.1.

**Position controller** To track the desired positions and velocities from the trajectory generator, we consider a Model Predictive Controller (MPC) formulated using the quadrotor dynamics of (4.4) linearized around hover. Given desired position and velocity commands in the fixed-world \( x, y, z \) co-ordinates, the controller outputs a desired thrust, roll, and pitch command (yaw fixed to zero) to the attitude controller. This controller also takes into account bounds on positions, velocities and the desired roll, pitch and thrust commands.

**Attitude controller** Given a desired angular position and thrust command generated by the MPC, the high-bandwidth (50 – 100 Hz) attitude controller maps them to motor forces. In our control architecture, this is implemented as part of the pre-existing firmware on board the Crazyflie 2.0 quadrotores. An example of an attitude
controller can be found in [Luukkonen, 2011].

4.4.1 Introduction to quadrotor dynamics

Multi-rotor dynamics have been studied extensively in the literature [Luukkonen, 2011, Mueller et al., 2015], and we closely follow the conventions of [Mueller et al., 2015]. Fig. 4.3 illustrates the following definitions. The quadrotor has 6 degrees of freedom. The first three, \((x, y, z)\), are the linear position of the quadrotor in \(\mathbb{R}^3\) expressed in the world frame. We write \(p = (x, y, z)\) for position. The remaining three are the rotation angles \((\phi, \theta, \psi)\) of the quadrotor body frame with respect to the fixed world frame. Their first time-derivatives, \(\omega_1, \omega_2, \omega_3\), resp., are the quadrotor’s angular velocities. We also write \(v = \dot{p}\) for linear velocity and \(a = \dot{v}\) for acceleration. If we let \(R\) denote the rotation matrix [Luukkonen, 2011] that maps the quadrotor frame to the world frame at time \(t\), \(e_3 = [0, 0, 1]'\), and \(h \in \mathbb{R}\) be the input to the system, which is the total thrust normalized by the mass of the quadrotor, then the dynamics are given by

\[
\ddot{p} = Re_3 h + [0, 0, 9.81]
\]

\[
\dot{R} = R \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}
\]

(4.4)

4.4.2 The trajectory generator

The mapping \(L\) between low-rate \(x[dt]\) and high-rate \(y[dt']\) is implemented by the following trajectory generator, adapted from [Mueller et al., 2015]. It takes in a motion duration \(T_f > 0\) and a pair of position, velocity and acceleration tuples, called waypoints: an initial waypoint \(q_0 = (p_0, v_0, a_0)\) and a final waypoint \(q_f = (p_f, v_f, a_f)\). It produces a continuous-time minimum-jerk (time derivative of acceleration) trajectory \(q(t) = (p(t), v(t), a(t))\) of duration \(T_f\) s.t. \(q(0) = q_0\) and \(q(T_f) = q_f\). In our control
architecture, the waypoints are the elements of the low-rate $x$ computed by solving (4.3). The generator of Mueller et al. [2015] formulates the quadrotor dynamics in terms of 3D jerk and this allows a decoupling of the equations along three orthogonal jerk axes. By solving three independent optimal control problems, one along each axis, it obtains three minimum-jerk trajectories, each being a spline $q^*: [0, T_f] \to \mathbb{R}^3$ of the form:

$$\begin{bmatrix}
    p^*(t) \\
    v^*(t) \\
    a^*(t)
\end{bmatrix} = \begin{bmatrix}
    \alpha t^5 + \frac{\beta}{24} t^4 + \frac{\gamma}{2} t^2 + a_0 t + v_0 \\
    \frac{\alpha}{24} t^4 + \frac{\beta}{6} t^3 + \frac{\gamma}{2} t^2 + a_0 + v_0 \\
    \frac{\alpha}{6} t^3 + \frac{\beta}{2} t^2 + \gamma t + a_0
  \end{bmatrix}$$

(4.5)

Here, $\alpha$, $\beta$, and $\gamma$ are scalar linear functions of the initial $q_0$ and final $q_f$. Their exact expressions depend on the desired type of motion:

1. **Stop-and-go motion.** [Mueller et al. 2015] This type of motion yields straight-line position trajectories $p(\cdot)$. These are suitable for navigating tight spaces, since we know exactly the robot’s path between waypoints. For Stop-and-Go, the quadrotor starts from rest and ends at rest: $v_0 = a_0 = v_f = a_f = 0$. I.e. the quadrotor has to come to a complete stop at each waypoint. In this case, the constants are defined as follows:

$$\begin{bmatrix}
    \alpha \\
    \beta \\
    \gamma
\end{bmatrix} = \frac{1}{T_f^5} \begin{bmatrix}
    720(p_f - p_0) \\
    -360T_f(p_f - p_0) \\
    60T_f^2(p_f - p_0)
  \end{bmatrix}$$

(4.6)

2. **Trajectories with free endpoint velocities.** [Mueller et al. 2015] Stop-and-go trajectories have limited reach, since the robot must spend part of the time coming to a full stop at every waypoint. In order to get better reach, the other case from [Mueller et al. 2015] that we consider is when the desired initial and endpoint velocities, $v_0$ and $v_f$, are free. Like the previous case, we still assume $a_0 = a_f = 0$. The constants in the spline (4.5) are then:

$$\begin{bmatrix}
    \alpha \\
    \beta \\
    \gamma
\end{bmatrix} = \frac{1}{T_f^5} \begin{bmatrix}
    90 -15T_f^2 \\
    90T_f -15T_f^3 \\
    30T_f^2 -3T_f^4
  \end{bmatrix} \begin{bmatrix}
    p_f - p_0 - v_0 T_f \\
    a_f - a_0
  \end{bmatrix}$$

(4.7)

In this case, the trajectories between waypoints are not restricted to be on a line, allowing for a wider range of maneuvers, as will be demonstrated in the simulations of Section 4.6. An example of such a spline (planar) is shown in Fig. 4.4.

### 4.4.3 Constraints for dynamically feasible trajectories

The splines (4.5) that define the trajectories come from solving an unconstrained optimal control problem, so they are not guaranteed to respect any state and input constraints, and thus might not be dynamically feasible. By dynamically feasible, we mean that the quadrotor can be actuated (by the motion controller) to follow the spline. Typically, feasibility requires that the spline velocity and acceleration be
within certain bounds. E.g. a sharp turn is not possible at high speed, but can be done at low speed. Therefore, we formally define dynamic feasibility as follows.

**Definition 4.1** (Dynamically feasible trajectories). Let \([v, \bar{v}]\) be bounds on velocity and \([a, \bar{a}]\) be bounds on acceleration. A trajectory \(q : [0, T_f] \rightarrow \mathbb{R}^3\), with \(q(t) = (p(t), v(t), a(t))\), is dynamically feasible if \(v(t) \in [v, \bar{v}]\) and \(a(t) \in [a, \bar{a}]\) for all \(t \in [0, T_f]\) for each of the three axes of motion.

**Assumption 4.1.** We assume that dynamically feasible trajectories, as defined here, can be tracked almost perfectly by the position (and attitude) controller. This assumption is validated by our experiments on physical quadrotor platforms. See Section 4.7.

In this section we derive constraints on the desired end state \((p_f, v_f, a_f)\) such that the resulting trajectory \(q(\cdot)\) computed by the generator [Mueller et al., 2015] is dynamically feasible.

Since the trajectory generator works independently on each jerk axis, we derive constraints for a one-axis spline given by (4.5). An identical analysis applies to the splines of other axes. Since a quadrotor can achieve the same velocities and
accelerations in either direction along an axis, we take \( v < 0 < \bar{v} = -v \) and \( a < 0 < \bar{a} = -a \). We derive the bounds for the two types of motion described earlier.

**Stop-and-go trajectories:** \( \mathbf{v}_T = \mathbf{v}_0 = 0 = \mathbf{a}_T = \mathbf{a}_0 = 0 \) Since the expressions for the splines are linear in \( p_f \) and \( p_0 \) (4.5), (4.6), without loss of generality we assume \( p_0 = 0 \). By substituting (4.6) in (4.5), we get:

\[
\begin{align*}
p^*_f &= (6 \frac{t^5}{T_f} - 15 \frac{t^4}{T_f} + 20 \frac{t^3}{T_f^2}) p_f \\
v^*_f &= (30 \frac{t^4}{T_f} - 60 \frac{t^3}{T_f^2} + 30 \frac{t^2}{T_f^3}) p_f \\
a^*_f &= (120 \frac{t^3}{T_f} - 180 \frac{t^2}{T_f^2} + 60 \frac{t}{T_f^3}) p_f
\end{align*}
\]

(4.8)

Fig. 4.6 shows the functions \( K_1 \) and \( K_2 \) for \( T_f = 1 \). The following lemma is proved by examining the first two derivatives of \( K_1 \) and \( K_2 \).

**Lemma 2.** The function \( K_1 : [0, T_f] \rightarrow \mathbb{R} \) is non-negative and log-concave. The function \( K_2 : [0, T_f] \rightarrow \mathbb{R} \) is anti-symmetric around \( t = T_f/2 \), concave on the interval \( t \in [0, T_f/2] \) and convex on the interval \( [T_f/2, T_f] \).

Let \( \max_{t \in [0, T_f]} K_1(t) = K_1^* \) and \( \max_{t \in [0, T_f]} |K_2(t)| = K_2^* \). These are easily computed thanks to Lemma 2. We can now state the feasibility constraints for Stop-and-Go motion. See section 4.4.4 for a proof sketch.

**Theorem 4.1** (Stop-and-go feasibility). Given an initial position \( p_0 \) (and \( v_0 = a_0 = 0 \)), a maneuver duration \( T_f \), and desired bounds \( [v, \bar{v}] \) and \( [a, \bar{a}] \), if \( v/K_1^* \leq p_f - p_0 \leq \bar{v}/K_1^* \) and \( a/K_2^* \leq p_f - p_0 \leq \bar{a}/K_2^* \) then \( v^*_f \in [v, \bar{v}] \) and \( a^*_f \in [a, \bar{a}] \) for all \( t \in [0, T_f] \).

Since \( v, \bar{v}, a, \bar{a}, K_1^*, K_2^* \) are all available offline, they can be used as constraints if solving problem (4.3) offline.

**Free end velocities:** \( a_T = a_0 = 0 \), free \( v_f \). Here too, without loss of generality \( p_0 = 0 \). Substituting (4.7) in (4.5) and re-arranging terms yields the following expression for the optimal translational state:

\[
\begin{align*}
p^*_f &= (90t^5 - 90t^4 + 30t^3) p_f - (90t^5 - 90t^4 + 30t^3 - t) v_0 \\
v^*_f &= (90t^4 - 90t^3 + 30t^2) p_f - (90t^4 - 90t^3 + 30t^2 - 1)v_0 \\
a^*_f &= (90t^3 - 90t^2 + 30t) p_f - (90t^3 - 90t^2 + 30t - 1)v_0
\end{align*}
\]

(4.9)
Figure 4.5: The upper and lower bounds on $p_f$ due to the acceleration and velocity constraints. Shown as a function of $v_0$ for $t = 0, 0.1, \ldots, T_f = 1$. The shaded region shows the feasible values of $p_f$ as a function of $v_0$.

Applying the velocity and acceleration bounds $v \leq v^* \leq \bar{v}$ and $a \leq a^* \leq \bar{a}$ to (4.9) and re-arranging terms yields:

$$\frac{(v - (1 - T_f K_3(t))v_0)}{K_3(t)} \leq p_f \leq \frac{(\bar{v} - (1 - T_f K_3(t))v_0)}{K_3(t)} \quad \forall t \in [0, T_f] \tag{4.10a}$$

$$\frac{a}{K_4(t)} + T_f v_0 \leq p_f \leq \frac{\bar{a}}{K_4(t)} + T_f v_0 \quad \forall t \in [0, T_f] \tag{4.10b}$$

The constraints on $p_f$ are linear in $v_0$, but parametrized by functions of $t$. Since $t$ is continuous in $[0, T_f]$, (4.10) is an infinite system of linear inequalities. Fig. 4.5 shows these linear bounds for $t = 0, 0.1, 0.2, \ldots, 1 = T_f$ with $\bar{v} = 1 = -\underline{v}, \bar{a} = 2 = -\underline{a}$. Fig. 4.6 shows the functions $K_3$ and $K_4$ for $T_f = 1$.

The infinite system can be reduced to 2 inequalities only, as proved in section 4.4.4.

Lemma 3. $p_f$ satisfies (4.10) if it satisfies the following
Theorem 4.2 (Free endpoint velocity feasibility). Given an initial translational state \( p_0, v_0 \in [\underline{v}, \bar{v}], a_0 = 0 \), and a maneuver duration \( T_f \), if \( p_f \) satisfies

\[
\frac{v - (1 - T_f K_3(T_f))v_0}{K_3(T_f)} \leq p_f \leq \frac{\bar{v} - (1 - T_f K_3(T_f))v_0}{K_3(T_f)}
\]

\[
T_f v_0 + \frac{a}{K_4(t')} \leq p_f \leq T_f v_0 + \bar{a}/K_4(t')
\]

(4.11)

where \( t' \) is a solution of the quadratic equation \( \frac{dK_3(t)}{dt} = 0 \), such that \( t' \in [0, T_f] \).

The main result follows:

**Theorem 4.2** (Free endpoint velocity feasibility). Given an initial translational state \( p_0, v_0 \in [\underline{v}, \bar{v}], a_0 = 0 \), and a maneuver duration \( T_f \), if \( p_f \) satisfies

\[
\frac{v - (1 - T_f K_3(T_f))v_0}{K_3(T_f)} \leq p_f - p_0 \leq \frac{\bar{v} - (1 - T_f K_3(T_f))v_0}{K_3(T_f)}
\]

\[
\frac{v}{K_1^*} \leq p_f \leq \bar{v}/K_1^*
\]

(4.12)

with \( t' \) defined as in Lemma 3, then \( v^*(t) \in [v, \bar{v}] \) and \( a^*_t \in [a, \bar{a}] \) for all \( t \in [0, T_f] \) and \( p^*(T_f) = p_f \).

4.4.4 Dynamic feasibility proofs for Section 4.4.3

**Stop-and-go motion.** Dynamical feasibility for velocities implies that \( v^*_t \in [\underline{v}, \bar{v}] \forall t \in [0, T_f] \), so by (4.3), \( \underline{v} \leq K_1(t)p_f \leq \bar{v} \). Similarly for accelerations, \( \underline{a} \leq K_2(t)p_f \leq \bar{a} \).

By non-negativity of \( K_1 \) and negativity of \( \underline{v} \), the velocity constraints are equivalent to:

\[
v/K_1^* \leq p_f \leq \bar{v}/K_1^*
\]

Similarly for acceleration, the constraints are

\[
a/K_2^* \leq p_f \leq \bar{a}/K_2^*
\]

(4.13)

(4.14)

This establishes the result for \( p_0 = 0 \). For the general case, simply replace \( p_f \) by \( p_f - p_0 \) and apply the \( p_0 = 0 \) result. Through the decoupling of axes, this result holds for all 3 jerk axes.

**Free velocity motion. Proof of Lemma 3.** We first prove the upper bound of the first inequality, derived from velocity bounds. The lower bound follows similarly. First, note that the upper bounds \( v_0 \mapsto (\bar{v} - (1 - T_f K_3(t))v_0)/K_3(t) \) are lines that intersect at \( v_0 = \bar{v} \) for all \( t \). Indeed, substituting \( v_0 = \bar{v} \) in the upper bound yields \( \bar{v} - (1 - T_f K_3(t))\bar{v})/K_3(t) = T_f \bar{v} \) regardless of \( t \). See Fig. 4.5. Thus the least upper bound is the line with the smallest intercept with the y-axis. Setting \( v_0 = 0 \) in (4.10) the intercept is \( \bar{v}/K_3(t) \). This is smallest when \( K_3(t) \) is maximized. Since \( K_3 \) is monotonically increasing (\( \frac{dK_3(t)}{dt} \geq 0 \)), \( K_3(t) \) is largest at \( t = T_f \). Thus the least upper bound on \( p_f \) is \( (\bar{v} - (1 - T_f K_3(T_f))v_0)/K_3(T_f) \).

We now prove the upper bound for the second inequality, derived from acceleration bounds. The lower bound follows similarly. The upper bounds \( t \mapsto \bar{a}/K_4(t) + T_f v_0 \)
have the same slope, $T_f$. See Fig. 4.5. The least upper bound therefore has the smallest intercept with the $y$-axis, which is $\bar{a}/K_4(t)$. The smallest intercept, yielding the smallest upper bound, occurs at the $t$ that maximizes $K_4$. Since $K_4(t)$ is concave in $t$ in the interval $[0, T_f]$, it is maximized at the solution of $\frac{dK_4(t)}{dt} = 0$. This concludes the proof. Refer to Fig. 4.5 for the intuition behind this proof.

4.5 Control of quadrotors for satisfaction of STL specifications

We are now ready to formulate the mathematical robustness maximization problem we solve for temporal logic planning. We describe it for the Free Endpoint Velocity motion; an almost-identical formulation applies for the Stop-and-Go case with obvious modifications.

Recall the notions of low-rate trajectory $x$ and high-rate discrete-time trajectory $q$ defined in Section 4.4. Consider an initial translational state $x_I = (p_I, v_I)$ and a desired final position $p_f$ to be reached in $T_f$ seconds, with free end velocity and zero acceleration. Given such a pair, the generator of Section 4.4.2 computes a trajectory $q = (p, v, a) : [0, T_f] \rightarrow \mathbb{R}^9$ that connects $p_I$ and $p_f$. By (4.5), for every $t \in [0, T_f]$, 

![Figure 4.6: The functions $K_1$ to $K_4$ for $T_f = 1$.](image)
q(t) is a linear function of \( p_I, p_f \) and \( v_f \). If the spline \( q(\cdot) \) is uniformly sampled with a period of \( dt' \), let \( H = T_f/dt' \) be the number of discrete samples in the interval \([0, T_f]\). Every \( q(dt') \) (sampled point along the spline) is a linear function of \( p_I, v_I, p_f \).

Hereinafter, we use \( x_I \rightarrow x_f \) as shorthand for saying that \( x_I \) is the initial state, and \( x_f = (p_f, v_f) \) is the final state with desired end position \( p_f \) and end velocity \( v_f = v(T_f) \) computed using the spline.

More generally, consider a sequence of low-rate waypoints \((x_0 \rightarrow x_1, x_1 \rightarrow x_2, \ldots, x_{N-1} \rightarrow x_N)\)

and a sequence \((\hat{q}_k)_{k=0}^{N-1}\) of splines connecting them and their high-rate sampled versions \(\hat{q}_k\) sampled with a period \(dt' \ll T_f\). Then every sample \(q_k(i \cdot dt')\) is a linear function of \(p_{k-1}, v_{k-1}\) and \(p_k\).

More generally, consider a sequence of low-rate waypoints

\[(x_0 \rightarrow x_1, x_1 \rightarrow x_2, \ldots, x_{N-1} \rightarrow x_N)\]

and a sequence \((\hat{q}_k)_{k=0}^{N-1}\) of splines connecting them and their high-rate sampled versions \(\hat{q}_k\) sampled with a period \(dt' \ll T_f\). Then every sample \(q_k(i \cdot dt')\) is a linear function of \(p_{k-1}, v_{k-1}\) and \(p_k\).

We now put everything together. Write \(\hat{q} = (q^0(0), \ldots, q^{N-1}(Hdt')) \in \mathbb{R}^{9N(H-1)}, \quad x = ((p_0, v_0), \ldots, (p_{N-1}, v_{N-1})) \in \mathbb{R}^{6N}\), and let \(L : \mathbb{R}^{6N} \rightarrow \mathbb{R}^{9N(H-1)}\) be the linear map between them. In the Stop-and-Go case, this uses all velocities to 0 and uses (4.8) for positions, and in the free velocity case, \(L\) uses (4.9). The robustness maximization problem is finally:

\[
\max_x \quad \tilde{\rho}_\phi(L(x)) \tag{4.15a}
\]

\[
s.t. \quad LB_v(v_{k-1}) \leq p_k - p_{k-1} \leq UB_v(v_{k-1}) \forall k = 1, \ldots, N, \tag{4.15b}
\]

\[
LB_a(v_{k-1}) \leq p_k - p_{k-1} \leq UB_a(v_{k-1}) \forall k = 1, \ldots, N, \tag{4.15c}
\]

\[
\tilde{\rho}_\phi(L(x)) \geq \varepsilon \tag{4.15d}
\]

where (4.15b) and (4.15c) are the constraints from (4.12) in Free Endpoint Velocity motion, and Thm. 4.1 in Stop-and-Go motion, with \(p_I = p_{k-1}\) and \(p_f = p_k\).

Since the optimization variables are only the waypoints \(p\), and not the high-rate discrete-time trajectory, this makes the optimization problem much more tractable. In general, the number \(N\) of low-rate waypoints \(p\) is a design choice that requires some mission specific knowledge. The higher \(N\) is, the more freedom of movement there is, but at a cost of increased computation burden of the optimization (more constraints and variables). A very small \(N\) on the other hand will restrict the freedom of motion and might make it impossible for the resulting trajectory to satisfy the STL specification.

### 4.5.1 Strictification for Continuous time guarantees

In general, if the sampled trajectory \(q\) satisfies \(\phi\), this does not guarantee that the continuous-time trajectory \(\bar{q}\) also satisfies it. For that, we use [Fainekos 2008, Thm. 5.3.1], which defines a strictification operator \(\text{str}: \phi \mapsto \phi_s\) that computes a syntactical variant of \(\phi\) having the following property.
Theorem 4.3. \textsuperscript{Fainekos [2008]} Let $dt$ be the sampling period, and suppose that there exists a constant $\Delta_g \geq 0$ s.t. for all $t$, $\|q(t) - q(t + dt)\| \leq \Delta_g dt$. Then $\rho_{\varphi_s}(q) > \Delta_g \Rightarrow (q, 0) \models \varphi$.

Intuitively, the stricter $\varphi_s$ tightens the temporal intervals and the predicates $\mu_k$ so it is ‘harder’ to satisfy $\varphi_s$ than $\varphi$. See \textsuperscript{Fainekos 2008, Ch. 5}. For the trajectory generator $g$ of Section 4.4.2, $\Delta_g$ can be computed given $T_f$, $v$, $\bar{v}$, $a$ and $\bar{a}$.

We need the following easy-to-prove result, to account for the fact that we optimize a smoothed robustness:

Corollary 4.4. Let $\varepsilon$ be the worst-case approximation error for smooth robustness. If $\tilde{\rho}_{\varphi_s}(q) > \Delta_g + \varepsilon$ then $(q, 0) \models \varphi$

This result can be proved by following the proof for \textsuperscript{Fainekos 2008, Thm. 5.3.1} and considering the approximation error for smooth robustness $\epsilon$ (see Theorem 3.1).

4.5.2 Robust and Boolean modes of solution

The control problem of (4.15) can be solved in two modes \textsuperscript{Raman et al. 2014}: the Robust (R) Mode, which solves the problem until a maximum is found (or some other optimizer-specific criterion is met). And a Boolean (B) Mode, in which the optimization (4.15) stops as soon as the smooth robustness value exceeds $\varepsilon$.

4.5.3 Implementation of the control

The controller can be implemented in one of two ways:

One-shot: The optimization of (4.15) is solved once at time 0 and the resulting trajectories are then used as a plan for the quadrotors to follow. In our simulations, where there are no disturbances, this method is acceptable or when any expected disturbances are guaranteed to be less than $\tilde{\rho}^*$, the robustness value achieved by the optimization.

Shrinking horizon: In practice, disturbances and modeling errors necessitate the use of an online feedback controller. We use a shrinking horizon approach. At time 0, the waypoints are computed by solving (4.15) and given to the quadrotors to track. Then every $T_f$ seconds, estimates for $p, v, a$ are obtained, and the optimization is solved again, while fixing all variables for previous time instants to their observed/computed values, to generate new waypoints for the remaining duration of the trajectory. For the $k^{th}$ such optimization, we re-use the $k - 1^{st}$ solution as an initial guess. This results in a faster optimization that can be run online, as will be seen in the experiments.

44
4.6 Simulations Results

We demonstrate the efficiency and the guarantees of our method through multiple examples, in simulation and in real experiments. For the simulations, we consider two case studies: a) A multi-drone reach-avoid problem in a constrained environment, and b) A multi-mission example where several drones have to fly one of two missions in the same environment. We assume a disturbance-free environment, and solve the problem in one shot, i.e. the entire trajectory that satisfies the mission is computed in one go (aka open-loop). Links to the videos for all simulations are in Table 4.5 on the last page of this chapter. The MATLAB code for all examples presented here can be obtained at https://github.com/yashpant/FlyByLogic.

4.6.1 Simulation setup

The following simulations were done in MATLAB 2016b, with the optimization formulated in Casadi [Andersson 2013] with Ipopt [Wächter and Biegler 2006] as the NLP solver. HSL routines [HSL] were used as internal linear solvers in Ipopt. All simulations were run on a laptop with a quadcore i7-7600 processor (2.8 GHz) and 16Gb RAM running Ubuntu 17.04. For all simulations, waypoints are separated by $T_f = 1$ second.

4.6.2 Multi drone reach-avoid problem

The objective of the drone $d$ is to reach a goal set $\text{Goal} ([1.5, 2] \times [1.5, 2] \times [0.5, 1])$ within the time interval $[0, T]$, while avoiding an unsafe set $\text{Unsafe} ([−1, 1] \times [−1, 1] \times [0, 1])$ throughout the interval, in the 3D position space. This environment is similar to the one in Fig. 1.8. With $p$ denoting the drone’s 3D position, the mission for a single drone $d$ is:

$$\varphi_{\text{sra}}^{d} = \square_{[0, T]}(p \notin \text{Unsafe}) \land \lozenge_{[0, T]}(p \in \text{Goal})$$ (4.16)

The Multi drone Reach-Avoid problem adds the requirement that every two drones $d, d'$ must maintain a minimum separation $\delta_{\text{min}} > 0$ from each other: $\varphi_{\text{sep}}^{d,d'} = \square_{[0, T]}(||p^d - p^{d'}|| \geq \delta_{\text{min}})$. Assuming the number of drones is $D$, the specification reads:

$$\varphi_{\text{mra}} = \land_{d=1}^{D} \varphi_{\text{sra}}^{d} \land \land_{d=1}^{D} (\land_{d' \neq d} \varphi_{\text{sep}}^{d,d'})$$ (4.17)

The horizon of this formula is $hrz(\varphi_{\text{mra}}) = T$. The robustness of $\varphi_{\text{mra}}$ is upper-bounded by 0.25, which is achievable if all drones visit the center of the set $\text{Goal}$ (as defined above) - at that time, they would be 0.25m away from the boundaries of $\text{Goal}$ - and maintain a minimum distance of 0.25 to $\text{Unsafe}$ and each other. We analyze the runtimes and achieved robustness of our controller by running a 100 simulations, each one from different randomly-chosen initial positions of the drones in the free space $X/(\text{Goal} \cup \text{Unsafe})$.
Stop and go trajectories

We show the results of controlling using (4.15) to satisfy $\varphi_{mra}$ for the case of Stop-and-Go motion (see Section 4.4.2) with $T = 6$ seconds. Videos of the computed trajectories are available at link 1 (in Boolean mode) and at link 2 (in Robust Mode) in Table 4.5.

Table 4.1 shows the run-times for an increasing number of drones $D$ in the Robust and Boolean modes. It also shows the robustness of the obtained optimal trajectories in Robust mode. (We maximize smooth robustness, then compute true robustness on the returned trajectories). As $D$ increases, the robustness values decrease. Starting from the upper bound of 0.25 with 1 drone, the robustness decreases to an average of 0.122 for $D = 5$. This is expected, as more and more drones have to visit Goal within the same time $[0,6]$, while maintaining a pairwise minimum separation of $\delta_{\text{min}}$. As a result, the drones cannot get deep into the goal set, and this lowers the robustness value.

Up to 5 drones, the controller is successful in accomplishing the mission every time. By contrast, for $D > 5$, the controller starts failing, in up to half the simulations. We conclude that for $T = 6$, the optimization can only handle up to 5 drones for this mission, in this environment.

**Comparison to MILP-based solution.** The problem of maximizing $\rho_{\varphi_{mra}}(y)$ to satisfy $\varphi_{mra}$ can be encoded as a Mixed Integer Linear Program (MILP) and solved. The tool BluSTL [Raman et al. 2014] implements this approach. We compare our runtimes to those of BluSTL for the case $D = 1$. The robustness maximization in BluSTL took $1.2 \pm 0.3s$ (mean $\pm$ standard deviation) and returned a robustness value of 0.25 for each simulation, while the Boolean mode took $0.65 \pm 0.2s$ seconds. (Note we do not include the time it takes BluSTL to encode the problem in these numbers.) Thus our approach out-performs MILP in both modes. For $D > 1$, BluSTL could not return a solution with positive robustness. The negative-robustness solutions it did return required several minutes to compute. It should however be noted that BluSTL is a general purpose tool for finding trajectories of dynamical systems that satisfy a given STL specification, while our approach is tailored to the particular problem of controlling multi-rotor robots for satisfying a STL specification.

Table 4.1: Stop-and-Go motion. Mean $\pm$ standard deviation for run-times (in seconds) and robustness values from 100 runs of the reach-avoid problem.

<table>
<thead>
<tr>
<th>$D$</th>
<th>Boolean mode</th>
<th>Robust mode</th>
<th>$\rho^*$ (Robust mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$0.078 \pm 0.004$</td>
<td>$0.40 \pm 0.018$</td>
<td>$0.244 \pm 0$</td>
</tr>
<tr>
<td>2</td>
<td>$0.099 \pm 0.007$</td>
<td>$1.313 \pm 0.272$</td>
<td>$0.198 \pm 0.015$</td>
</tr>
<tr>
<td>3</td>
<td>$0.134 \pm 0.015$</td>
<td>$2.364 \pm 0.354$</td>
<td>$0.176 \pm 0.018$</td>
</tr>
<tr>
<td>4</td>
<td>$0.181 \pm 0.024$</td>
<td>$3.423 \pm 0.370$</td>
<td>$0.160 \pm 0.031$</td>
</tr>
<tr>
<td>5</td>
<td>$0.214 \pm 0.023$</td>
<td>$7.009 \pm 3.177$</td>
<td>$0.122 \pm 0.058$</td>
</tr>
</tbody>
</table>
Trajectories with free end point velocities

We also solved the multi-drone reach-avoid problem for Free Velocity motion (Section 4.4.2), with $T = 6$. An instance of the resulting trajectories are available in links 3 and 4 for Boolean mode and at links 5 and 6 for Robust mode in Table 4.5. Table 4.2 shows the runtimes for an increasing number of drones $D$ in the Robust and Boolean modes. It also shows the robustness of the obtained optimal trajectories in Robust mode. As before, the achieved robustness value decreases as $D$ increases.

Unlike the stop-and-go case, positive robustness solutions are achieved for all simulations, up to $D = 16$ drones. This is due to the added freedom of motion between waypoints. This matches the intuition that a wider range of motion is possible when the quadrotors do not have to come to a full stop at every waypoint.

For this case, we did not compare with BluSTL as the there is no easy way to incorporate this formulation in BluSTL.

Table 4.2: Free Endpoint Velocity motion. Mean ± standard deviation for runtimes (in seconds) and robustness values from 100 runs of the reach-avoid problem.

<table>
<thead>
<tr>
<th>$D$</th>
<th>Boolean mode</th>
<th>Robust mode</th>
<th>$\rho^*$ (Robust mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.081 ± 0.005</td>
<td>0.32 ± 0.18</td>
<td>0.247 ± 0</td>
</tr>
<tr>
<td>2</td>
<td>0.112 ± 0.016</td>
<td>0.86 ± 0.15</td>
<td>0.188 ± 0.026</td>
</tr>
<tr>
<td>4</td>
<td>0.244 ± 0.017</td>
<td>1.88 ± 0.17</td>
<td>0.149 ± 0.040</td>
</tr>
<tr>
<td>5</td>
<td>0.307 ± 0.056</td>
<td>3.48 ± 0.56</td>
<td>0.137 ± 0.018</td>
</tr>
<tr>
<td>6</td>
<td>0.439 ± 0.073</td>
<td>9.08 ± 0.85</td>
<td>0.102 ± 0.032</td>
</tr>
<tr>
<td>8</td>
<td>0.651 ± 0.042</td>
<td>15.86 ± 2.15</td>
<td>0.0734 ± 0.018</td>
</tr>
<tr>
<td>10</td>
<td>0.843 ± 0.077</td>
<td>16.64 ± 1.30</td>
<td>0.051 ± 0.017</td>
</tr>
<tr>
<td>12</td>
<td>1.123 ± 0.096</td>
<td>23.99 ± 5.81</td>
<td>0.033 ± 0.003</td>
</tr>
<tr>
<td>16</td>
<td>1.575 ± 0.114</td>
<td>32.21 ± 6.25</td>
<td>0.028 ± 0.005</td>
</tr>
</tbody>
</table>

Discussion The simulations show the performance and scalability of our method, finding satisfying trajectories for $\varphi_{mra}$ for 16 drones in less than 2 seconds on average, while maximizing robustness in 35 seconds. On the other hand, the MILP-based approach does not scale well, and for the cases where it does work, is considerably slower than our approach.

Since the high-level control problem is solved at 1 Hz, the runtimes (in the boolean mode) suggest that we can control up to 2 drones online in real-time without too much computation delay: Stop-and-Go takes an average of 0.099s for 2 drones (Table 4.1), and Free Endpoint Velocity takes an average of 0.11s for 2 drones (Table 4.2). Moreover, when applied online, we solve the optimization in a shrinking horizon fashion, drastically reducing runtimes in later iterations.
4.6.3 Multi drone multi mission example

Our method can be applied to scenarios with multiple missions We illustrate this with the following 2-mission scenario:

- Mission **Pkg**: A package delivery mission. The drone(s) \( d \) has to visit a Delivery region to deliver a package within the first 10 seconds and then visit the Base region to pick up another package, which becomes available between 10 and 20 seconds later. In STL,

\[
\varphi_{\text{pkg}}^d = \Diamond_{[0,10]}(p^d \in \text{Deliver}) \land \Diamond_{(10,20]}(p^d \in \text{Base})
\]

- Mission **Srv**: A surveillance mission. The drone(s) \( d \) has to, within 20 seconds, monitor two regions sequentially. In STL,

\[
\varphi_{\text{srv}}^d = \Diamond_{[0,5]}(p^d \in \text{Zone}_1) \land \Diamond_{(5,10]}(p^d \in \text{Zone}_2) \\
\land \Diamond_{[10,15]}(p^d \in \text{Zone}_1) \land \Diamond_{(15,20]}(p^d \in \text{Zone}_2)
\]

In addition to these requirements, all the drones have to always maintain a minimum separation of \( \delta_{\text{min}} \) from each other (\( \varphi_{\text{sep}}^{d,d'} \) above), and avoid two unsafe sets Unsafe_1 and Unsafe_2. Given an even number \( D \) of drones, the odd-numbered drones are flying mission **Pkg**, while the even-numbered drones are flying mission **Srv**. The overall mission specification over all \( D \) drones is:

\[
\varphi_{x\text{-mission}} = \land_{d \text{ Odd}} \varphi_{\text{pkg}}^d \land \land_{d \text{ Even}} \varphi_{\text{srv}}^d \land \land_{d \leq D} \land_{d' \neq d} \varphi_{\text{sep}}^{d,d'} \\
\land \land_{d \leq D} \land_{i=1}^2 \Box_{[0,20]} \neg(p^d \in \text{Unsafe}_i)
\]

The mission environment is shown in Fig. 4.7. Note that the upper bound on robustness is again 0.25.

Table 4.3: Mean ± standard deviation for runtimes (in seconds) and robustness values for one-shot optimization. Obtained from 50 runs of the multi-mission problem with random initial positions.

<table>
<thead>
<tr>
<th>( D )</th>
<th>Boolean mode</th>
<th>Robust mode</th>
<th>( \rho^* ) (Robust mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.33 ± 0.08</td>
<td>4.93 ± 0.18</td>
<td>0.2414 ± 0</td>
</tr>
<tr>
<td>4</td>
<td>0.65 ± 0.10</td>
<td>16.11 ± 4.05</td>
<td>0.2158 ± 0.0658</td>
</tr>
<tr>
<td>6</td>
<td>2.38 ± 0.28</td>
<td>24.83 ± 7.50</td>
<td>0.1531 ± 0.0497</td>
</tr>
<tr>
<td>8</td>
<td>20.82 ± 4.23</td>
<td>32.87 ± 2.26</td>
<td>0.0845 ± 0.0025</td>
</tr>
</tbody>
</table>

**Results.** We solved this problem for \( D = 2, 4, 6, 8 \) drones, again with randomly generated initial positions in both Robust and Boolean modes. Videos of the resulting trajectories are in links 7-10 of Table 4.5. The runtimes increase with the number of drones, as shown in Table III. The runtimes are very suitable for offline computation. For online computation in a shrinking horizon fashion, they impose an update rate...
of at most 1/2 Hz (once every 2 seconds). In general, in the case of longer horizons, this method can serve as a one-shot planner.

For $D > 8$, the optimization could not return a solution that satisfied the STL specification. This is likely due to the small size of the sets that have to be visited by all drones in the same time interval, which is difficult to do while maintaining minimum separation (see Fig. 4.7). So it is likely the case that no solution exists, which is corroborated by the fact that maximum robustness decreases as $D$ increases.

4.7 Experiments on real quadrotors

We evaluate our method on Crazyflie 2.0 quadrotors (Fig. 4.1). Through these experiments we aim to show that: a) the velocity and acceleration constraints from Section 4.4.3 indeed ensure that the high-rate trajectory $y$ generated by robustness maximization is dynamically feasible and can be tracked by the MPC position controller, and b) our approach can control the quadrotors to satisfy their specifications in a real-time closed loop manner. A real-time playback of the experiments is in the links 11-16 of Table 4.5 on the last page of this chapter.
4.7.1 Experimental Setup

The Crazyflies are controlled by a single laptop running ROS and MATLAB. For state estimation, a Vicon motion capture system gave us quadrotor positions, velocities, orientations and angular velocities. In order to control the Crazyflies, we: a) implemented the robustness maximization using Casadi in MATLAB, and interfaced it to ROS using the MATLAB-ROS interface provided by the Robotics Toolbox, b) implemented a Model Predictive Controller (MPC) using the quadrotor dynamics linearized around hover for the position controller, coded in C using CVXGEN and ROS, c) modified the ETH tracker in C++ to work with ROS. The Crazyflie has its own attitude controller flashed to the onboard microcontroller. The robustness maximizer runs at 1Hz, the trajectory generator runs at 20Hz, the position controller runs at 20Hz and the attitude controller runs at 50Hz.

4.7.2 Validating the feasibility of generated trajectories

Figs. 4.8 and 4.9 shows the tracking of the positions and velocities commanded by the spline. The near-perfect tracking in x,y axes and satisfactory tracking in the z
axis shows that we are justified in assuming that imposing velocity and acceleration bounds on the robustness maximizer produces dynamically feasible trajectories that can be tracked by the position and attitude controllers. Note that the tracking in $z$ is not perfect due to a combination of modeling error in the model for the MPC and the lack of thrust compensation as the batteries onboard the Crazyflies deplete.

### 4.7.3 Online real-time control

We fly the reach-avoid problem (in both Stop-and-Go and Free Endpoint Velocity modes), for one and two drones, with $T = 6$ seconds, and with a maneuver duration $T_f$ set to 1 second. The controller operates in Boolean mode.

The shrinking horizon approach of Section 4.5 is used with a re-calculation rate of 1 Hz. This approach can be implemented in an online manner in real-time when the computation time for the high-level optimization (4.15) is much smaller than the re-calculation rate of the optimization, as it is in the cases we consider here.

We repeat each experiment multiple times. For every run, the quadrotors satisfied
the STL specification of (4.17). The runtimes are shown in Table 4.4. Using the optimal solution at the previous time step as the initial solution for the current time step results in very small average runtimes per time-step. This shows that our method can be easily applied in a real-time closed-loop manner. Videos are in links 11-13 in Table 4.5.

Table 4.4: Average runtime per time-step (in seconds) of shrinking horizon robustness maximization in Boolean mode. These are averaged over 5 repetitions of the experiment from the same initial point, to demonstrate the reproducibility of the experiments.

<table>
<thead>
<tr>
<th>D</th>
<th>Stop-and-Go</th>
<th>Free Endpoint Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.052</td>
<td>0.065</td>
</tr>
<tr>
<td>2</td>
<td>0.071</td>
<td>0.088</td>
</tr>
</tbody>
</table>

We observed that for more than 2 quadrotors, the online delay due to the optimization and the MATLAB-ROS interface (the latter takes up to 10 ms for receiving a state-estimate and publishing a waypoint command) is large enough that the quadrotor has significantly less than $T_f$ time to execute the maneuver between waypoints, resulting in trajectories that sometimes do not reach the goal state.

4.7.4 Offline planning for multiple drones
Our approach can be used as an offline path planner. Specifically, we solve the problem (4.15) offline for Free Endpoint Velocity motion, and use the solution low-rate trajectory $x$ as waypoints. Online, we run the trajectory generator of Section 4.4.2 (and lower-level controllers) to navigate the Crazyflies between the waypoints in a shrinking horizon fashion. We did this for all 8 Crazyflies at our disposal, and we expect it is possible to support significantly more Crazyflies, since the online computation (for the individual position and attitude controllers of the drones) is completely independent for the various drones. A video of this experiment is available in the links of Table 4.5.

4.8 Links to simulation and experiment videos
Table 4.5 has the links for the visualizations of all simulations and experiments performed in this work.
Table 4.5: Links for the videos for simulations and experiments. Here, Sim. stands for Matlab simulations, CF2.0 for experiments on the Crazyflies. Stop-go and vel. free are the two modes of operation of the trajectory generator, and B (R) is the Boolean (Robust) mode of solving the control problem. Shr. Hrz. stands for the shrinking horizon mode for online control. The reader is advised to make sure while copying the link that special characters are not ignored when pasted in the browser.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Mode</th>
<th>Specification</th>
<th>Drones (D)</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sim.</td>
<td>One-shot (B), stop-go</td>
<td>$\varphi_{m,RA}$</td>
<td>1,2,4,5</td>
<td><a href="http://bit.ly/RABstopgo">http://bit.ly/RABstopgo</a></td>
</tr>
<tr>
<td>Sim.</td>
<td>One-shot (R), stop-go</td>
<td>$\varphi_{m,RA}$</td>
<td>1,2,4,5</td>
<td><a href="http://bit.ly/RARstopgo">http://bit.ly/RARstopgo</a></td>
</tr>
<tr>
<td>Sim.</td>
<td>One-shot (B), vel. free</td>
<td>$\varphi_{m,RA}$</td>
<td>1,2,4,5,6</td>
<td><a href="http://bit.ly/RAB1to6varvel">http://bit.ly/RAB1to6varvel</a></td>
</tr>
<tr>
<td>Sim.</td>
<td>One-shot (B), vel. free</td>
<td>$\varphi_{m,RA}$</td>
<td>8,10,12,16</td>
<td><a href="http://bit.ly/RAB8to16varvel">http://bit.ly/RAB8to16varvel</a></td>
</tr>
<tr>
<td>Sim.</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{m,RA}$</td>
<td>1,2,4,5,6</td>
<td><a href="http://bit.ly/RAR1to6varvel">http://bit.ly/RAR1to6varvel</a></td>
</tr>
<tr>
<td>Sim.</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{m,RA}$</td>
<td>8,10,12,16</td>
<td><a href="http://bit.ly/RAR8to16varvel">http://bit.ly/RAR8to16varvel</a></td>
</tr>
<tr>
<td>Sim.</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{x-mission}$</td>
<td>2</td>
<td><a href="http://bit.ly/multi2mission">http://bit.ly/multi2mission</a></td>
</tr>
<tr>
<td>Sim.</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{x-mission}$</td>
<td>4</td>
<td><a href="http://bit.ly/multi4mission">http://bit.ly/multi4mission</a></td>
</tr>
<tr>
<td>Sim.</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{x-mission}$</td>
<td>6</td>
<td><a href="http://bit.ly/multi6mission">http://bit.ly/multi6mission</a></td>
</tr>
<tr>
<td>Sim.</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{x-mission}$</td>
<td>8</td>
<td><a href="http://bit.ly/multi8mission">http://bit.ly/multi8mission</a></td>
</tr>
<tr>
<td>CF2.0</td>
<td>Shr.Hrz (B), vel. free</td>
<td>$\varphi_{x,RA}$</td>
<td>1</td>
<td><a href="http://bit.ly/varvel1">http://bit.ly/varvel1</a></td>
</tr>
<tr>
<td>CF2.0</td>
<td>Shr.Hrz (B), vel. free</td>
<td>$\varphi_{m,RA}$</td>
<td>2</td>
<td><a href="http://bit.ly/varvel2">http://bit.ly/varvel2</a></td>
</tr>
<tr>
<td>CF2.0</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{m,RA}$</td>
<td>1</td>
<td><a href="http://bit.ly/stopgo1">http://bit.ly/stopgo1</a></td>
</tr>
<tr>
<td>CF2.0</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{m,RA}$</td>
<td>4</td>
<td><a href="http://bit.ly/varvel4">http://bit.ly/varvel4</a></td>
</tr>
<tr>
<td>CF2.0</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{m,RA}$</td>
<td>6</td>
<td><a href="http://bit.ly/varvel6">http://bit.ly/varvel6</a></td>
</tr>
<tr>
<td>CF2.0</td>
<td>One-shot (R), vel. free</td>
<td>$\varphi_{m,RA}$</td>
<td>8</td>
<td><a href="http://bit.ly/varvel8">http://bit.ly/varvel8</a></td>
</tr>
</tbody>
</table>
Chapter 5

The Fly-by-Logic toolchain for UAV fleet planning

5.1 Introduction

Safe planning for fleets of Unmanned Aircraft Systems (UAS) performing complex missions in urban environments has typically been a challenging problem. In the United States of America, the National Aeronautics and Space Administration (NASA) and the Federal Aviation Administration (FAA) have been studying the regulation of the airspace when multiple such fleets of autonomous UAS share the same airspace, outlined in the Concept of Operations document (ConOps). While the focus is on the infrastructure and management of the airspace, the Unmanned Aircraft System (UAS) Traffic Management (UTM) ConOps also outline a potential airspace reservation based system for operation where operators reserve a volume of the airspace for a given time interval to operate in, but it makes clear that the safety (separation from other aircraft, terrain, and other hazards) is a responsibility of the drone fleet operators. This chapter presents a tool that allows an operator to plan out missions for fleets of multi-rotor UAS, performing complex time-bound missions. The tool builds upon the correct-by-construction planning method of chapter 4 by translating missions to Signal Temporal Logic (STL). Along with a simple user interface, it also has fast and scalable mission planning abilities. We demonstrate our tool for one such mission later in the chapter.

It is inevitable that autonomous UAS will be operating in urban airspaces [Federal Aviation Authority] [2018]. In the near future, operators will increasingly rely on fleets of multiple UAS to perform a wide variety of complicated missions which could consist of a combination of: 1) spatial objectives, e.g. geofenced no fly zones, or delivery zones, 2) temporal objectives, e.g. a time window to deliver a package, 3) reactive objectives, e.g. action when battery is low.

In this chapter, we present a tool [https://github.com/yashpant/FlyByLogic] that allows an operator to specify such require-
The Fly-by-Logic tool-chain. Through a MATLAB-based graphical interface (figure 5.2), the user defines the workspace and the multi-UAS mission. This mission is interpreted as an STL specification (of the form in equation 5.1), the parameters of which are passed from the interface to the Fly-by-Logic C++ library. Through interfacing with off-the-shelf optimization tools, trajectories that satisfy the mission are generated for each UAS and visualized through the user interface. The way-points that generate these trajectories can also be sent to a Robot Operating Systems (ROS) implementation of trajectory following control to be deployed on board actual robots (e.g. bit.ly/varvel8).

ments over a fleet of UAS operating in a bounded workspace and generates trajectories for all UAS such that they all satisfy their given mission in a safe manner. In order to generate these flights paths, or trajectories, our tool relies on interpreting the mission objectives as Signal Temporal Logic (STL) specifications Maler and Nickovic [2004]. We then formulate the problem of mission satisfaction as that of maximizing a notion of robustness of STL specifications Fainekos [2008]. Using the approach of Pant et al. 2018 (see chapter 4), we generate trajectories for all the UAS involved such that they satisfy the given mission objectives.
5.2 Fly-by-Logic: The tool

5.2.1 Architecture and outline

Figure 5.1 shows the architecture of the Fly-by-Logic tool. Through the user interface in MATLAB, the user defines the missions (more details in section 5.2.2). The mission specific spatial and temporal parameters are then read in by the Fly-by-Logic C++ back-end. Here, these parameters are used to generate a function for the continuously differentially approximation of the robustness of the STL specification associated with the mission. An optimization to maximize this function [Pant et al. 2018] value is then formulated in Casadi [Andersson 2013]. Solving this optimization via IPOPT [Wächter and Biegler 2006] results in a sequence of way-points for every UAS (uniformly apart in time). Also taken into account in the formulation is the motion to connect these way-points, which is via jerk-minimizing splines [Mueller et al. 2015] and results in trajectories for each UAS. Through the Fly-by-Logic library, the (original non-smooth) robustness of these trajectories is evaluated for the mission STL specification and displayed back to the user via the MATLAB interface. A positive value of this robustness implies that the generated trajectories satisfy the mission and can be flown out, while a negative value (or 0) implies that the trajectories do not satisfy the mission [Fainekos and Pappas 2009] and either some additional parameters need to be tweaked (e.g. allowable velocity and acceleration bounds for the UAS, time intervals to visit regions, or a constant for the smooth robustness computation) or that the solver is incapable of solving this particular mission from the given initial positions of the UAS.

5.2.2 The mission template

Through the interface, the user starts by defining the number of way-points \( N \) (same number for each drone), as well as the (fixed) time, \( T \) that the UAS take to travel from one way-point to the next. These way-points are the variables that the tool optimizes over, and the overall duration of the mission is then \( H = NT \) seconds. Next, the user defines regions in a bounded 3-dimensional workspace (see figure 5.2). These regions are axis-aligned hyper-rectangles and can be either Unsafe no-fly zones (in red), or Goal regions that the UAS can fly to. For each UAS, the user specifies their starting position in the workspace, as well as the velocity and acceleration bounds that their respective trajectories should respect. Finally, the user also specifies the time intervals within which the UAS need to visit some goal sets.

Through the user interface, the user-defined missions result in specifications corresponding to the following fragment of STL:

\[
\phi = \bigwedge_{u=1}^{U} \bigwedge_{d=1}^{D} (\Box I_{\neg (p_d \in \text{Unsafe}_u)}) \land \bigwedge_{d \neq d'} (\Box I_{(\|p_d - p_{d'}\|_2 \geq d_{\text{min}})}) \land \bigwedge_{g=1}^{G} \bigwedge_{d=1}^{D} (\Diamond I_{g,d} (p_d \in \text{Goal}_g)) \land \ldots \land \Diamond I_{g,d} (p_d \in \text{Goal}_g))
\] (5.1)
Here, $D$, $U$, $G$ are the number of UAS, Unsafe sets and Goal sets in the mission respectively. $I = [0, NT]$ is an interval that covers the entire mission duration, while $I^i_{g,d} \subseteq I$, $\forall i = 1, \ldots, c$ is the $i^{th}$ interval in which UAS $d$ must visit Goal $g$. $\neg$ is the boolean negation operator. $p_d$ is the position of UAS $d$.

The symbol $\square_I \phi$ corresponds to the Always operator of STL and encodes the requirement that a boolean formula $\phi$ should be true through the time interval $I$. We use this operator to enforce that the UAS never enter the Unsafe zones or get closer than $d_{\text{min}}$ meters of each other. Similarly, $\diamond_I \phi$ corresponds to the Eventually operator which encodes the requirement that $\phi$ should be true at some point in time in the interval $I$. We use this to capture the requirement that the a UAS visits a Goal region within the user defined interval $I$. More details on STL and its grammar can be found in Donzé and Maler [2010a].

Example 6. Two UAS patrolling mission. Two UAS, starting off at positions
[2, 2, 0] and [−2, −2, 0], are tasked with patrolling two sets (in green), while making sure not to enter the set in red, and also maintaining a minimum distance of 0.5m from each other. For a mission of time 20s, we set the number of way-points to 20, and the time between them to be 1s. The timing constraints on the patrolling are as follows: UAS 1 has to visit the first set in green in an interval of time [0, 5] seconds from the missions starting time, has to visit the other green set in the interval [5, 10] seconds, re-visit the first set in the interval [10, 15], and the second set again in the interval [15, 20]. UAS 2 has a similar mission, visiting the first set in the intervals the UAS 1 has to visit the second set and so on. Figure 5.2 shows the trajectories generated by our method, and [http://bit.ly/fblguieamp] shows a real-time playback of the planned trajectories visualized through the user interface.

For the mission of example 6, the temporal logic specification is:

\[
\varphi = \bigwedge_{u=1}^{2} \bigwedge_{d=1}^{2} \left( \square_{[0,20]} \neg (p_{d} \in \text{Unsafe}_{u}) \right) \land \square_{[0,20]} (||p_{1} - p_{2}||_2 \geq 0.5) \land \\
\Diamond_{[0,5]} (p_{1} \in \text{Goal}_{1}) \land \Diamond_{[5,10]} (p_{1} \in \text{Goal}_{2}) \land \Diamond_{[10,15]} (p_{1} \in \text{Goal}_{1}) \land \\
\Diamond_{[15,20]} (p_{1} \in \text{Goal}_{2}) \land \Diamond_{[0,5]} (p_{2} \in \text{Goal}_{2}) \land \Diamond_{[5,10]} (p_{2} \in \text{Goal}_{1}) \land \\
\Diamond_{[10,15]} (p_{2} \in \text{Goal}_{2}) \land \Diamond_{[15,20]} (p_{2} \in \text{Goal}_{1})
\]  

(5.2)

5.2.3 Behind-the-scenes: Generating the trajectories

In order to generate the trajectories that satisfy the mission specification, an optimization is solved (in the C++ back-end) to maximize, over \( N \) way-points for each drone, the smooth robustness of the mission STL specification evaluated for the UAS trajectories of \( NT \) seconds in duration. The constraints in the optimization ensure that the resulting trajectories are such that the resulting trajectories have velocity and accelerations within the user-defined bounds for each UAS, i.e. are kinematically feasible for the UAS to fly. See Pant et al. [2018] for details.

5.3 Planning missions with longer time-frames

While most of the examples presented so far gave a mission horizon of the order of dozens of seconds, missions for UAS fleets in the real world will be of the order of minutes. The air-space being flow in will also be significantly larger than the ones used in the proof-of-concept examples upto now. Here we present a few examples
to demonstrate how the methods of this thesis can scale up to longer timescales and larger air-spaces.

5.3.1 Case Study: Low-altitude, rural scenario

We consider an pipeline equipment surveillance example from Federal Aviation Administration [2015], which outlines a low-altitude mission profile for UAS within a sparsely populated area. Specifically, an operator will deploy UAS in a beyond-line-of-sight setting to survey pumpjacks along an active oil pipeline. The properties and characteristics of this mission profile could serve as a template for other rural use cases for UAS, such as end-point package deliveries and wildfire management. The time duration for this mission is 10 minutes, for which we plan for using the methods of chapter 4.

Mission description

Figure 5.3 depicts the airspace setting for our low-altitude UAS use case. Note that the allocated location and blocked-off altitudes for this particular mission profile indicate that we do not need to consider interference with commercial aviation. In this use case, we deploy three small multi-rotor UAS tasked with surveying five pumpjacks located along an oil pipeline. In order to carry out a successful surveillance mission, the team of UAS must fly directly over each pumpjack and collect information regarding the pumpjack, e.g. taking high-definition aerial photographs of the pumpjacks.

The mission must also be completed within a predefined time interval, wherein all three UAS must successfully fly over and survey all five pumpjacks. In actuality, the UAS operator deploying the UAS swarm will choose this mission time interval, so long as the length of the time interval is not too short. Note that the underlying STL and trajectory specifications will alert the operator if the chosen time interval is insufficient for the mission profile to be successfully completed due to UAS performance constraints.

For safety reasons and in compliance with FAA regulations regarding UAS operations, the mission profile also specifies the enforcement of pairwise separation requirements. Each UAS must maintain a separation of at least 5 meters from another UAS for the entirety of the mission time interval. Another important operating constraint enforced by the mission profile is that the UAS must stay within the predefined airspace at all times, as well as respect maximum allowable velocities and accelerations. Both physical constraints can be modified a priori as needed by the operator. Finally, the mission is considered to be completed once all UAS reaches a predefined landing area, where they will then be recovered by the operator.
Figure 5.3: Trajectories for 3 UAS tasked with reaching the green-colored goal set within 2 minutes, while avoiding the all black-colored obstacles. A pairwise separation requirement of $\geq 2$ meters is enforced for all UAS.

**STL formalization of the mission profile**

Recall that we can reformulate and express a complex set of mission specifications and constraints via STL (Section 2.2), providing a translation between the high-level mission profile and low-level trajectory synthesis. In order to capture this rural case study using STL specifications, we must first define three-dimensional sets that demarcate the various physical attributes of our mission environment. Recall that each UAS must perform a fly-over for each pumpjack; let $P_{jack_i}$ denote the airspace region directly above each pumpjack $i \in \{1, ..., 5\}$. Within the STL specifications, each UAS will be required to visit each $P_{jack_i}$ set within the time interval of the mission.

While $P_{jack_i}$ denotes airspace regions that UAS will be required to fly to, there may also be predefined regions of the airspace that must not be visited by UAS. An example could be telecommunication infrastructures in rural areas that UAS must stay clear of. We will denote such no-fly zones within the mission environment as NoFly. For our case study specifically, the no-fly zones include the physical infrastructures for each pumpjack, as well as a safety buffer region around them. Finally, for each of the three UAS $d \in \{1, 2, 3\}$, we specify a recovery region within the mission environment, denoted by $Recovery_d$. The sets $P_{jack_i}$, NoFly, and $Recovery_d$ are depicted in Figure 5.3.

In addition to the spatial components that we need to specify, we also need to analogously specify the temporal components of the mission profile. More precisely, we will define the main time interval of the mission, as well as sub-intervals during which relevant events must occur. Let $I = [0, T]$ be the main time interval of the mission, during which all five pumpjacks must be surveyed by each of the three UAS. Note that $T$ is the maximum allowable flight time allocated to the mission, specified...
in our case study in seconds. We also specify a sub-interval of time $I_d^i \subseteq I$ wherein all three UAS $d \in \{1, 2, 3\}$ must perform a fly-over and survey pumpjack $i \in \{1, ..., 5\}$. Complimentary to the recovery sets $Recovery_d$, we specify time sub-intervals $I_d \subseteq I$ wherein each UAS must enter their recovery sets, signifying the end of their mission tasks.

Now that we have defined the spatial and temporal components of the mission environment, we move on to finalize the overall mission specifications by defining the minimum pairwise separation distance as $d_{min} = 5$ meters. Let the STL specification formula for the mission assigned to each UAS $d$ be denoted by $\varphi_d$; we have that the mission for each UAS $d$ is formalized in STL as

$$\varphi_d = \bigwedge_{i=1}^5 \left( \Diamond I_d^i (p_d \in P\text{jack}_i) \right) \land \Box I (p_d \notin NoFly)$$

$$\land \Diamond I_d (p_d \in Recovery_d).$$

(5.3)

This STL formula can be parsed as follows: Each UAS $d$ must visit and fly-over the five pumpjacks ($p_d \in P\text{jack}_i$) within the time sub-intervals allocated ($\Diamond I_d^i$). While completing their flyover, all UAS must stay away from no-fly zones ($p_d \notin NoFly$), and this is enforced throughout the entirety of the main mission time interval ($\Box I$). Finally, each UAS $d$ must eventually navigate to their recovery sets ($p_d \in Recovery_d$); this must be done within the specified time sub-interval for reaching the recovery set as well ($\Diamond I_d$).

Let $\varphi_{pipeline}$ be the overall mission specification across all UAS, defined in terms of $\varphi_d$ as well as the required pairwise separation constraints. We can write $\varphi_{pipeline}$ explicitly as

$$\varphi_{pipeline} = \left( \bigwedge_{d=1}^3 \varphi_d \right) \land \left( \bigwedge_{d,d',d''} \Box I ||p_d - p_{d'}|| \geq d_{min} \right).$$

(5.4)

The overall mission specification $\varphi_{pipeline}$ states that in addition to carrying out the pipeline pumpjack surveillance mission, each unique pair $(d, d') \in \{1, 2, 3\}^2$ of UAS should be separated by at least $d_{min}$. Our STL specification for this particular mission profile is complete. Finally, we note that the requirements of staying within the predefined mission environment, as well as the predefined bounds on velocity and acceleration, are linear constraints imposed on the state of the UAS. Thus, we directly incorporate these kinematic constraints within the Fly-by-Logic planning algorithm (chapter 4).

5.3.2 Case Study: Urban Scenario

We also implemented a case study with 5 drones carrying out operations in an airport area, in particular the Philadelphia International (PHL) Airport region. Figure 5.4 shows the top-down view of the workspace. The time duration of the mission is 11 minutes.
Figure 5.4: Workspace for the PHL airport example. A 10x speedup of the planned trajectories are shown in https://youtu.be/vBYRFfnLwu8.

Mission description

No fly zones: The drones are not allowed to fly in the regions shown in red in Figure 5.4 and each of the regions are high enough to exceed the flight ceiling of the drones (100 m) so that they do not attempt to fly over the No fly zones. The planning for the problem is done in the full 3-dimensional co-ordinate space, but the visualizations are in a 2-d top down view to reduce clutter. The safety specification for all drones is:

$$\varphi_{\text{No fly}} = \bigwedge_{i=1}^{5} [0,660] \neg (p \in \text{No fly}_i)$$  \hspace{1cm} (5.5)

The 5 drones are tasked with carrying out 3 kinds of missions in the airspace.

Autonomous air-shuttle: Drone 1 is an autonomous air-shuttle flies that carries passengers between the terminal area and two parking lots. Starting from the WallyPark parking, it has to reach the terminal region. It must stay in the terminal region for 30 seconds to drop-off and pick up passengers. Following this it flies to the PreFlight parking area, where it also waits for 30 seconds before flying back in a similar manner to the terminal region and the WallyPark area. This, along with the no fly zone requirements, is captured in the following specification

$$\varphi_{\text{shuttle-1}} = \Diamond [0,100] \square [0,30] (p_1 \in \text{Terminal}) \land \Diamond [130,270] \square [0,30] (p_1 \in \text{PreFlight}) \land$$
$$\Diamond [300,440] \square [0,30] (p_1 \in \text{Terminal}) \land \Diamond [470,660] \square [0,30] (p_1 \in \text{WallyPark}) \land \varphi_{\text{No fly}}$$  \hspace{1cm} (5.6)

Drone 2 is an autonomous air-shuttle that has to ferry passengers from the Colonial
airport parking lot to the terminal region, waiting for 30 seconds at each stop. It has
to repeat the loop twice in the 11 minutes. The following captures its specification:

$$\varphi_{\text{shuttle-2}} = \Diamond [0, 150] \Box [0, 30] (p_2 \in \text{Terminal}) \land \Diamond [180, 330] \Box [0, 30] (p_2 \in \text{Colonial airport}) \land \Diamond [360, 490] \Box [0, 30] (p_2 \in \text{Terminal}) \land \Diamond [520, 660] \Box [0, 30] (p_2 \in \text{Colonial airport}) \land \varphi_{\text{No fly}}$$  \hspace{1cm} (5.7)

**Last mile autonomous delivery:** Drones 3 and 4 are tasked with carrying
packages from the DHL unloading area in the PHL airport and carrying them to the
airport business complex and returning back to the DHL area. The drones have to
wait for 50 seconds in these regions to unload and load the packages.

$$\varphi_{\text{deliver-1}} = \Diamond [0, 50] \Box [0, 50] (p_3 \in \text{Business}) \land \Diamond [100, 150] \Box [0, 50] (p_3 \in \text{DHL}) \land \Diamond [200, 250] \Box [0, 50] (p_3 \in \text{Business}) \land \Diamond [300, 350] \Box [0, 50] (p_3 \in \text{DHL}) \land \Diamond [400, 450] \Box [0, 50] (p_3 \in \text{Business}) \land \Diamond [500, 550] \Box [0, 50] (p_3 \in \text{DHL}) \land \Diamond [600, 650] \Box [0, 10] (p_3 \in \text{Business}) \land \varphi_{\text{No fly}}$$  \hspace{1cm} (5.8)

A similar specification $\varphi_{\text{deliver-2}}$ applies for drone 4.

**Autonomous air-taxi:** The final drone is an on-demand air-taxi that in this
instance picks up a passenger from the terminal region, and then has to drop the
passenger off at Stephenson equipment (in the top left corner of figure 5.4) in under
11 minutes. The specification is given by:

$$\varphi_{\text{air-taxi}} = \Diamond [0, 660] (p_5 \in \text{Stephenson}) \land \varphi_{\text{No fly}}$$  \hspace{1cm} (5.9)

Using the method of chapter 4, the planning is done for all 5 drones (in a central-
ized manner) carrying out their missions, along with the requirement that they must
all be pairwise at least 5 m away from each other. This is captured in the specification:

$$\varphi_{\text{PHL}} = \bigwedge_{d=1}^2 \varphi_{\text{shuttle-d}} \land \bigwedge_{d=3}^4 \varphi_{\text{deliver-d}} \land \varphi_{\text{air-taxi}} \land \bigwedge_{d \neq d'}^4 \Box [0, 660] (||p_d - p_{d'}|| \geq 5)$$  \hspace{1cm} (5.10)

**Results:** The Fly-by-Logic algorithm (chapter 4) generated trajectories for all
5 drones such that they satisfy the requirement $\varphi_{\text{PHL}}$ (taking about 1 minute to
find trajectories that satisfy the specification). This shows the methods ability to
plan for long missions (order of minutes) and deal with nested operators. [https://youtu.be/vBYRFfnLwu8](https://youtu.be/vBYRFfnLwu8) shows a 10x speed up of the trajectories.

### 5.4 Conclusion

This chapter presented an interface that allows UAS operators to specify missions
graphically. These missions correspond a fragment of STL, and the planning is then
done through an implementation of the Fly-by-Logic approach of chapter 4. The underlying code base can handle nested operators and can also work for missions with long time horizons as shown in the examples above.
Chapter 6

Anytime Computation and Control for Autonomous Systems

6.1 Introduction

The real-time control of many autonomous robots, e.g. self-driving cars and Unmanned Aerial Systems (UASs), usually includes closed loops between the controller that drives the actuation, and the estimator that computes state estimates which are used by the controller. Of particular importance in this traditional feedback control architecture are: a) the delay in the control action due to the time taken by estimator for computing the state estimate and, b) the inaccuracy in the state estimate. Either of these factors can result in control actions that can drive the system into an unsafe state.

In most conventional feedback control designs, controllers are tasked with realizing the functional goals of the system under simplistic assumptions on the performance of the estimator, in particular, perfect state estimates and negligible computation time. This design principle based on separation of concerns simplifies the control design process but often does not accurately reflect real implementations. On the other hand, most perception-based state estimation algorithms (e.g. SVO [Forster et al. 2014] and ORB-SLAM [Mur-Artal et al. 2015]) do not take into account how their output will be used to close the control loop. More specifically, an estimator will more often than not run to completion: i.e., its termination criteria are designed to provide the best quality output (estimate). This can result in large delays in the control action, leading to degraded control performance. It can also result in the computation platform consuming a significant amount of energy, reducing the amount of time the system can operate on a full charge. This is specially of concern in mobile robotic systems like autonomous drones and cars, that operate on batteries with limited capacity.

In this work we focus on these problems, that when the real-time requirements on the closed-loop system become more demanding, this disconnect between the es-
Perception-based Estimator

Controller

State Estimate (Delay, Error)

Contract

Control Action to Physical System (e.g. Motor Speed)

Physical System (e.g. Autonomous Robot)

Estimation and Control

Measurement from Sensors (e.g. Video Feed)

Figure 6.1: Contract-driven controller and estimator. The co-design allows the controller to set a contract for the perception-based state estimator, in addition to controlling the dynamical system.

timator and controller can lead to poor system performance. The following example shows how this problem can manifest in even simple settings.

Example 7. To illustrate the impact of estimation delay $\delta$ and state estimation error $\epsilon$ on control performance, we show a simple PID tracker controlling the motion of a point mass in the $(x,y)$ plane. The position of the point mass must follow a reference constant trajectory, whose $x$ dimension is shown in Fig. 6.2 (the same plot can be obtained for the $y$ position). We simulate two cases of estimation (and therefore actuation) delay and error, where a larger delay value $\delta$ implies a smaller estimation error $\epsilon$. As can be noted in Fig. 6.2, the effect of delay can be non-negligible. In this example, it can be seen that the increased delay causes the tracking performance to worsen. Running an estimation task with a fixed smaller delay but larger estimation error does not necessarily solve the problem of degraded performance, as can be seen in Fig. 6.2. Therefore, there is a need to rigorously quantify the trade-off between computation time and estimation error, then exploit that trade-off to achieve the best control performance under the problem constraints. Rather than always running the estimation task to completion, it is useful to have several delay/error run-time modes for the estimator. These can then be used at run-time to satisfy the control objectives.

The goal of this chapter is to develop a rigorous framework for the co-design of the controller and estimation algorithms. In this framework, the estimator has a range of computation time/estimate quality operating modes, and in order to best maintain control performance and reduce energy consumption, the controller at run-
time selects one of these modes for the estimator to operate in. This is motivated by
the following observations:

1) The traditional engineering approach to account for the estimator’s run-time is
to gauge the Worst-Case Execution Time (WCET) of the estimation task, and design
the system to meet deadlines under the WCET conditions. In practice however,
the actual execution time of perception-based estimators can be much less than the
WCET and depends on the actual data being processed. Hence, considering the
WCET can lead to a conservative design of the system. Additionally, the classical
timing analysis alone does not guarantee functional correctness of the closed-loop
system under control.

2) Moreover, in the context of closed loop control, we do not always require the
best quality state estimate: more often than not, a lower quality estimate, computed
using lesser energy and time, is acceptable to achieve the control objectives.

3) In the case where obtaining a better quality state estimate requires longer
computation time, it can be detrimental to the control performance to require a high
quality state estimate all the time. For example, when the on-board computer is
overloaded, there may be a need to spend less time computing a state estimate so
that not only the control action has less delay, but also so that other processes can
access the computation resource as scheduled.

Here, we develop the observations above into a co-design framework for a real-time
control systems, where the controller and estimator are interfaced via contracts. A
contract is an assurance requested by the controller, and provided by the estimator, that the latter can give an estimate with a certain accuracy $\epsilon$, and within a predefined time deadline $\delta$. The computation time given to the estimator, as well as the quality of the state estimate define the contract. This can be interpreted as turning the estimator into a discretized version of an anytime algorithm [Boddy and Dean 1989] where its computation can be interrupted at runtime to get a state estimate, usually with a trade-off between the computation time given to the algorithm and the quality of output that it returns. Through this notion of contracts, we show how the controller can vary the computation time of the estimation algorithm to maintain control performance and to reduce energy consumption. The work presented here is focused on estimation algorithms that rely on computationally intensive Computer Vision (CV) algorithms in order to get a state estimate of a dynamical system, e.g. those in autonomous robot navigation with visual (camera, Lidar) sensors. We refer to these as perception-based estimators. Through experiments, we show that the computation time of such algorithms can be significant (and much greater than that of the control algorithm), resulting in an adverse impact on the closed loop control performance.

The architecture for the co-design framework proposed in this work is shown in Fig. 6.1. It resembles the conventional closed loop control architecture involving the estimator, the controller, and the system being controlled, but also incorporates the (delay, error) contract as an interface between the controller and the estimation algorithm.

**Summary of contributions.** In this chapter present a framework for the co-design of control and estimation algorithms for the real-time control of dynamical systems. This approach consists of:

- a well-defined interface between control and estimation, in the form of operating modes, or contracts, on the accuracy and computation time of the estimator (Section 6.2),
- characterizing the estimator accuracy as either deterministic (worst-case) or stochastic through offline profiling of the perception-based estimator (Section 6.6),
- a predictive control algorithm that can change the operating mode of the estimator at run-time to achieve control objectives at a lower energy cost (Section 6.3), while providing guarantees on satisfaction of constraints for both deterministic (Section 6.4) and probabilistic (Section 6.5) characterizations of the estimation error, and,
- a straightforward, low-touch and low-effort approach to design a contract-driven estimation algorithm starting from an off-the-shelf, run-to-completion version of it (Section 6.6).
We demonstrate our method on an autonomous flying robot (shown in Fig. 6.3) and show its performance and energy gains over a classical controller (Section 6.7).

In this chapter we extend the framework of Pant et al. [2015] to also allow for a probabilistic representation for estimation error. We also provide guarantees on satisfaction of constraints and recursive feasibility of the new control predictive algorithm resulting from this probabilistic setup. In addition, we also extend the experimental setup and incorporate a real-time implementation of the new control algorithm and evaluate our approaches with two sets of new experiments on a hex-rotor autonomous robot.

6.2 Co-design of estimation and control

Conventional closed loop control systems are generally designed in a manner where the controller is incognizant of the implementation details of the state estimation module, while the estimation module is designed independent of the requirements of the controller. For example, a feedback controller, that gets state estimates from a camera based visual odometry algorithm, might not be designed to take into account the non-negligible time taken to process the video frames to get a state estimate. We refer to this computation time as the estimation delay. On the other hand, the design of most perception-based estimators does not take into account the varying real-time constraints that the controlled closed-loop system must satisfy. Also of importance, especially in autonomous systems deployed in the field, is the power consumed by the computation platform which can have a significant impact on the duration the
Taking these factors into account, we propose the co-design of estimation and control to improve the closed loop performance of real-time control on systems with computationally and power limited platforms. This is done through a contract-driven framework for both estimator and controller in which the controller asks for a state estimate within a certain deadline \( \delta \) seconds, with an associated bound on the inaccuracy of the estimation. This inaccuracy can either be in the form of a hard bound \( \epsilon \), e.g. an infinite-norm bound on the estimation error vector, or have a probabilistic characterization \( \Sigma \), e.g. the covariance of the estimation error vector, depending on the application. For the sake of simplicity, we use \( \epsilon \) for the characterization of the estimation error in the following text.

In our framework, the tuple \( (\delta,\epsilon) \) forms the contract between controller and estimator. The estimator is tasked with providing a state estimate that respects the contract. Aware of these contracts, the controller can set the appropriate contract in a time varying manner to adapt the closed-loop system performance in real-time to take into account the control requirements of the physical system. For example, it can decide when an estimate is needed fast (but usually with higher error), and when a more accurate estimate is needed (but with greater delay). Note, the \( (\delta,\epsilon) \) contract can also be thought of as setting an operating mode for the perception-based estimator. A high-level view of this setup is shown in Fig. 6.1.

In order to make sure that the contracts are such that the estimator can indeed fulfill them, the estimator is profiled off-line. To do this, the estimator’s internal parameters are varied, and for each parameter setting, it is run on a profiling data set (with a known ground-truth baseline). This results in a set of \( (\delta,\epsilon) \) values, each one
corresponding to a particular setting of the parameters. These values can be plotted on a curve, which we call the error-delay curve made up of discrete points, \((\delta, \epsilon)\). Examples of such a curve are shown in Figs. 6.7 and 6.9. Section 6.6 provides the detailed procedure for obtaining this curve for a perception-based estimator.

During run-time execution, upon receiving a \((\delta, \epsilon)\) contract request from the controller, the estimator can adapt its parameter settings to fulfil the contract, i.e. to provide a state estimate within the requested deadline \(\delta\) that also respects the requested error bound \(\epsilon\).

The controller, in the co-design framework, is designed with the awareness of the error-delay curve of the estimation algorithm, and requests contracts from that curve. The error-delay curve, thus constitutes the interface between the controller and state estimator. The controller leverages the flexible nature of the estimation algorithm to maximize some measure of control performance.

The closed loop architecture in a system with co-design of the estimator and controller is shown in Fig. 6.4. In this co-designed system, the controller can make the estimation algorithm switch to lower or higher time (and/or energy) consuming modes based on the control objective at the current time step. The main components of the co-design architecture presented here are: a) a contract perception-based estimator, b) a robust control algorithm that computes an input to be sent to the physical system being controlled as well as the contract for the estimator, and c) the interface between them. More details on these components are in the following sections.

### 6.3 Control with Contract-driven Estimation

In this section, we formalize how the error-delay curve of the estimator can be utilized by the control algorithm to optimize the control performance while minimizing the power consumed by the computations for the perception-based estimator.

#### 6.3.1 System Model

In order to model the co-design process, consider the closed-loop control of an autonomous hex-rotor robot (more details in 6.7), shown in Fig. 6.3. The state \(x\) of the hexrotor consists of its 3D position and 3D velocity, while the input \(u\) to the robot consists of the desired pitch and roll angles, and the desired thrust. The hexrotor’s task is to fly a pre-defined trajectory given by \(x_{\text{ref}}\), where \(x_{\text{ref}}(t)\) gives the desired position at each time \(t\). The dynamics of the hexrotor, relating the time-evolution of its state to the current state and input, can be linearized around hover and approximated by the following Linear Time-Invariant (LTI) ODE:

\[
\dot{x}(t) = A_c x(t) + B_c u(t) + w_c(t)
\] (6.1)

Here, the state vector \(x \in \mathbb{R}^n\) is constrained to be within set \(X \subset \mathbb{R}^n\), the control input \(u \in \mathbb{R}^m\) is constrained within set \(U \subset \mathbb{R}^m\), and \(w_c \in \mathbb{R}^n\) is the process noise.
Figure 6.5: Time-triggered sensing and actuation. The figure shows the varying execution time for the estimator and the blue area shows the execution time for the controller, which is small.

assumed to lie in a (bounded) set $W_c \subset \mathbb{R}^n$. $A_c \in \mathbb{R}^{n \times n}$ and $B_c \in \mathbb{R}^{n \times m}$ are matrices. LTI ODEs can model a wide range of systems, and our results apply to arbitrary LTI systems of the form given in (6.1) with compact and convex constraint sets $X, U$ and $W_c$. The sets $X$ and $U$ are determined by the control designer or by physical constraints on the system. For example, $X$ captures limits on the state to define the region which the hexrotor can fly and the velocity limits on it. The set $U$ restricts the inputs to values that can be supported by the rotors, as well as within which the linearized system provides a good approximation to the true nonlinear dynamics.

### 6.3.2 Time-Triggered Sensing and Actuation

For feedback control of the hexrotor, the controller needs to be aware of the hexrotor’s current position and speed, i.e. requires an estimate of its current state $x$. This is done via a perception-based estimator, that process video frames (at a fixed rate) obtained through a downward facing camera mounted on the hexrotor. The estimator detects and tracks features across frames, and deduces its own position through the relative motion of these features.

A new frame is captured by the camera every $T > 0$ seconds, which results in periodic measurements at instants $t_{s,k} = kT$, where $k \in \mathbb{N}$. This measurement is used by the estimator to compute the state estimate $\hat{x}_k := \hat{x}(t_{s,k})$ with the desired accuracy $\epsilon_k$ determined by the contract set by the controller in the previous time step. The controller then acts on this state estimate to compute the control input $u_k$ as well as decide on the perception-based estimator’s delay and accuracy contract $(\delta_k, \epsilon_k)$ for the next time step. The control is then applied to the physical system according to (6.1) at instant $t_{a,k} = t_{s,k} + \delta_k + \tau_k$, where $\tau_k$ is the time it takes to compute the input. See Fig. 6.5 for the timing diagram of this process.

The controller has access to the delay-error curve, or operating modes $\Delta$ of the estimator, and at each time step selects contracts from that curve. This curve is obtained offline as explained in Section 6.2 and illustrated in Section 6.6. Note that at each step $k \geq 0$, the estimation accuracy $\epsilon_k$, and hence the delay $\delta_k$ are already
decided in the previous time step and known to the controller. For the very first step \( k = 0 \), the initial estimation mode \( \delta_0, \epsilon_0 \), as well as the initial control input \( u_{-1} \) are chosen by the designer.

### 6.3.3 Control Performance

The controller has a goal that is twofold: it needs to ensure that the reference trajectory is tracked as closely as possible, and that the computation energy consumed to do so is minimized. To capture this, we define two (stage) cost functions: first, 
\[
\ell(x, u) = (x - x_{\text{ref}})^T Q(x - x_{\text{ref}}) + u^T R u
\]
defines a weighted sum of the tracking error (first summand) and the input power (second summand). Here, \( Q \) and \( R \) are positive semidefinite and positive definite matrices respectively. Second, \( \pi(\delta) \) captures the average power consumed to perform a perception-based estimation computation duration \( \delta \). This power information is collected offline during the estimator profiling phase.

The total cost function for the controller to minimize is 
\[
J = \sum_{k=0}^{M} \left( \ell(x_k, u_k) + \alpha \pi(\delta_k) \right),
\]
where \( M \geq 0 \) is the duration of the system’s operation.

### 6.3.4 Discretized Dynamics

Due to the time-triggered sensing and actuation of the system (see Sec. 6.3.2), from time \( t_{s,k} \) to \( t_{a,k} \), the previous control input \( u_{k-1} \) is still being applied. Then at \( t_{a,k} \) the new control input \( u_k \) is computed and applied by the controller (see Fig. 6.5). For the sake of simplicity, we assume the computation time for the controller \( \tau \) is small and constant, and so lump it with the time for the estimator \( \delta \). This is justified experimentally for our problem (in Sec. 6.7) where the time for the controller is negligible compared to the time taken by the estimation algorithm. The discrete time dynamics for this setup, with a periodic sensing time of \( T \), are given by

\[
x_{k+1} = Ax_k + B_1(\delta_k)u_{k-1} + B_2(\delta_k)u_k + w_k, k \geq 0
\]

in which

\[
A = e^{AcT}, \quad w_k = \int_0^T e^{Ac(T-t)}w_c(t_{s,k} + t)dt
\]

\[
B_1(\delta) = \int_0^\delta e^{Ac(T-t)}B_c dt, \quad B_2(\delta) = \int_\delta^T e^{Ac(T-t)}B_c dt.
\]

Here, \( w_k \) is the process noise accumulated during the interval. It is constrained to lie in a compact convex set \( W \) since \( w_c(t) \) lies in the compact convex set \( W_c \) and \( T \) is finite. As explained above, both the current control \( u_k \) and the previous control \( u_{k-1} \) appear in (6.2). In addition, the input matrices \( B_1(\delta_k) \) and \( B_2(\delta_k) \) depend on the delay \( \delta_k \). The estimation accuracy \( \epsilon_k \), indirectly affects the dynamics via the control input, which is computed using the state estimate \( \hat{x}_k \). These discrete time dynamics therefore show how the operation mode of the estimator \((\delta, \epsilon)\) affects the dynamics of the system.
6.4 Robust Model Predictive Control Solution

In this section we give an overview of the Robust Adaptive Model Predictive Controller (RAMPC) that we use in the contract-driven setup of Fig. 6.4. Here, we consider the estimation errors to be bounded, and use these worst-case bounds in the controller formulation. The mathematical details and derivations are available in the appendix. Experiments confirm that the following controller can be run in real-time, and its computation uses a negligible amount of time relative to the estimation delay.

6.4.1 Solution overview

Recall the operation of the contract-driven control and estimation framework as presented in Section 6.2 and Fig. 6.4. First, the estimator is profiled offline to obtain its delay-error curve, which we denote by $\Delta$. The curve $\Delta$ represents a finite number of $(\delta, \epsilon)$ contracts that the estimator can satisfy. At every time step $k$, the controller receives a state estimate $\hat{x}_k$ and uses it to compute the control input $u_k$ to be applied to the physical system at time $t_{a,k}$ and the contract $(\delta_{k+1}, \epsilon_{k+1}) \in \Delta$ that will be requested from the estimator at the next step. At $k+1$, the estimator provides an estimate with error at most $\epsilon_{k+1}$ and within delay $\delta_{k+1}$. Finally, recall that $J = \sum_{k=0}^{M} (\ell(x_k, u_k) + \alpha \pi(\delta_k))$ combines tracking error and input power in the $\ell$ terms, and estimation power consumption in the $\pi$ terms. The scalar $\alpha$ quantifies the importance of power consumption to the overall performance of the system.

The contract-driven controller’s task is to find a sequence of inputs $u_k \in U$ and of contracts $(\delta_k, \epsilon_k) \in \Delta$ such that the cost $J$ is minimized, and the state $x_k$ is always in the set $X$. The challenge in finding the control inputs is that the controller does not have access to the real state $x_k$, but only to an estimate $\hat{x}_k$. The norm of the error $e_k = \hat{x}_k - x_k$ is bounded by the contractual $\epsilon_k$, which varies at each time step.

Let us fix the prediction horizon $N \geq 1$. Assume that the current contract (under which the current estimate $\hat{x}_k$ was obtained) is $(\delta_k, \epsilon_k)$, and that the previously applied input is $u_{k-1}$. To compute the new input value $u_k$ and next contract $(\delta_{k+1}, \epsilon_{k+1})$, the proposed Robust Adaptive Model Predictive Controller (RAMPC) seeks to solve the following optimization problem which we denote by $P_{\Delta}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})$:

$$J^*[0:N] = \min_{u,x,\delta,\epsilon} \sum_{j=0}^{N} (\ell(x_{k+j}, u_{k+j}) + \alpha \pi(\delta_k))$$

s.t. \forall j \in \{0, \ldots, N\}
$$x_{k+j+1} = Ax_{k+j} + B_1(\delta_k)u_{k+j-1} + B_2(\delta_k)u_{k+j}$$
$$[x_{k+j+1}, u_{k+j}]' \in X \times U$$

Here, RAMPC needs to find the optimal length-$N$ input sequence $u^* = (u_k^*, \ldots, u_{k+N}^*) \in U^N$, corresponding state sequence $x = (x_k, \ldots, x_{k+N}) \in X^N$, delay sequence $\delta = (\delta_k, \ldots, \delta_{k+N})$ and error sequence $\epsilon = (\epsilon_k, \ldots, \epsilon_{k+N})$ such that $(\delta_k, \epsilon_k) \in \Delta$, which minimize the $N$-step cost $J[0:N]$. The matrices that make up the system dynamics
are defined in Section [6.3.4]. As in regular MPC [Camacho and Bordons 2004], once a solution \( u^* \) is found, only the first input value \( u^*_k \) is applied to the physical system, thus yielding the next state \( x_{k+1} \) as per (6.2). At the next time step \( k + 1 \), RAMPC sets up the new optimization \( P_{\Delta}(\hat{x}_{k+1}, \delta_{k+1}, \epsilon_{k+1}, u_{k+1-1}) \) and solves it again.

To make this problem tractable, we first assume that the mode is fixed throughout the \( N \)-step horizon, i.e. \( (\delta_{k+j}, \epsilon_{k+j}) = (\delta, \epsilon) \) for all \( 1 \leq j \leq N \). Thus for every value \( (\delta, \epsilon) \) in \( \Delta \), we can set up a different problem (6.3) and solve it. Let \( J^*_{(\delta, \epsilon)} \) be the corresponding optimum. The solution with the smallest objective function value yields the input value \( u^*_k \) to be applied and the next contract \( (\delta^*, \epsilon^*) \).

Because RAMPC only has access to the state estimate, we extend the RMPC approach in [Richards and How 2005b; Chisci et al. 2001]. Namely, the problem is solved for the nominal dynamics which assume zero process and observation noise \( (w_{k+j} = 0) \) and zero estimation error \( (\hat{x}_{k+j} = x_{k+j}) \) over the prediction horizon. Let \( \bar{x} \) be the state of the system under nominal conditions. To compensate for the use of nominal dynamics, RMPC replaces the constraint \((\bar{x}_{k+j}, u_{k-1+j}) \in X \times U := Z\) by \((\bar{x}_{k+j}, u_{k+j}) \in \mathcal{Z}_j(\epsilon_k, \epsilon)\), where \( \mathcal{Z}_j(\epsilon_k, \epsilon) \subset Z \) ‘shrunk’ by an amount corresponding to \( \epsilon \), as explained in the appendix. Intuitively, by forcing \((\bar{x}_{k+j}, u_{k-1+j})\) to lie in the reduced set \( \mathcal{Z}_j(\epsilon_k, \epsilon) \), the bounded estimation error and process noise are guaranteed not to cause the true state and input to exit the constraint sets \( X \) and \( U \).

The tractable optimization for a given \( (\delta, \epsilon) \), denoted by \( P_{(\delta, \epsilon)}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1}) \), is then

\[
J^*_{(\delta, \epsilon)} = \min_{u, x} \sum_{j=0}^{N} \left( \ell(\bar{x}_{k+j}, u_{k+j}) + \alpha \pi(\delta) \right)
\]

s.t. \( \forall j \in \{0, \ldots, N\} \)
\[
\bar{x}_{k+j+1} = A\bar{x}_{k+j} + B_1(\delta)u_{k+j-1} + B_2(\delta)u_{k+j}
\]
\[
(\bar{x}_{k+j}, u_{k+j}) \in \mathcal{Z}_j(\epsilon_k, \epsilon)
\]

Algorithm 1 summarizes the RAMPC algorithm.

**Algorithm 1** Robust Adaptive MPC algorithm with Anytime Estimation.

1: \((\delta_0, \epsilon_0)\) and \(u_{-1}\) specified by designer
2: Apply \( u_{-1} \)
3: for \( k = 0, 1, \ldots, M \) do
4: Estimate \( \hat{x}_k \) with guarantee \((\delta_k, \epsilon_k)\)
5: for each \((\delta, \epsilon) \in \Delta\) do
6: \((u^*_k, J^*_{(\delta, \epsilon)}) \leftarrow \text{Solve } P_{(\delta, \epsilon)}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})\)
7: \((\delta^*, \epsilon^*, u^*_k) \leftarrow \text{arg min } J^*_{(\delta, \epsilon)}\)
8: Apply control input \( u_k = u^*_k \) and estimation mode \((\delta_{k+1}, \epsilon_{k+1}) = (\delta^*, \epsilon^*)\)

We prove the following result in the appendix:
Theorem 6.4.1. If at the initial time step there exists a contract value \((\delta, \epsilon) \in \Delta\), an initial state estimate \(\hat{x}_0 \in X\), and an input value \(u_{-1} \in U\), such that \(P(\delta, \epsilon)(\hat{x}_0, \delta_0, \epsilon_0, u_{0-1})\) is feasible then the system (6.2) controlled by Alg. 2 and subjected to disturbances constrained by \(w_k \in W\) robustly satisfies the state constraint \(x \in X\) and the control input constraint \(u \in U\), and all subsequent iterations of the algorithm are feasible.

6.5 Stochastic Model Predictive Control Solution

The control algorithm developed in section 6.4 assumes that the state-estimation error \(e\) lies in a bounded set, \(E\). In practice, this can result in a very conservative approximation. Assuming instead that the error arises from a random distribution allows us to develop a chance constrained formulation for the controller, outlined in this section. We call this control algorithm the Stochastic Adaptive Model Predictive Controller (SAMPC). Here, the constraints on the state have to be satisfied with some probability \(1 - \zeta\), rather than in a deterministic manner as in the RAMPC formulation. This work is presented in detail in Pant et al. [2019].

6.5.1 Solution overview

Starting from the contract-driven control and estimation framework of Sec. 6.2 we denote the profiled delay-error curve of the estimator by \(\Delta\). This curve \(\Delta\) consists of a finite number of contract options \((\delta, \Sigma)\) that the estimator can satisfy at runtime. Here, \(\Sigma \in \mathbb{R}^{n \times n}\) is the positive semi-definite co-variance matrix associated with the now stochastic state-estimation error \(e\). It can be obtained through profiling the performance of the estimator as outlined in Sec. 6.6. We assume that the mean of the estimation errors is zero in all the contracts, but the formulation and analysis that follows also extends to distributions with non-zero means. \(\delta\) is again the computation time the estimator takes in a particular mode of operation.

The SAMPC works in a manner similar to the RAMPC. At each time step \(k\), the controller receives a state estimate \(\hat{x}_k\) and uses it to compute: a) the control signal \(u_k\), as well as b) the contract \((\delta_{k+1}, \Sigma_{k+1}) \in \Delta\) that will be met by the estimator at the following time step. Following this, at time step \(k + 1\) the estimator give a state estimate \(\hat{x}_{k+1}\) with error \(e_{k+1} = \hat{x}_{k+1} - x_{k+1}\) drawn from a distribution with co-variance \(\Sigma_{k+1}\) and within time \(\delta_{k+1}\).

The cost function to be minimized is \(J = \sum_{k=0}^{M}(l(x_k, u_k) + \alpha \pi(\delta_k))\) that combines the tracking error and input power through the \(l\) term and the estimator power consumption through the \(\pi\) terms. The SAMPC control algorithm then finds a sequence of control signals \(u_k\) and the contracts at each time step \((\delta_k, \Sigma_k) \in \Delta\) such that \(J\) is minimized and the state \(x_k\) and input \(u_k\) respect chance constraints of the form:

\[
P([x_k, u_k] \in X \times U) \geq 1 - \zeta \forall k \tag{6.5}
\]
Here, $0 < \zeta \leq 1$ is a design parameter that decides the lower bound on the constraint satisfaction probability. To achieve these objectives, the **Stochastic Adaptive Model Predictive Controller (SAMPC)** aims to solve the following optimization (with horizon $N \geq 1$), denoted by $\bar{P}_\Delta(\hat{x}_k, \delta_k, \Sigma_k, u_{k-1})$, at each time step $k$:

$$
J^*[0 : N] = \min_{u, x : \delta, \Sigma} \sum_{j=0}^{N} (\ell(x_{k+j}, u_{k+j}) + \alpha \pi(\delta_k)) 
$$

s.t. $\forall j \in \{0, \ldots, N\}$

$$
x_{k+j+1} = A x_{k+j} + B_1(\delta_k) u_{k+j-1} + B_2(\delta_k) u_{k+j} 
$$

$$
P([x_k, u_k] \in X \times U) \geq 1 - \zeta
$$

Similar to the RAMPC, the SAMPC needs to find the optimal length-$N$ input sequence $u = (u_k^*, \ldots, u_N^*)$, the corresponding state sequence $x = (x_{k+1}, \ldots, x_{k+N+1})$, the delay sequence $\delta = (\delta_k, \ldots, \delta_{k+N})$ and associated error co-variance sequence $\Sigma = (\Sigma_k, \ldots, \Sigma_{k+N})$ (such that $(\delta_k, \Sigma_k) \in \Delta$) which minimize the the N-step cost $J[0 : N]$ and ensuring the chance constraint of (6.5) is satisfied.

Consistent with regular MPC framework, once a solution $u$ is found, only the first input $u_k$ is applied to the system, resulting in state $x_{k+1}$. At the next time step, after receiving the state estimate $\hat{x}_{k+1}$ from the estimator based on the contract of step $k$, the SAMPC sets up the new optimization $\bar{P}_\Delta(\hat{x}_{k+1}, \delta_{k+1}, \Sigma_{k+1}, u_{k+1})$ and solves it, repeating the process at each subsequent time step.

Similar to RAMPC, the SAMPC only has access to the state estimate, we extend the Stochastic MPC (SMPC) approach in [Kouvarakis et al. 2010](#). Namely, the problem is solved for the nominal dynamics which assume zero process and observation noise ($w_{k+j} = 0$) and zero estimation error ($\hat{x}_{k+j} = x_{k+j}$) over the prediction horizon. Let $X$ be the state of the system under nominal conditions. To compensate for the use of nominal dynamics, SMPC replaces the constraint of (6.5) by $([x_{k+j}, u_{k+j}] \in \tilde{Z}_j(\Sigma_k, \Sigma))$, where $\tilde{Z}_j(\Sigma_k, \Sigma) \subset Z = X \times U$ ‘shrunk’ by an amount corresponding to $\Sigma$, as explained in the appendix. Intuitively, by forcing $([x_{k+j}, u_{k-1+j}]$ to lie in the reduced set $\tilde{Z}_j(\Sigma_k, \Sigma)$, the stochastic estimation error and process noise are guaranteed to be such that that the state and the input respect the joint chance constraint of (6.5).

The tractable optimization for a given $(\delta, \Sigma)$, denoted by $\bar{P}_{(\delta, \Sigma)}(\hat{x}_k, \delta_k, \Sigma_k, u_{k-1})$, is then

$$
J^*_\delta(\Sigma) = \min_{u, x : \delta, \Sigma} \sum_{j=0}^{N} (\ell(\bar{x}_{k+j}, u_{k+j}) + \alpha \pi(\delta_k))
$$

s.t. $\forall j \in \{0, \ldots, N\}$

$$
\bar{x}_{k+j+1} = A \bar{x}_{k+j} + B_1(\delta_k) u_{k+j-1} + B_2(\delta_k) u_{k+j} 
$$

$$
(\bar{x}_{k+j}, u_{k+j}) \in \tilde{Z}_j(\Sigma_k, \Sigma)
$$

77
Construction of the shrunk constraint sets $\tilde{Z}_j$ is covered in the appendix. In practice, we solve the optimization for each $(\delta, \Sigma) \in \Delta$ in parallel and pick the optimal contract and the corresponding control signal as outlined in Algorithm 1 (solving $J^*_{(\delta, \Sigma)}$ instead of $J^*_{(\delta, \epsilon)}$ in this case). The following theorem (proven in the appendix) states the guarantees of this control algorithm:

**Theorem 6.5.1.** For any estimation mode $(\delta, \Sigma)$, if $\tilde{P}_{(\delta, \Sigma)}(\hat{x}_k, \delta_k, \Sigma_k, u_{k-1})$ is feasible then the system (6.2) controlled by the SAMPC and subjected to disturbances constrained by $w_k \in W$ satisfies, with probability at least $1 - \zeta$, the state constraint $x_k \in X$ and control input constraint $u_k \in U$, and the subsequent optimization $\tilde{P}_{(\delta, \Sigma)}(\hat{x}_k, \delta_k, \Sigma_k, u_{k-1})$, are feasible with Probability 1.

### 6.6 Contract based perception algorithms

We presupposed, in Section 6.2, the existence of an Estimation Error vs Computation Delay curve $\Delta$ for the state estimator. The controller uses this curve at each discrete time step to select the operating mode $(\delta, \epsilon)$ for the estimator at the next time step, as seen in Sec. 6.4. In this section, we show how this curve can be obtained for particular applications, as well as ways for the contract based estimation algorithm to realize the points on the curve at runtime.

#### 6.6.1 Profiling and Creating an Anytime Contract Perception-based State Estimation Algorithm

In order to profile a contract estimator, we first need to identify the distinct building blocks (or tasks) of the perception algorithm. Next, we need to find the relevant parameters used in each task, such that varying these parameters results in corresponding changes in the computation time and the quality of the overall output of the estimation algorithm. This can be done, e.g. by varying the number of iterations of a loop [Sidiropoulos-Douskos et al. 2011] such that the resulting computation time $\delta$ and estimation quality $\epsilon$ are different. We refer to these parameters as knobs of the components of the estimation algorithm.

This procedure is tested through implementation on a Computer Vision (CV)-based object detection tool chain, an overview of which is shown in Fig. 6.6. This object recognition tool chain is tasked with tracking an Object of Interest (OOI) across the frames of a video stream. The first level of this is a pixel classifier that assigns a probability for each pixel being a part of the OOI. After thresholding over some minimum probability, we obtain a binary image with the pixels of interest taking a value 1, others being 0. The second level involves denoising the binary image, and then finding the Connected Components (CC), i.e. collecting adjacent pixels of interest into (possibly disconnected) objects. The third and final level is a
Figure 6.6: Illustration of the building blocks used to compose the Contract Object Detector and their representation as real-time tasks. For a given $(\delta, \epsilon)$ contract, knob settings are chosen at run-time resulting in a schedule to execute these sequential components, or tasks, to respect the contract.
shape classifier that is run on the output of the connected components to determine whether each object from it is of interest or not.

Our implementation uses a Gaussian Mixture Model (GMM) classifier as the pixel classifier. The knob here is the number of Gaussian distributions in the GMM. A smaller number of Gaussians will result in a faster, but possibly inaccurate classifier. On the other hand, more Gaussians can result in improved performance, but at the cost of higher computation time. As is typically done, knob values that result in an overfit are identified and rejected via cross-validation during the training process.

The filtering for denoising the binary image, and the Connected Components algorithm form the second level of the object recognition tool chain and the knob here consists of selecting either a 4-connected or 8-connected implementation.

We use a GMM for the shape classifier, but unlike the first level, the knob here is the number of features used to define the shape of the object of interest (e.g. eccentricity, linear eccentricity and major and minor axis lengths for ellipsoidal objects). In this implementation, the number of knob settings for the object recognition tool chain is \( K = (\text{#Gaussians for pixel classifier} \times \text{#neighbors for CC} \times \text{#features for shape classifier}) \), and has a total of \( 3 \times 2 \times 2 = 12 \) values.

The trade-off curve for the entire toolchain is obtained by profiling all 12 knob settings by running it on a data set for profiling. Through this process we obtain, for each of the different knob values: a) the output quality error \( \epsilon \), and b) the computation times \( \delta \) for the entire tool chain. This offline gathering of information gives us the information to be used at run-time in the co-design framework. The profiled performance of the CV-based object recognition toolchain considered here is shown in Fig. 6.7.

It should be noted that for each block of the tool chain, the relation between knob value and quality of output is not necessarily monotonic. The GMM based classifiers must be trained on a data set before deployment and like all machine learning algorithms, their output quality for a given knob setting will depend on the specific data set. This also holds for the output quality of the entire chain, and is reflected in Fig. 6.7 which shows the mean perception error \( \epsilon \) and the 90th percentile execution time for the different knob settings. While the trend is that perception error decreases with increasing execution time, there are some knob settings leading to both larger perception error and larger execution time, which is seen in the non-monotonic behavior seen in Fig. 6.7.

### 6.6.2 Run-time execution of the contract-driven perception algorithm

After profiling the contract-driven estimator, we can use the information at run-time to choose which knob settings are needed to respect a given \((\delta, \epsilon)\) contract. This is tantamount to choosing altered versions of tasks and scheduling them to execute one

---

1 Error is the distance between the true centroid and the estimated centroid of the OOI
after the other in a pre-defined manner to optimally perform the job of detecting an object of interest. Fig. 6.6 shows the various tasks and their different versions for every knob setting and the resulting task schedules.

6.6.3 Visual Odometry

An example of a vision based state estimation algorithm is Semi-Direct Monocular Visual Odometry (SVO) [Forster et al. 2014], which we will use in Sec. 6.7 to get state estimates for control of the hexrotor robot. SVO detects corners in an image, and tracks them across consecutive frames of a video feed in order to localize the moving robot and generate a state estimate. Since this state estimate is used for closed loop control of the hexrotor, SVO has to run in real-time at a frame rate that is fast enough for the purpose of controlling a flying robot. The number of corners \(C\) (as well as their quality) being tracked from frame to frame affects the computation time of the localization algorithm and the resulting quality of the state estimate. In general, assuming that the camera is looking at a feature rich environment, detecting and tracking a higher number of corners results in better localization accuracy but also takes larger computation time. For the profiling of SVO, the number of corners \(C\) is the only knob and is varied to obtain an error-delay curve of the localization performance.

Profiling SVO performance

Fig. 6.8 outlines the profiling process for SVO. We start with the hexrotor, running ROS, flying (either manually or autonomously) in an environment with a Vicon
Figure 6.8: The profiling process to characterize the performance of SVO in terms of estimation error, computation time and power consumption. Sensor and ground truth data is logged from flights of the hexrotor, and then played back and processed offline to generate the error-delay curve (shown in Fig. 6.9) for SVO. The code snippet shows how little modification is needed to the SVO code base to be able to profile its timing characteristics. Through this offline profiling process, we avoid the need of performing separate flights for each knob setting of SVO.

motion-capture system [vic]. Throughout the flight, the downward facing monocular camera captures frames at the desired rate of 20 HZ. We also log the IMU data, as well as the high-accuracy 6-DOF pose estimate generated by the motion capture system, which we will use as the ground truth for the hexrotor positions and velocities. We collected data, recorded as rosbags, over 15 minutes of flights with randomly chosen paths, flown both manually and autonomously.

After collecting the data from our flights, in order to profile the estimation performance of SVO for a particular setting of the number of corners used, we playback this recorded rosbag, accurately recreating the in-flight environment that is present for the visual odometry algorithm. We process the camera frames with SVO running at the desired setting of \#C, and use the SVO generated position estimate along with the corresponding time-stamped IMU data to generate an estimate of the hexrotor’s position and velocity (the hexrotor’s state, see fig. 6.10) at that time instant. By comparing this the state estimate to the VICON measurement at that time instant, we get the state estimation error of SVO. By doing so for the entire recorded data set, we can get the estimation error characteristics of SVO operating at this knob setting. We repeat this process for all knob settings of SVO, going from 50 to 350 corners, and through this get the estimation error profile for SVO across all its operating modes. Fig. 6.8 shows an overlay of the position estimates from SVO (in green) and those from VICON (in red) for a segment of the profiling data set. It also shows a frame captured from the downward facing camera on the hexrotor, and the corners (green dots) that SVO is tracking in that particular frame.

We also need to measure the timing and power consumption of SVO for each knob setting. For the former, we insert C++ code for timing how long SVO takes to process each frame, i.e. the time from receiving a frame from the camera to generating
a position estimate. We do this for each frame in the profiling data set, log this data, and repeat the process for each knob setting of SVO. Fig. 6.17 shows the cumulative distribution function for this computation time across the entire profiling data set, for each knob setting of SVO.

For the power consumption of SVO at different values of the knob #C, we record power measurements made using the Odroid Smart Power meter Odr, which measures consumption to milliwatt precision. By playing back the logged data and running SVO offline for profiling, we avoid the physical challenges of fitting the power meter onto our hexrotor platform and can measure the power consumption of the Odroid board on the ground, while running the workloads as it does during flight. We measure the power consumption of the entire Odroid board, including CPU and DRAM power consumption. Since the profiling of power is done offline with other peripherals plugged into the odroid (e.g. a monitor and keyboard), we measure the idle power of the Odroid and subtract that from the power measurements when the SVO algorithm is running on it in different modes. This gives us a more accurate measure of the workload due to the visual odometry task.

Through this offline profiling process, we avoid having to fly separate flights to get profiling information for every knob setting (#C), and the result of this profiling is used in the formulation of the controller and used at run-time by it to generate contracts for the contract-driven estimator (Fig.6.4).

The error-delay curve for SVO

Obtained from the profiling process outlined above, Fig. 6.9 shows the error-delay curve(s) of the localization error (in positions) of the hexrotor with SVO running on an Odroid-U3 Odr the on-board computation platform of the hexrotor robot. The curve, obtained through data collected over multiple flights in a fixed environment, shows the worst case error \( \epsilon \) (over all flights and all components of the 3D position, used in Sec. 6.4), as well as the the standard deviation of the error for all components of the 3D position (used in the stochastic control formulation of Sec. 6.5) versus the computation time \( \delta \) for varying number of corners being tracked #C. \( \delta \) is obtained by considering the 90th percentile of computation times, while \( \epsilon \) is obtained by computing the infinite norm of the 90th percentile error over the 3 components \((x, y, z)\) of the position. Note that as the number of corners being tracked increases, the computation time increases and the estimation error decreases as expected, but only up to a point. At #C = 250, the estimation error increases. We hypothesize this is due to the decreasing quality of the corners in the environment now being tracked. This is because if the scene is not particularly feature rich, and a sizable fraction of the #C corners are of poor quality (i.e., unstable or hard to track across frames), and we can expect the localization error to increase as the poor quality of the corners detected adds noise to the visual odometry estimates.
Figure 6.9: (Color online) Error-delay curve for the SVO algorithm running on the Odroid-U3 with different settings of maximum number of features (#C) to detect and track. The vertical line shows the cut-off for maximum delay and the SVO settings that are allowable (upto #C = 200) for closed loop control of a hexrotor at 20Hz. No value of #C is used above this as it results in the delay approaching the sampling period of the controller.

6.7 Case Study: Feedback control of a hex-rotor robot

To evaluate the performance of our proposed methods, we implemented the contract-driven estimator and control scheme on a KMel robotics hex-rotor robot [kmel]. The hex-rotor is equipped with a downward facing camera, allowing us to use SVO for localization. The on-board computation platform is an Odroid U-3 [odr] computer running Ubuntu as the operating system. The computer also runs Robot Operating System (ROS) [Quigley et al. 2009] which is responsible for executing the estimation and control algorithm at a fixed rate, as well as the communication between them.


6.7.1 Experimental Setup

Fig. 6.10 shows the feedback control loop and flow of information on-board the hex-rotor. Camera images are processed via SVO to generate position estimates, which are used along with IMU information to generate a 6-Degree of Freedom (linear and angular positions and velocities) pose estimate via Unscented Kalman Filtering. The linear components of this state estimate are used by the position control algorithm (RAMPC). The position controller, tasked with tracking a given reference trajectory, generates desired thrust, roll and pitch to be tracked by the low level attitude controller running at a high-rate. The RAMPC also generates the Delay/Error ($\delta, \epsilon$) contract for the Contract SVO algorithm to respect at the next discrete time step. More details on the experimental setup are in the appendix.

6.7.2 Experiment design

To compare the performance of the RAMPC and SAMPC algorithms developed in this work with that of a MPC that does not leverage co-design, we task the controllers with following two pre-defined reference trajectories, shown in Fig. 6.11. The reference trajectories are generated using the jerk minimizing trajectory generator of Mueller et al. [2015].

1. **The hourglass trajectory**: This trajectory involves flying straight lines between the desired waypoints, as shown in Fig. 6.11. In order to get the straight lines, the waypoints are associated with desired velocities of zero (in each axis). The duration of this trajectory is around 14s. The entire trajectory is flown at
Table 6.1: SVO Modes used in the experiments

<table>
<thead>
<tr>
<th>Mode</th>
<th>#C</th>
<th>δ (ms)</th>
<th>ε (m)</th>
<th>σ(e_x)</th>
<th>σ(e_y)</th>
<th>σ(e_z)</th>
<th>P (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50</td>
<td>24</td>
<td>0.054</td>
<td>0.021</td>
<td>0.033</td>
<td>0.038</td>
<td>778</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>30</td>
<td>0.049</td>
<td>0.019</td>
<td>0.027</td>
<td>0.033</td>
<td>862</td>
</tr>
<tr>
<td>2</td>
<td>150</td>
<td>34</td>
<td>0.041</td>
<td>0.019</td>
<td>0.024</td>
<td>0.030</td>
<td>870</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>38</td>
<td>0.035</td>
<td>0.018</td>
<td>0.022</td>
<td>0.024</td>
<td>951</td>
</tr>
</tbody>
</table>

a constant height of 1m. A video of the hex-rotor flying this trajectory can be found at https://youtu.be/-ltJ02gVxWs

2. Spiral in x, y with sinusoidal variations in z: This trajectory consists of smooth curves between waypoints, with the waypoints such that in the x,y plane the trajectory looks like a spiral converging towards the origin, while in the z-axis it consists of sinusoidal variations along a reference height of 1m. The duration of this trajectory is 17s. A video of the hex-rotor flying this trajectory is at https://youtu.be/hmTRxrq4NJg

These trajectories are flown with: a) the baseline, a Robust MPC formulation that does not leverage the co-design of computation and control, with all four chosen modes of SVO used for the state feedback, b) the RAMPC algorithm with varying values of $\alpha$, the weight for the computation power in the optimization, c) the SAMPC (with $\zeta = 0.82$) with varying values of $\alpha$. Each trajectory is flown twice for each one of these settings to get a comparison of control performance and computation energy consumption. This lead to a total of 56 flights to gather the data presented in this case study.

6.7.3 Experimental Results

To measure the performance of the controllers in a standardized manner, we used the following measure of control performance:

$$J_{true} = \frac{1}{T_{max}} \frac{T_{max}}{h} \sum_{k=0}^{T_{max}/h} (x_k - x_k^{ref})^T Q (x_k - x_k^{ref}) + u_k^T R u_k$$

(6.8)

Here, $x^{ref}$ is the desired trajectory, and $Q$ and $R$ are the matrices used in the cost of MPC/RAMPC/SAMPC, $h$ is the sampling time (50ms) and $T_{max}$ is the duration of the particular trajectory flown. $J_{true}$ can be accurately evaluated as we have access to the true state, $x_k$, of the hex-rotor from the Vicon system.

Comparison to the baseline

Fig. 6.12 shows the control performance and the SVO energy consumption for the hourglass trajectory for the baseline RMPC, RAMPC, and SAMPC for different settings. The SAMPC and RAMPC result in lower (average across flights) values of
Figure 6.11: The two reference trajectories, the spiral is in dashed red and the hourglass is in solid black (color in online version). The figure on the right shows the trajectories projected on the x,y plane. Note, the spiral starts on the outside and ends inwards while the hourglass trajectory starts and ends at (0,0,1).

$J_{\text{true}}$ than the baseline controller, i.e. better control performance. As the value of $\alpha$ increases, the power consumption decreases and the control performance degrades for the RAMPC and SAMPC. This is expected as $\alpha$ is the weight for the computation power in the overall optimization cost of (6.3) (and (6.6)) and increasing it would make computation power more important relative to the control performance. Fig. 6.13 shows a similar behavior for the spiral trajectory. The notable exception is in the baseline performance, where the most accurate mode (mode 3) of SVO does not result in the best control performance of the fixed mode RMPC controller. This is possibly because the spiral trajectory is more aggressive than the hourglass trajectory, which involves stopping at each corner waypoint of the trajectory, and spending time in mode 3 comes with a computation delay that degrades the control performance despite the increases accuracy of the state estimate. It should be noted that for either trajectory, SAMPC and RAMPC give a better control performance than the baseline for the corresponding computation energy consumption. For both cases,
Figure 6.12: Performance, hourglass trajectory. The vertical axis has the average control performance (eq. 6.8) over the flights for the labeled settings, with lower values implying better control performance. The horizontal axis shows the computation power (in Joules) consumed by SVO to perform the state estimation task. The figure shows how our methods (RAMPC/SAMPC) leveraging the co-design have both better control performance while consuming less computation power than the baseline method.

the control performance of SAMPC and RAMPC are close to each other, with the SAMPC slightly outperforming the RAMPC for the spiral trajectory.

Summary: Across both the trajectories, the best case control performance of our methods results in about a 10% improvement compared to that of the baseline. To achieve this performance, our methods result in SVO using about 5 − 6% and less computation energy compared to the baseline (at the setting resulting in best control performance). This clearly demonstrates the benefit of the co-design between the perception-based estimation and the control algorithms.

Impact of the weight for computation power ($\alpha$)

As $\alpha$ takes on a high value, the control performance of RAMPC and SAMPC for the hourglass trajectory approaches that of the baseline RAMPC with SVO mode fixed.
Figure 6.13: Performance, spiral trajectory. The vertical axis has the average control performance (eq. (6.8)) over the flights for the labeled settings, with lower values implying better control performance. The horizontal axis shows the computation power (in Joules) consumed by SVO to perform the state estimation task. Similar to the case for the hourglass trajectory, our methods outperform the baseline.

to 0. This is backed up the observation of tables 6.2, 6.3 which show that for \( \alpha = 1 \), the RAMPC and SAMPC select mode 0, the low-power but high estimation error mode, of SVO all the time. The tables 6.2, 6.3 show the fraction of time spent in each mode of SVO as \( \alpha \) changes. Note that as \( \alpha \), the weight for the computation power, increases the time spent in the low power mode 0 also increases while the time spent in the more accurate but higher power modes accordingly decreases. Similar behavior is noted for the spiral trajectory, and tables 6.4 and 6.5 show the fraction of time spent in the different SVO modes as \( \alpha \) changes for RAMPC and SAMPC flying the spiral trajectory respectively.

**Snapshots of the control performance of RAMPC and SAMPC**

Fig. 6.14 shows the reference and actual positions of the hex-rotor (in x, y and z coordinates) as function of time for the hourglass trajectory controlled by the SAMPC.
Figure 6.14: (Color online) Reference positions (dashed red) and actual positions (blue) of the hex-rotor flying the hourglass trajectory while being controlled by the SAMPC ($\alpha = 0$). Note the near perfect tracking in $x$ and $y$. The small dip in the height ($z$ co-ordinate) is due to combination of model error (due to inaccuracy of the mass) as well the effect of linearization around hover. Fig. [6.15] shows the reference and actual positions versus time for the RAMPC ($\alpha = 0.1$) flying the spiral trajectory, showing similarly good tracking performance as in the hourglass trajectory.

Finally, Fig. [6.16] shows the selected mode of the SVO (with SAMPC, for the spiral trajectory flown by the SAMPC ($\alpha = 0.001$) changing over the discrete time steps, as well as the evolution of the tracking cost at each time step.

6.8 Conclusion

In this chapter we presented a contract-driven methodology for co-design of estimation and control for autonomous systems. The basic idea is that the control algorithm requests a delay and estimation error ($\delta, \epsilon$) contract that the perception-and-estimation algorithm realizes. The control algorithm we designed aims to set time-varying contracts to maximise a performance function while respecting feasibility constraints and
Table 6.2: Fraction of time spent in modes: Hourglass trajectory, RAMPC

<table>
<thead>
<tr>
<th></th>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0$</td>
<td>0.398</td>
<td>0.008</td>
<td>0.024</td>
<td>0.570</td>
</tr>
<tr>
<td>$\alpha = 0.001$</td>
<td>0.523</td>
<td>0.004</td>
<td>0.024</td>
<td>0.440</td>
</tr>
<tr>
<td>$\alpha = 0.01$</td>
<td>0.557</td>
<td>0.000</td>
<td>0.067</td>
<td>0.374</td>
</tr>
<tr>
<td>$\alpha = 0.1$</td>
<td>0.820</td>
<td>0.000</td>
<td>0.055</td>
<td>0.123</td>
</tr>
<tr>
<td>$\alpha = 1$</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 6.3: Fraction of time spent in modes: Hourglass trajectory, SAMPC

<table>
<thead>
<tr>
<th></th>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0$</td>
<td>0.374</td>
<td>0.000</td>
<td>0.004</td>
<td>0.621</td>
</tr>
<tr>
<td>$\alpha = 0.001$</td>
<td>0.514</td>
<td>0.016</td>
<td>0.051</td>
<td>0.418</td>
</tr>
<tr>
<td>$\alpha = 0.01$</td>
<td>0.617</td>
<td>0.000</td>
<td>0.032</td>
<td>0.351</td>
</tr>
<tr>
<td>$\alpha = 0.1$</td>
<td>0.793</td>
<td>0.000</td>
<td>0.076</td>
<td>0.131</td>
</tr>
<tr>
<td>$\alpha = 1$</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 6.4: Fraction of time spent in modes: Spiral trajectory, RAMPC

<table>
<thead>
<tr>
<th></th>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0$</td>
<td>0.381</td>
<td>0.015</td>
<td>0.015</td>
<td>0.589</td>
</tr>
<tr>
<td>$\alpha = 0.001$</td>
<td>0.422</td>
<td>0.012</td>
<td>0.018</td>
<td>0.548</td>
</tr>
<tr>
<td>$\alpha = 0.01$</td>
<td>0.504</td>
<td>0.000</td>
<td>0.041</td>
<td>0.455</td>
</tr>
<tr>
<td>$\alpha = 0.1$</td>
<td>0.680</td>
<td>0.000</td>
<td>0.082</td>
<td>0.238</td>
</tr>
<tr>
<td>$\alpha = 1$</td>
<td>0.995</td>
<td>0.000</td>
<td>0.015</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 6.5: Fraction of time spent in modes: Spiral trajectory, SAMPC

<table>
<thead>
<tr>
<th></th>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0$</td>
<td>0.396</td>
<td>0.003</td>
<td>0.018</td>
<td>0.584</td>
</tr>
<tr>
<td>$\alpha = 0.001$</td>
<td>0.434</td>
<td>0.009</td>
<td>0.018</td>
<td>0.540</td>
</tr>
<tr>
<td>$\alpha = 0.01$</td>
<td>0.531</td>
<td>0.000</td>
<td>0.038</td>
<td>0.431</td>
</tr>
<tr>
<td>$\alpha = 0.1$</td>
<td>0.695</td>
<td>0.000</td>
<td>0.073</td>
<td>0.232</td>
</tr>
<tr>
<td>$\alpha = 1$</td>
<td>0.971</td>
<td>0.000</td>
<td>0.029</td>
<td>0.000</td>
</tr>
</tbody>
</table>
Figure 6.15: (Color online) Reference positions (dashed red) and actual positions (blue) of the hex-rotor flying the spiral trajectory while being controlled by the RAMPC ($\alpha = 0.1$).

stability under the time varying execution delay and estimation error from the estimator. We also illustrate how the contract-driven perception-and-estimation algorithm is designed offline and used at run-time to best meet the $(\delta, \epsilon)$ contracts set for it. Through a case study on a flying hexrotor, we showed the applicability of our scheme to real-time closed loop system. The experimental results show the good performance of our scheme and how it outperforms regular Model Predictive Control which does not leverage co-design. A key result showed how our closed loop solution is more energy efficient than MPC while achieving better tracking performance. A focus of ongoing research is to overcome the necessity of the contracts always being met by the estimator. Another focus is on an automated tool chain to profile perception algorithms commonly used in autonomous systems.
Figure 6.16: SVO Mode and control cost over time for the spiral trajectory flown with SAMPC at $\alpha = 0.001$.

6.9 Proofs of the main results

6.10 The Robust case

In this appendix we give the detailed mathematical derivation of the results of Section III. The controller is designed using a Robust Model Predictive Control (RMPC) approach via constraint restriction [Richards and How 2005b], [Chisci et al. 2001], and augments it by an adaptation to the error-delay curve of the estimator. In order to ensure robust safety and feasibility, the key idea of the RMPC approach is to tighten the constraint sets iteratively to account for possible effect of the disturbances. As time progresses, this “robustness margin” is used in the MPC optimization with the nominal dynamics, i.e., the original dynamics where the disturbances are either removed or replaced by nominal disturbances. Because only the nominal dynamics are used, the complexity of the optimization is the same as for the nominal problem.

Since the controller only has access to the estimated state $\hat{x}$, we need to rewrite the plant’s dynamics with respect to $\hat{x}$. The error between $x_k$ and $\hat{x}_k$ is $e_k = x_k - \hat{x}_k$. 

93
At time step $k + 1$ we have
\[
\hat{x}_{k+1} = x_{k+1} - e_{k+1} = Ax_k + B_1(\delta_k)u_{k-1} + B_2(\delta[k])u_k + w_k - e_{k+1},
\]
then, by writing $x_k = \hat{x}_k + e_k$, we obtain the dynamics
\[
\hat{x}_{k+1} = A\hat{x}_k + B_1(\delta[k])u_{k-1} + B_2(\delta[k])u_k + \hat{w}_k
\] (6.9)
where $\hat{w}_k = w_k + Ae_k - e_{k+1}$. The set of possible values of $\hat{w}_k$ depends on the estimation accuracy at steps $k$ and $k + 1$ and is denoted by $\hat{W}(\epsilon[k], \epsilon[k + 1])$, i.e.,
\[
\hat{W}(\epsilon, \epsilon') := \{w + Ae - e' \mid w \in \mathcal{W}, e \in \mathcal{E}(\epsilon), e' \in \mathcal{E}(\epsilon')\}. \quad \text{Note that } \hat{W}(\epsilon[k], \epsilon[k + 1]) \text{ is independent of the time step } k.
\]
It can be computed as $\hat{W}(\epsilon, \epsilon') = \mathcal{W} \oplus A\mathcal{E}(\epsilon) \oplus (-\mathcal{E}(\epsilon'))$ where the symbol $\oplus$ denotes the Minkowski sum of two sets.

The dynamics in (6.9) have a non-standard form where it depends on both the current and the previous control inputs. However we can expand the state variable to store the previous control input as
\[
\hat{z}_k = \begin{bmatrix} \hat{x}_k \\ u_{k-1} \end{bmatrix} \in \mathbb{R}^{n+m}
\]
and rewrite the dynamics as, for all $k \geq 0$,
\[
\hat{z}_{k+1} = \hat{A}(\delta_k)\hat{z}_k + \hat{B}(\delta_k)u_k + \hat{F}\hat{w}_k
\] (6.10)
Here, the system matrices are
\[
\hat{A}(\delta_k) = \begin{bmatrix} A & B_1(\delta_k) \\ 0_{m \times n} & 0_{m \times m} \end{bmatrix}, \quad \hat{B}(\delta_k) = \begin{bmatrix} B_2(\delta_k) \\ I_m \end{bmatrix}, \quad \hat{F} = \begin{bmatrix} I_n \\ 0_{m \times n} \end{bmatrix}.
\] (6.11)

Let the actual expanded state be $z_k = [x_k^T, u_{k-1}^T]^T$. Because the expanded state consists of both the plant’s state and the previous control input, the state constraint $x_k \in X$ and the control constraint $u_k \in U$ are equivalent to the joint constraint $z_k \in X \times U$. We can now describe the RAMPC algorithm for the dynamics in (6.10).

### 6.10.1 Tractable RAMPC Algorithm

Let $N \geq 1$ be the horizon length of the RMPC optimization. Because the system matrices in the state equation (6.10) depend nonlinearly on the variables $\delta_k$, the RMPC optimization is generally a mixed-integer nonlinear program, which is very hard to solve. To simplify the RMPC optimization to make it tractable, we fix the estimation mode for the entire RMPC horizon.
Let $P_{(\delta, \epsilon)}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})$ denote the RMPC optimization problem at step $k \geq 0$ where the current state estimate is $\hat{x}_k$, the current estimation mode is $(\delta_k, \epsilon_k) \in \Delta$, the previous control input is $u_{k-1}$, and the estimation mode for the entire horizon (after step $k$) is fixed at $(\delta, \epsilon) \in \Delta$. Since the system matrices become constant now, if the stage cost $\ell(\cdot)$ is linear or positive semidefinite quadratic, each optimization problem $P_{(\delta, \epsilon)}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})$ is tractable and can be solved efficiently as we will show later. The RAMPC algorithm with Anytime Estimation is stated in Alg. 1.

### 6.10.2 RMPC Formulation

We formulate the RMPC optimization $P_{(\delta, \epsilon)}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})$ with respect to the nominal dynamics, which is the original dynamics in Eq. (6.10) but the disturbances are either removed or replaced by nominal disturbances. To ensure robust feasibility and safety, the state constraint set is tightened after each step using a candidate stabilizing state feedback control, and a terminal constraint is derived. In this RMPC formulation, we extend the approach in Richards and How [2005b], Chisci et al. [2001].

At time step $k$, given $(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})$ and for a fixed $(\delta, \epsilon)$, we solve the following optimization

$$J^*_{\delta, \epsilon}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1}) = \min_{u, x} \sum_{j=0}^{N} \ell(\bar{x}_{k+j|k}, u_{k+j|k})$$  \hspace{1cm} (6.12a)

subject to, $\forall j \in \{0, \ldots, N\}$

$$\bar{z}_{k+j+1|k} = A(\delta_{k+j|k})\bar{z}_{k+j|k} + B(\delta_{k+j|k})u_{k+j|k}$$  \hspace{1cm} (6.12b)

$$(\delta_{k+j|k}, \epsilon_{k+j+1|k}) = (\delta, \epsilon)$$  \hspace{1cm} (6.12c)

$$(\delta_{k|k}, \epsilon_{k|k}) = (\delta_k, \epsilon_k)$$  \hspace{1cm} (6.12d)

$$\bar{x}_{k+j|k} = \left[ I_n \hspace{1cm} 0_{n \times m} \right] \bar{z}_{k+j|k}$$  \hspace{1cm} (6.12e)

$$\bar{z}_{k+j|k} = \left[ \hat{x}_k^T, u_{k-1}^T \right]^T$$  \hspace{1cm} (6.12f)

$$\bar{z}_{k+j|k} \in \mathcal{Z}_j(\epsilon_k, \epsilon)$$  \hspace{1cm} (6.12g)

in which $\bar{z}$ and $\bar{x}$ are the variables of the nominal dynamics. The constraints of the optimization are explained below.

- **(6.12b)** is the nominal dynamics.
- **(6.12c)** states that the estimation mode is fixed at $(\delta, \epsilon)$ except for the first time step when it is $(\delta_k, \epsilon_k)$.
- **(6.12d)** extracts the nominal state $\bar{x}$ of the plant from the nominal expanded state $z$. 

95
• (6.12e) initializes the nominal expanded state at time step $k$ by stacking the current state estimate and the previous control input.

• (6.12f) tightens the admissible set of the nominal expanded states by a sequence of shrinking sets.

• (6.12g) constrains the terminal expanded state to the terminal constraint set $Z_f$.

The state constraint $Z_j$: The tightened state constraint sets $Z_j(\epsilon_k, \epsilon)$ are parameterized with two parameters $\epsilon_k$ and $\epsilon$. They are defined as follows, for all $j \in \{0, \ldots, N\}$

\[
Z_0(\epsilon_k, \epsilon) = Z \ominus \hat{F}E(\epsilon_k) \quad (6.13)
\]

\[
Z_{j+1}(\epsilon_k, \epsilon) = Z_j(\epsilon, \epsilon) \ominus L_j \hat{F} \hat{W}(\epsilon_k, \epsilon) \quad (6.14)
\]

in which the symbol $\ominus$ denotes the Pontryagin difference between two sets. The set $Z$ combines the constraints for both the plant’s state and the control input: $Z = X \times U$. The matrix $L_j$ is the state transition matrix for the nominal dynamics in (6.12b) under a candidate state feedback gain $K_j(\delta)$, for $j \in \{0, \ldots, N\}$

\[
L_0 = \mathbb{I} \quad (6.15)
\]

\[
L_{j+1} = (\hat{A}(\delta) + \hat{B}(\delta)K_j(\delta))L_j \quad (6.16)
\]

Note that the possibly time-varying sequence $K_j(\delta)$ is designed for each choice of $\delta$ (i.e., the system matrices $\hat{A}(\delta)$ and $\hat{B}(\delta)$), hence $L_j$ depends on $\delta$; however we write $L_j$ for brevity. The candidate control $K_j(\delta)$ is designed to stabilize the nominal system (6.12b), desirably as fast as possible so that the sets $Z_j$ are shrunk as little as possible. In particular, if $K_j(\delta)$ renders the nominal system nilpotent after $M < N$ steps then $L_j = \mathbf{0}$ for all $j \geq M$, therefore $Z_j(\epsilon_k, \epsilon) = Z_M(\epsilon_k, \epsilon)$ for all $j > M$.

The terminal constraint $Z_f$: $Z_f$ is given by

\[
Z_f(\epsilon, \epsilon) = C(\delta, \epsilon) \ominus L_N \hat{F} \hat{W}(\epsilon_k, \epsilon) \quad (6.17)
\]

where $C(\delta, \epsilon)$ is a robust control invariant admissible set for $\delta$ [Kerrigan 2000], i.e., there exists a feedback control law $u = \kappa(z)$ such that $\forall z \in C(\delta, \epsilon)$ and $\forall w \in \hat{W}(\epsilon, \epsilon)$

\[
\hat{A}(\delta)z + \hat{B}(\delta)\kappa(z) + L_N \hat{F}w \in C(\delta, \epsilon) \quad (6.18)
\]

\[
z \in Z_N(\epsilon, \epsilon) \quad (6.19)
\]

We remark that $C(\delta, \epsilon)$ does not depend on $(\delta_k, \epsilon_k)$, therefore it can be computed offline for each mode $(\delta, \epsilon)$. 96
6.10.3 Proofs of Feasibility

The RMPC formulation of the previous section, with a fixed estimation mode \((\delta, \epsilon) \in \Delta\), is designed to ensure that the control problem is robustly feasible, as stated in the following theorem.

**Theorem 6.10.1** (Robust Feasibility of RAMPC). For any estimation mode \((\delta, \epsilon)\), if \(\mathbb{P}_{(\delta, \epsilon)}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})\) is feasible then the system (2) controlled by the RAMPC and subjected to disturbances constrained by \(w_k \in \mathcal{W}\) robustly satisfies the state constraint \(x_k \in X\) and the control input constraint \(u_k \in U\), and all subsequent optimizations \(\mathbb{P}_{\delta, \epsilon}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1}), \forall k > k_0\), are feasible.

**Proof.** We will prove the theorem by recursion. We will show that if at any time step \(k\) the RMPC problem \(\mathbb{P}_{\delta, \epsilon}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})\) is feasible and feasible control input \(u_k = u^*_k\) is applied with estimation mode \((\delta_k+1, \epsilon_k+1) = (\delta, \epsilon)\) then \(u_k\) is admissible and at the next time step \(k+1\), the actual plant’s state \(x_{k+1}\) is inside \(X\) and the optimization \(\mathbb{P}_{\delta, \epsilon}(\hat{x}_{k+1}, \delta_k+1, \epsilon_k+1, u_k)\) is feasible for all disturbances. Then we can conclude the theorem because, by recursion, feasibility at time step \(k_0\) implies robust constraint satisfaction and feasibility at time step \(k_0 + 1\), and so on at all subsequent time steps.

Suppose \(\mathbb{P}_{\delta, \epsilon}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})\) is feasible. Then it has a feasible solution

\[
\{\{z^*_{k+j|k}\}_{j=0}^{N+1}, \{u^*_{k+j|k}\}_{j=0}^{N}\}
\]

that satisfies all the constraints in (6.12). Now we will construct a feasible candidate solution for \(\mathbb{P}_{\delta, \epsilon}(\hat{x}_{k+1}, \delta_k+1, \epsilon_k+1, u_{k})\) at the next time step by shifting the above solution by one step. Consider the following candidate solution for \(\mathbb{P}_{\delta, \epsilon}(\hat{x}_{k+1}, \delta_k+1, \epsilon_k+1, u_{k})\):

\[
\begin{align*}
z_{k+j+1|k+1} &= z^*_{k+j+1|k} + L_j \hat{F}_k w_k \quad \text{(6.20a)} \\
z_{k+N+2|k+1} &= \hat{A}(\delta) z_{k+N+1|k+1} + \hat{B}(\delta) u_{k+N+1|k+1} \quad \text{(6.20b)} \\
u_{k+i+1|k+1} &= u^*_{k+i+1|k+1} + K_i(\delta)L_i \hat{F}_k w_k \quad \text{(6.20c)} \\
u_{k+N+1|k+1} &= \kappa(z_{k+N+1|k+1}) \quad \text{(6.20d)}
\end{align*}
\]

in which \(j \in \{0, \ldots, N\}, i \in \{0, \ldots, N - 1\}\), and \(\kappa(\cdot)\) is the feedback control law for the invariant set \(C(\delta, \epsilon)\) that is used in the terminal set. We first show that the input and state constraints are satisfied for \(u_k\) and \(x_{k+1}\), then we will prove the feasibility of the above candidate solution for \(\mathbb{P}_{\delta, \epsilon}(\hat{x}_{k+1}, \delta_k+1, \epsilon_k+1, u_{k})\).

**Validity of the applied input and the next state:** The next plant’s state is

\[
x_{k+1} = A x_k + B_1(\delta[k]) u_{k-1} + B_2(\delta[k]) u_k + w_k
\]

\[
= A (\hat{x}_k + e_k) + B_1(\delta[k]) u_{k-1} + B_2(\delta[k]) u^*_k + w_k
\]

\[
= \left[ A \quad B_1(\delta[k]) \right] \begin{bmatrix} \hat{x}_k \\ u_{k-1} \end{bmatrix} + B_2(\delta[k]) u^*_k + e_{k+1} + (w_k + A e_k - e_{k+1})
\]

97
in which \( e_{k+1} \in \mathcal{E}(\epsilon) \) and \((w_k + Ae_k - e_{k+1}) \in \hat{\mathcal{W}}(\epsilon[k], \epsilon)\). Note that \( \tilde{z}^*_k = [\hat{x}^T_k, u^T_{k-1}]^T \).

Hence we have

\[
\begin{bmatrix}
x_{k+1} \\
u_k
\end{bmatrix} = \hat{A}(\delta[k])\tilde{z}^*_k + \hat{B}(\delta[k])u^*_k
+ \hat{F}e_{k+1} + \hat{F}(w_k + Ae_k - e_{k+1})
= \tilde{z}^*_{k+1|k} + \hat{F}e_{k+1} + \hat{F}(w_k + Ae_k - e_{k+1})
\]

where we use the dynamics in (6.12b). From (6.12b) and (6.14), \( \tilde{z}^*_{k+1|k} \) satisfies \( \tilde{z}^*_{k+1|k} \in \mathcal{Z}_1(\epsilon[k], \epsilon) = \mathcal{Z} \otimes \hat{\mathcal{F}} \mathcal{E}(\epsilon) \otimes \hat{\mathcal{F}} \hat{\mathcal{W}}(\epsilon[k], \epsilon) \). It follows that \( [x^T_{k+1}, u^T_k]^T \in Z = X \times U \), therefore \( x_{k+1} \in X \) and \( u_k \in U \).

**Initial condition:** We have from (6.10) that \( \hat{z}_{k+1} = \hat{A}(\delta[k])\hat{z}_k + \hat{B}(\delta[k])u_k + \hat{F}\hat{w}_k \). On the other hand, by (6.20a),

\[
\tilde{z}_{k+1|k+1} = \tilde{z}^*_{k+1|k} + L_0\hat{F}\hat{w}_k
= \hat{A}(\delta[k])\tilde{z}^*_k + \hat{B}(\delta[k])u^*_k + L_0\hat{F}\hat{w}_k.
\]

Note that \( \tilde{z}^*_k = \hat{z}_k, u_k = u^*_k, \) and \( L_0 = \mathbb{I} \). Therefore \( \tilde{z}^*_{k+1|k+1} = \hat{z}_{k+1} \), hence the initial condition is satisfied.

**Dynamics:** We show that the candidate solution satisfies the dynamics constraint in Eq. (6.12b). For \( 0 \leq j < N \) we have

\[
\begin{aligned}
\tilde{z}^*_{k+j+2|k+1} &= \tilde{z}^*_{k+j+2|k} + L_{j+1}\hat{F}\hat{w}_k \\
&= \hat{A}(\delta)\tilde{z}^*_{k+j+1|k} + \hat{B}(\delta)u^*_{k+j+1|k} + L_{j+1}\hat{F}\hat{w}_k \\
&= \hat{A}(\delta)\left(\tilde{z}^*_{k+j+1|k+1} - L_j\hat{F}\hat{w}_k\right) \\
&\quad + \hat{B}(\delta)\left(u_{k+j+1|k+1} - K_j(\delta)L_j\hat{F}\hat{w}_k\right) + L_{j+1}\hat{F}\hat{w}_k \\
&= \hat{A}(\delta)\tilde{z}^*_{k+j+1|k+1} + \hat{B}(\delta)u_{k+j+1|k+1} \\
&\quad - \left(\hat{A}(\delta) + \hat{B}(\delta)K_j(\delta)\right)L_j\hat{F}\hat{w}_k + L_{j+1}\hat{F}\hat{w}_k \\
&= \hat{A}(\delta)\tilde{z}^*_{k+j+1|k+1} + \hat{B}(\delta)u_{k+j+1|k+1}
\end{aligned}
\]

where the equality in (6.16) is used to derive the last equality. Therefore the dynamics constraint is satisfied for all \( 0 \leq j < N \). For \( j = N \), the constraint is satisfied by construction by (6.20b).

**State constraints:** We need to show that \( \tilde{z}_{(k+1)+j|k+1} \in \mathcal{Z}_j(\epsilon, \epsilon) \) for all \( j \in \{0, \ldots, N\} \). Consider any \( 0 \leq j < N \). (6.14) states that \( \mathcal{Z}_{j+1}(\epsilon[k], \epsilon) = \mathcal{Z}_j(\epsilon, \epsilon) \otimes L_j\hat{F}\hat{\mathcal{W}}(\epsilon[k], \epsilon) \).

From the construction of the candidate solution we have \( \tilde{z}_{k+j+1|k+1} = \tilde{z}^*_{k+j+1|k} + L_j\hat{F}\hat{w}_k \), where \( \hat{w}_k \in \hat{\mathcal{W}}(\epsilon[k], \epsilon) \) and \( \tilde{z}^*_{k+j+1|k} \in \mathcal{Z}_{j+1}(\epsilon[k], \epsilon) \). By definition of the Pontryagin difference, we conclude that \( \tilde{z}_{k+j+1|k+1} \in \mathcal{Z}_j(\epsilon, \epsilon) \) for all \( j \in \{0, \ldots, N - 1\} \).
At \( j = N \) the candidate solution in (6.20a) gives us \( \overline{z}_{(k+1)+N|k+1} = \overline{z}_{k+N+1|k}^* + L_N \hat{F} \hat{w}_k \). Because \( \overline{z}_{k+N+1|k}^* \in \mathcal{Z}_f (\epsilon[k], \epsilon) = C (\delta, \epsilon) \ominus L_N \hat{F} \hat{W} (\epsilon[k], \epsilon) \) and \( \hat{w}_k \in \tilde{W} (\epsilon[k], \epsilon) \), we have \( \overline{z}_{(k+1)+N|k+1} \in C (\delta, \epsilon) \). The definition of \( C (\delta, \epsilon) \) in (6.18) implies \( C (\delta, \epsilon) \subseteq Z_N (\epsilon, \epsilon) \). Therefore \( \overline{z}_{(k+1)+N|k+1} \in Z_N (\epsilon, \epsilon) \).

**Terminal constraint:** We need to show that \( \overline{z}_{k+N+2|k+1} \in \mathcal{Z}_f (\epsilon, \epsilon) = C (\delta, \epsilon) \ominus L_N \hat{F} \hat{W} (\epsilon, \epsilon) \). Add \( L_N \hat{F} \hat{w} \), for any \( \hat{w} \in \hat{W} (\epsilon, \epsilon) \), to both sides of (6.20b) and note that \( u_{k+N+1|k+1} = \kappa (\overline{z}_{k+N+1|k+1}) \), we have

\[
\overline{z}_{k+N+2|k+1} + L_N \hat{F} \hat{w} = \hat{A} (\delta) \overline{z}_{k+N+1|k+1} + \hat{B} (\delta) \kappa (\overline{z}_{k+N+1|k+1}) + L_N \hat{F} \hat{w}.
\]

It follows from \( \overline{z}_{k+N+1|k+1} \in C (\delta, \epsilon) \) and from the definition of the invariant control invariant admissible set \( C (\delta, \epsilon) \) (Eq. (6.18)) that \( \overline{z}_{k+N+2|k+1} + L_N \hat{F} \hat{w} \in C (\delta, \epsilon) \) for all \( w \in \hat{W} (\epsilon, \epsilon) \). Then by definition of the Pontryagin difference, we conclude that \( \overline{z}_{k+N+2|k+1} \in C (\delta, \epsilon) \ominus L_N \hat{F} \hat{W} (\epsilon, \epsilon) = \mathcal{Z}_f (\epsilon, \epsilon) \).

The control algorithm in Alg. 1, in each time step \( k \), solves \( \mathbb{P}_{(\delta, \epsilon)} (\hat{x}_k, \delta_k, \epsilon_k, u_{k-1}) \) for each estimation mode \( (\delta, \epsilon) \in \Delta \) and selects the control input \( u_k \) and the next estimation mode \( (\delta_{k+1}, \epsilon_{k+1}) \) corresponding to the best total cost \( J_{(\delta, \epsilon)} \). Therefore, during the course of control, the algorithm may switch between the estimation modes in \( \Delta \) depending on the system’s state. Thm. 6.10.2 states that if the control algorithm Alg. 1 is feasible in its first time step then it will be robustly feasible and the state and control input constraints are also robustly satisfied.

**Theorem 6.10.2.** If at the initial time step there exists \( (\delta, \epsilon) \in \Delta \) such that \( \mathbb{P}_{(\delta, \epsilon)} (\hat{x}_0, \delta_0, \epsilon_0, u_{0-1}) \) is feasible then the system Eq. (6.9) controlled by Alg. 1 and subjected to disturbances constrained s.t. \( w_k \in \mathcal{W}, \forall k \geq 0 \) robustly satisfies the state constraint \( x_k \in X, \forall k \geq 0 \) and the control input constraint \( u_k \in U, \forall k \geq 0 \), and all subsequent iterations of the algorithm are feasible.

**Proof.** The Theorem can be proved by recursively applying Thm. 6.10.1. Indeed, suppose at time step \( k \) the algorithm is feasible and results in control input \( u_k \) and next estimation mode \( (\delta_{k+1}, \epsilon_{k+1}) \), then \( \mathbb{P}_{(\delta_{k+1}, \epsilon_{k+1})} (\hat{x}_k, \delta_k, \epsilon_k, u_{k-1}) \) is feasible.

By Theorem 6.10.1 \( u_k \in U \) and at the next time step \( k+1 \), \( x_{k+1} \in X \) and \( \mathbb{P}_{(\delta_{k+1}, \epsilon_{k+1})} (\hat{x}_{k+1}, \delta_{k+1}, \epsilon_{k+1}, u_{k-1}) \) is also feasible, hence the algorithm is feasible. Therefore, the Theorem holds by induction.

### 6.11 The Stochastic case

When the estimation errors are drawn from a distribution, the contracts for the perception algorithm are of the form \((\delta, \Sigma)\) (computation time and estimation error...
covariance respectively, assume 0 mean distributions w.l.o.g. In the following section we consider the case where the estimation errors come from a general distribution (with bounded second moment) and have bounded support.

The main results are summarized in the following theorem (restated here):

**Theorem 6.11.1.** For any estimation mode \((\delta, \Sigma)\), if \(P_{(\delta, \epsilon)}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})\) is feasible then the system \((2)\) controlled by the RAMPC and subjected to disturbances constrained by \(w_k \in W\) satisfies, with probability at least \(1 - \zeta\), the state constraint \(x_k \in X\) and the control input constraint \(u_k \in U\), and the subsequent optimization \(P_{\delta, \epsilon}(\hat{x}_{k+1}, \delta[k], \epsilon[k], u_k)\), are feasible with Probability 1.

We begin with a candidate solution similar to the one from the robust (worst case) Anytime MPC case, i.e. \((6.20)\). Since the proofs are very similar in nature to those in the robust case, we will build on top of those existing proofs, dropping subscripts for mode, time step, and constraint number where necessary for ease of notation.

### 6.11.1 Constraint tightening

Here, we assume that the estimation error \(e\) comes from a distribution with a known bounded variance and a known mean (set to 0 w.l.o.g) for each mode of the perception-based estimator \((\delta, \Sigma)\). For the sake of simplicity, we assume that the process noise \(w\) is also such a distribution and has a bounded support.

Starting from a chance constraint of the form \(P(Hz_{k+j|k} \leq g) \geq 1 - \zeta\) with \(g \in \mathbb{R}^p\), constraint separation tells us that this constraint is satisfied when:

\[
P(H^T_{i}(\bar{z}_{k+j|k} \leq g_i) \geq 1 - \zeta_i)
\]

(6.21)

where \(\zeta_i \geq 0 \forall i, \sum_{i=1}^{p} \zeta_i = \zeta\). This is satisfied by the candidate solution when:

\[
P(H^T_{i}(\bar{z}_{k+j|k} + \sum_{l=0}^{j-1} L_l \hat{F}_w k_{l+j} - \hat{F}e_{k+j} \leq g_i) \geq 1 - \zeta_i)
\]

(6.22)

Let

\[
\lambda_{i,k+j|k} = H^T_{i} \sum_{l=0}^{j-1} L_l \hat{F}_w k_{l+j} - \hat{F}e_{k+j}
\]

(6.23)

then for the optimization formulation we need \((6.22)\) in a form:

\[
H^T_{i} \bar{z}_{k+j|k} \leq g_i - \gamma_{i,k+j|k}
\]

(6.24a)

where, \(\gamma_{i,k+j|k}\) s.t.

\[
P(\lambda_{i,k+j|k} \leq \gamma_{i,k+j|k}) \geq 1 - \zeta_i
\]

(6.24b)

Assume \(\lambda_{i,k+j|k}\) has variance \(\sigma^2_{i,k+j|k}\), which can be computed as \(w\) and \(e\) are independent and have bounded variances. Now in order to compute such a \(\gamma_{i,k+j|k}\), we have the option of using one of multiple concentration inequalities:
Case A: $e$ has unbounded support: In this case, we can use the very commonly used Kouvaritakis et al. [2010], Boucheron et al. [2013] Chebyschev inequality:

$$P(\lambda_{i,k+j|k} \geq \gamma_{i,k+j|k}) \leq \zeta_i \tag{6.25}$$

This gives us

$$\gamma_{cheb}^{i,k+j|k} \geq \sigma_{i,k+j|k} \sqrt{1 - \zeta_i} \zeta_i \tag{6.26}$$

We can use this $\gamma_{cheb}^{i,k+j|k}$ in (6.24a) to be used for the constraints of the optimization. In this work, we do not develop the formulation for this further as strong guarantees on recursive feasibility cannot be achieved when the error distribution does not have a finite support.

Case B: $e$ has bounded support: In this case, we have the option of using either the Hoeffding or the Bernstein concentration inequalities Boucheron et al. [2013] (based on the form of the bound available). We know that $\lambda_{i,k+j|k}$ is formed by a sum of multiple independent random variables (6.23). Let this sum be $\lambda_{i,k+j|k} = \sum_v \lambda_v$, with $\lambda_v$ generally referring to elements of the sum in (6.23). Since $e$ and $w$ have bounded support, so do the $\lambda_v$’s, let their bounds be $a_v \leq \lambda \leq b_v \forall v$. Also, let their variances be $\sigma_v^2$. In this case, we can use the Hoeffding concentration inequality:

$$P(\lambda_{i,k+j|k} \geq \gamma_{i,k+j|k}) \leq \zeta_i \tag{6.27}$$

Solving this gives us a value of $\gamma_{i,k+j|k}$ to be used in the constraints for the optimization:

$$\gamma_{hoeff}^{i,k+j|k} \geq \sqrt{\sum_v (b_v - a_v)^2 \log(1/\zeta_i)} \tag{6.28}$$

Another option is to use the Bernstein concentration inequality. In order to use this, define $M = \max_v b_v$ (therefore $\lambda_v \leq M \forall v$). With this

$$P(\lambda_{i,k+j|k} \geq \gamma_{i,k+j|k}) \leq \zeta_i \tag{6.29}$$

$$= \exp \left( -\gamma_{i,k+j|k}^2 \frac{1}{M} \sum_v \sigma_v^2 \right)$$

101
Define $c_1 = -(2/3) \log(1/\zeta_i)$ and $c_2 = -2 \log(1/\zeta_i) \sum_v \sigma_v^2$. We can compute a $c_{i,k+j|k}$ that can be used in the optimization formulation from the above equation as follows:

$$\gamma_{i,k+j|k}^{\text{bernst}} \geq (1/2)(-c_1 \pm \sqrt{c_1^2 - 4c_2}) \quad (6.30)$$

Combining these with (6.24a) results in linear constraints on the optimization variables of the SAMPC such that the chance constraints $P(Hz_{k+j|k} \leq g) \geq 1 - \zeta$ are satisfied for all $j$ in the optimization horizon.

The rest of this section will focus on the recursive feasibility of the candidate solution of (6.20) (as constructed for the robust case) for the case where the estimation error (and process noise) distributions have bounded support.

### 6.11.2 Sketch of proof for recursive feasibility

#### Validity of the applied input and next state, initial condition, dynamics

Again, via construction (as shown in the robust case), these conditions are met by the candidate solution.

#### State Constraints

Similar to the case when $e$ came from a normal distribution, the condition for recursive feasibility takes on the form:

$$H_i^T L_j \hat{F} \hat{w}_{k+1} \leq \gamma_{i,(k+1)+j|k+1} - \gamma_{i,k+j+1|k} = p_{ij} \quad (6.31)$$

This $p_{ij}$ can be computed offline since $\gamma_{i,(k+1)+j|k+1}$, $\gamma_{i,k+j+1|k}$ are computed apriori (in both cases where the mode remains the same from time $k$ to $k+1$, or changes). With this, and given the samples that form the distribution of $e$ (through the profiling step), the probability can be computed via brute force by summing over all combinations of the elements that make up the sum $H_i^T L_j \hat{F} \hat{w}_{k+1}$.

Recall that we assume bounded support of the distributions of $e$ and $w$. In this case, we also know that $\hat{w}_{k+1} \in \hat{W}$. For such cases, we can show prove recursive feasibility probability to 1 by using the approach presented in Kouvaritakis et al. [2010]. A sketch of this proof follows.

For simplicity of notation, denote $\gamma_{i,l+j|l} = \gamma_{i,j}$, and similarly for other variables with the same indexing that follow.

First, using the bounded support of the uncertainties, we can compute:

$$\kappa_{i,j} = \max_{\hat{w}_{k+1} \in \hat{W}} H_i^T L_j \hat{F} \hat{w}_{k+1} \quad (6.32)$$
Now let $\tilde{\gamma}_{i,j}$ be the maximum element of the $j^{th}$ column of the following matrix:

$$
\begin{bmatrix}
\tilde{\gamma}_i,1 & \gamma_i,2 & \gamma_i,3 & \cdots \\
0 & \gamma_i,1 + \kappa_i,1 & \gamma_i,2 + \kappa_i,2 & \cdots \\
\vdots & 0 & \gamma_i,1 + \kappa_i,1 + \kappa_i,2 & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{bmatrix}
$$

(6.33)

Replacing $\gamma$ in (6.24a) by this new $\tilde{\gamma}$ gives us the constraints:

$$
H_i^T \tilde{z}_{k+j|k} \leq g_i - \tilde{\gamma}_{i,k+j|k}
$$

(6.34)

This added conservativeness turns the recursive feasibility probability to 1. This can be observed by rewriting (6.31):

$$
P(H_i^T \hat{L}_j \hat{F} \hat{w}_{k+1} \leq \tilde{\gamma}_{i,j} - \tilde{\gamma}_{i,j+1})
$$

Which, by definition of $\tilde{\gamma}_{i,j}$

$$
P(H_i^T \hat{L}_j \hat{F} \hat{w}_{k+1} \leq \kappa_{i,j})
$$

further, by definition of $\kappa_{i,j}$

$$
= 1
$$

(6.35)

Terminal Constraint

The terminal constraint is recursively feasible by the definition of the invariant set (same formulation as for the robust case) and using the fact that $\hat{w}_{k+1} \in \hat{W}$, where $W$ can be computed because of the bounded support of the disturbances. The proof follows from the deterministic (robust) case.

This concludes the proof sketch to show that the SAMPC of (6.7) formulated using the set shrinking of Sec. 6.11.1 both satisfies the chance constraints and is recursively feasible with probability 1 (i.e. proves Theorem 6.11.1).

6.12 More details on the experimental setup

To evaluate our methodology on a real platform, we applied it to a hexrotor with the Odroid-U3 as a computation platform, running the Robot Operating System (ROS) [Quigley et al. 2009] in Ubuntu. For the evaluations, the hexrotor is tasked with following the two trajectories shown in Fig. 6.11. As can be seen in Fig. 6.17, the visual odometry algorithm can occasionally take a long time to give a pose estimate.

In our formulation we have assumed that the estimator satisfies the $(\delta, \epsilon)$ contract requested by the controller. Thus, to ensure that the estimator fulfils the contract and that the mathematical guarantees provided by our RAMPC formulation hold, instead of using the visual odometry algorithm to fly the robot, we injected delays
and errors into the measurements from Vicon, which is a high accuracy localization system. These delays and errors were selected from the $\Delta$ curve obtained by profiling the SVO algorithm (Fig 6.9). The hexrotor flies using these pose estimates and our control algorithms for both the position/velocity control and setting the time deadline for the next estimate. The RAMPC has the positions and velocities in the 3-axes as its references, $x_k^{ref}$, to track, and generates control inputs in the form of desired thrust, roll and pitch for a low-level attitude controller to track. The RAMPC and SAMPC are coded in CVXGEN [Mattingley and Boyd 2012] and the generated C Code is integrated in the ROS module for control of the hexrotor, running at 20Hz. The constraint sets for the RAMPC and SAMPC are computed offline in MATLAB and then used in CVXGEN as polyhedron type constraints. The constraint set $X$ defines a safe set of positions and velocities in the flying area. The constraint set $U$ of inputs keeps desired pitch and roll magnitudes less than 30 degrees and desired thrust within limits of the hex-rotor abilities.
Chapter 7

Robust Model Predictive Control for Constrained Non-Linear Systems

7.1 Introduction

The Robust MPC solution in the co-design framework presented in Chapter 6 is limited to linear time-invariant systems. This chapter develops the theoretical foundation to extend the co-design framework to non-linear systems, particularly input-affine non-linear systems. Here, we are concerned with the problem of controlling nonlinear dynamical systems \( S \) of the form \( \dot{x} = f(x) + G(x)u \) under state and input constraints, and subject to errors in the state estimate. This problem is formulated as

\[
\min_{x,u} l(x, u)
\]

s.t. \( \dot{x} = f(x) + G(x)u \)
\( x \in X, u \in U \)

where \( l(x, u) \) is a cost function whose minimization over the state and input trajectories \( x \) and \( u \) ensures stability of the system. Sets \( X \subset \mathbb{R}^{n_x} \) and \( U \subset \mathbb{R}^{n_u} \) encode constraints on the state (e.g., safety) and the input. The input \( u = u(\hat{x}) \) is a function of a state estimate that in general differs from the true state of the system.

Whereas Model Predictive Control (MPC) is a widely used technique for controlling linear systems with constraints, its application to nonlinear systems involves the repeated solution of generally non-quadratic, non-convex optimizations. Various approaches for solving (or approximately solving) the optimizations and their trade-offs are reviewed in [Cannon 2004]. Another approach is to first feedback linearize the system \( S \) [Khalil 2002]: namely, the applied control \( u = u(x, v) \) is designed in such a way that the resulting closed-loop dynamics \( S_{fl} \) are now linear:

\[
S_{fl} : \dot{z} = Az + Bv
\]
The input $v$ to the linearized dynamics can now be computed so as to optimize system performance and ensure stability. The state $z$ of the linearized system $S_{fl}$ is related to the state $x$ of the nonlinear system $S$ via a (system-specific) function $T$: $z = T(x)$.

Contributions: In this chapter we develop a feedback linearization solution to the above control problem, with state estimation errors, input and state constraints, and non-identity $T$. To the best of our knowledge, this is the first feedback linearization solution to this problem. The resulting control problem is solved by RMPC with time-varying linear constraint sets.

The chapter is organized as follows: in the next section we formulate the feedback linearized control problem. In Sec. 7.3, we describe the RMPC algorithm we use to solve it, and prove that it stabilizes the nonlinear system. Sec. 7.4 shows how to compute the various constraint sets involved in the RMPC formulation, and Sec. 7.5 applies our approach to two examples. Sec. 7.6 concludes this chapter.

7.2 Problem Formulation

A common method for control of nonlinear systems is Feedback linearization [Khalil 2002]. Briefly, in feedback linearization, one applies the feedback law $u(x, v) = R(x)^{-1}(-b(x) + v)$ to (7.1), so that the resulting dynamics, expressed in terms of the transformed state $z = T(x)$, are linear time-invariant:

$$S_{fl}: \dot{z} = A_c z + B_c v$$  (7.2)

By using the remaining control authority in $v$ to control $S_{fl}$, we can effectively control the non-linear system for, say, stability or reference tracking. $T$ is a diffeomorphism [Khalil 2002] over a domain $D \subset X$. The original and transformed states, $x$ and $z$, have the same dimension, as do $u$ and $v$, i.e. $n_x = n_z$ and $n_u = n_v$. Because we are controlling the feedback linearized system, we must find constraint sets $Z$ and $V$ for the state $z$ and input $v$, respectively, such that $(z, v) \in Z \times V \implies (T^{-1}(z), u(T^{-1}(z), v)) \in X \times U$. This is done in Sec. 7.4.4. We assume that the system (7.1) has no zero dynamics [Khalil 2002] and all states are controllable. In case there are zero dynamics, then our approach is applicable to the controllable subset of the states as long as the span of the rows of $G(x)$ is involutive [Khalil 2002].

For feedback linearizing (7.1) and for controlling (7.1), only a periodic state estimate $\hat{x}$ of $x$ is available. This estimate is available periodically every $\tau$ time units, so we may write $\hat{x}_k := \hat{x}(k\tau) = x_k + e_k$, where $x_k$ and $e_k$ are sampled state and error respectively. We assume that $e_k$ is in a bounded set $E$ for all $k$. This implies that the feedback linearized system can be represented in discrete-time: $z_{k+1} = A z_k + B z_k$.

The corresponding z-space estimate $\hat{z}_k$ is given by $\hat{z}_k = T(\hat{x}_k)$. In general the z-space error $\hat{e}_k := T(\hat{x}_k) - T(x_k)$ is bounded for every $k$ but does not necessarily lie in $E$. Let $\hat{E}_k$ be the set containing $\hat{e}_k$; in Sec. 7.4.3 we show how to compute it.

Because the linearizing control operates on the state estimate and not $x_k$, we add a process noise term to the linearized, discrete-time dynamics. Our system model is
therefore

\[ z_{k+1} = Az_k + Bv_k + w_k \]  

where the noise term \( w_k \) lies in the bounded set \( W \) for all \( k \). An estimate of \( W \) can be obtained using the techniques presented here. The control problem (7.1) is therefore replaced by:

\[
\min_{z,v} \quad \sum_{k=0}^{\infty} z_k^T Q z_k + v_k^T R v_k \\
\text{s.t.} \quad z_{k+1} = Az_k + Bv_k + w_k, \quad z_k \in Z, v_k \in V, w_k \in W
\]  

(In Thm. 7.2 we show that minimizing this cost function implies stability of the system).

It is easy to derive the dynamics of the state estimate \( \hat{z}_k \):

\[
\dot{\hat{z}}_{k+1} = \hat{z}_{k+1} + \hat{e}_{k+1} \\
= Az_k + Bv_k + w_k + \hat{e}_{k+1} \\
= A\hat{z}_k + Bv_k + (w_k - \hat{e}_{k+1} - A\hat{e}_k) \\
= A\hat{z}_k + Bv_k + \hat{w}_{k+1}
\]  

where \( \hat{w}_{k+1} = w_k - \hat{e}_{k+1} - A\hat{e}_k \), and lies in the set \( \hat{W}_{k+1} := W \oplus \hat{E}_{k+1} \oplus (-A\hat{E}_k) \).

**Example 8.** Consider the 2D system

\[
\dot{x}_1 = \sin(x_2), \quad \dot{x}_2 = -x_1^2 + u
\]  

The safe set for \( x \) is given as \( X = \{ |x_1| \leq \pi/2, |x_2| \leq \pi/3 \} \), and the input set is \( U = [-2.75, 2.75] \). For the measurement \( y = h(x) = x_1 \), the system can be feedback linearized on the domain \( D = \{ x | \cos(x_2) \neq 0 \} \), where it has a relative degree of \( \rho = 2 \). The corresponding linearizing feedback input is \( u = -\tan(x_2) + (\cos(x_2))v \). The feedback linearized system is \( \dot{z}_1 = z_2, \dot{z}_2 = v \), where \( T \) is given by \( z = T(x_1, x_2) = (x_1, \sin(x_2)) \). We can analytically compute the safe set in z-space as \( Z = T(X) = \{ |z_1| \leq \pi/2, |z_2| \leq 0.8660 \} \).

\( \Delta \)

For a more complicated \( T \), it is not possible to obtain analytical expressions for \( Z \). The computation of \( Z \) in this more general case is addressed in Sec. 7.4.4

**Notation.** Given two subsets \( A, B \) of \( \mathbb{R}^n \), their *Minkowski sum* is \( A \oplus B := \{ a + b | a \in A, b \in B \} \). Their *Pontryagin difference* is \( A \ominus B = \{ c \in \mathbb{R}^n | c+b \in A \forall b \in B \} \). Given integers \( n \leq m \), \( [n : m] := \{ n, n+1, \ldots, m \} \).

**Assumption.** The approach we use applies when \( X, U, E \) and \( W \) are arbitrary convex polytopes (i.e. bounded intersections of half-spaces). For the sake of simplicity, we assume they are all hyper-rectangles that contain the origin, i.e. sets of the form \( [a_1, \bar{a}_1] \times \ldots \times [a_n, \bar{a}_n], \ a_i \leq 0 \leq \bar{a}_i \ \forall i \).
7.3 Robust MPC for the feedback linearized system

Following [Richards and How 2005a; Pant et al. 2015a], we formulate a Robust MPC (RMPC) controller of (7.4) via constraint restriction. We outline the idea before providing the technical details. The key idea is to move the effects of estimation error $\tilde{e}_k$ and process noise $w_k$ (the ‘disturbances’) to the constraints, and work with the nominal (i.e., disturbance-free) dynamics: $\tilde{z}_{k+1} = A\tilde{z}_k + Bv_k$, $\tilde{z}_0 = \hat{z}_0$. Because we would be optimizing over disturbance-free states, we must account for the noise in the constraints. Specifically, rather than require the next (nominal) state $\tilde{z}_{k+1}$ to be in $Z$, we require it to be in the shrunk set $Z \ominus \hat{W}_{k+1|k} \ominus \tilde{E}_{k+1|k}$: by definition of Pontryagin difference, this implies that whatever the actual value of the noise $\hat{w}_{k+1} \in \hat{W}_{k+1|k}$ and of the estimation error $\tilde{e}_{k+1} \in \tilde{E}_{k+1|k}$, the actual state $z_{k+1}$ will be in $Z$. This is repeated over the entire MPC prediction horizon $j = 1, \ldots, N$, with further shrinking at every step. For further steps ($j > 1$), the process noise $\hat{w}_{k+j|k}$ is propagated through the dynamics, so the shrinking term $\hat{W}$ is shaped by a stabilizing feedback controller $\tilde{z} \mapsto K\tilde{z}$. At the final step ($j = N + 1$), a terminal constraint is derived using the worst case estimation error set $\hat{E}_{max}$ and a global inner approximation for the input constraints, $V_{inner-global}$.

Through this successive constraint tightening we ensure robust safety and feasibility of the feedback linearized system (and hence of the non-linear system). Since we use just the nominal dynamics, and show that the tightened constraints are linear in the state and inputs, we still solve a Quadratic Program (QP) for the RMPC optimization. The difficulty of applying RMPC in our setting is that the amounts by which the various sets are shrunk vary with time because of the time-varying state estimation error, are state-dependent, and involve set computations with the non-convexity preserving mapping $T$. One of the contributions here is to establish recursive feasibility of RMPC with time-varying constraint sets.

The RMPC optimization $P_k(\hat{z}_k)$ for solving (7.4) is:

$$J^*(\hat{z}_k) = \min_{\bar{z},u} \sum_{j=0}^{N} \{ \bar{z}_{k+j|k}^T Q \bar{z}_{k+j|k} + v_{k+j|k}^T R v_{k+j|k} \}$$

(7.7a)

$$\bar{z}_{k|k} = \hat{z}_k$$

(7.7b)

$$\bar{z}_{k+j|k} = A\bar{z}_{k+j|k} + Bv_{k+j|k}, j = 0, \ldots, N$$

(7.7c)

$$\bar{z}_{k+j|k} \in \bar{Z}_{k+j|k}, j = 0, \ldots, N$$

(7.7d)

$$v_{k+j|k} \in V_{k+j|k}, j = 0, \ldots, N - 1$$

(7.7e)

$$P_{N+1} = [\bar{z}_{k+N+1|k}, v_{k+N|k}]^T \in P_f$$

(7.7f)

Here, $\bar{z}$ is the state of the nominal feedback linearized system. The cost and constraints of the optimization are explained below:
Eq. (7.7a) shows a cost quadratic in $\bar{z}$ and $v$, where as usual $Q$ is positive definite and $R$ is positive semi-definite. In the terminal cost term, $Q_f$ is the solution of the Lyapunov equation $Q_f - (A + BK)^T Q_f (A + BK) = Q + K^T R K$. This choice guarantees that the terminal cost equals the infinite horizon cost under a linear feedback control $\bar{z} \mapsto K \bar{z}$ [Kouvaritakis and Cannon 2015].

Eq. (7.7b) initializes the nominal state with the current state estimate.

Eq. (7.7c) gives the nominal dynamics of the discretized linearized system.

Eq. (7.7d) tightens the admissible set of the nominal state by a sequence of shrinking sets.

Eq. (7.7e) constrains $v_{k+j|k}$ such that the corresponding $u(x, v)$ is admissible, and the RMPC is recursively feasible.

Eq. (7.7f) constrains the final input and nominal state to be within a terminal set $P_f$.

The details of these sets’ definitions and computations are given in Sec. 7.4.

### 7.3.1 State and Input Constraints for the Robust MPC

The state and input constraints for the RMPC are defined as follows:

The state constraints $Z_{k+j|k}$: The tightened state constraints are functions of the error sets $\tilde{E}_{k+j|k}$ and disturbance sets $\hat{W}_{k+j|k}$, and defined $\forall j = 0, \ldots, N$

$$Z_{k+j|k} = Z \ominus \bigoplus_{i=0}^{j-1} (L_i \hat{W}_{k+(j-i)|k}) \ominus (-\tilde{E}_{k+j|k})$$

(Recall $Z$ is a subset of $T(X)$, $\hat{W}_{k+j|k}$ and $\tilde{E}_{k+j|k}$ bound the estimation error and noise, resp., and are formally defined in Sec. 7.4). The state transition matrix $L_j$, $\forall j = 0, \ldots, N$ is defined as $L_0 = I$, $L_{j+1} = (A + BK) L_j$. The intuition behind this construction was given at the start of this section.

The input constraints $V_{k+j|k}$: $\forall j = 0, \ldots, N - 1$

$$V_{k+j|k} = V_{k+j|k} \ominus \bigoplus_{i=0}^{j-1} K L_i \hat{W}_{k+(j-i)|k}$$

where $V_{k+j|k}$ is an inner-approximation of the set of admissible inputs $v$ at prediction step $j + k|k$, as defined in Sec. 7.4.2. The intuition behind this construction is similar to that of $Z_{k+j|k}$: given the inner approximation $V_{k|k}$, it is further shrunk at each prediction step $j$ by propagating forward the noise $\hat{w}_k$ through the dynamics, and shaped according to the stabilizing feedback law $K$, following Richards and How 2005a.

The terminal constraint $P_f$: This constrains the extended state $p_k = [\bar{z}_k, v_{k-1}]^T$, and is given by

$$P_f = C_p \ominus \left[ (A + BK)^N K (A + BK)^{N-1} \right] \hat{W}_{max}$$
where \( \hat{W}_{\text{max}} \subset \mathbb{R}^{n_z} \) is a bounding set on the worst-case disturbance (we show how it’s computed in Sec. 7.4.3), and \( C_p \subset \mathbb{R}^{n_z} \times \mathbb{R}^{n_v} \) is an invariant set of the nominal dynamics subject to the stabilizing controller \( \tilde{z} \mapsto K\tilde{z} \), naturally extended to the extended state \( p \): that is, there exists a feedback control law \( p \mapsto \hat{K}p \), such that \( \forall p \in C_p \)

\[
\hat{A}p + \hat{B}\hat{K}p + \hat{L}_N[\hat{w}^T, 0^T]^T \in C_p, \forall \hat{w} \in \hat{W}_{\text{max}}
\]  

(7.11)

with \( \hat{A} = \begin{bmatrix} A & 0_{n \times m} \\ 0_{m \times n} & 0_{m \times m} \end{bmatrix} \), \( \hat{B} = \begin{bmatrix} B \\ I_{m \times m} \end{bmatrix} \), \( \hat{K} = [K \ 0_{m \times m}] \), \( \hat{L}_N = (\hat{A} + \hat{B}\hat{K})^N \). It is important to note the following:

- The set \( P_f \) can be computed offline since it depends on \( \hat{W}_{\text{max}}, \tilde{E}_{\text{max}} \) and the global inner approximation for the constraints on \( v, V_{\text{inner-global}} \), all of which can be computed offline.

- If \( P_f \) is non-empty, then all intermediate sets that appear in (7.7) are also non-empty, since \( P_f \) shrinks the state and input sets by the maximum disturbances \( \hat{W}_{\text{max}} \) and \( \tilde{E}_{\text{max}} \). Thus we can tell, before running the system, whether RMPC might be faced with empty constraint sets (and thus infeasible optimizations).

- Note that all constraints are linear.

### 7.3.2 The Control Algorithm

We can now describe the algorithm used for solving (7.7) by robust receding horizon control.

### 7.3.3 Robust Feasibility and Stability

We are now ready to state the main result of this chapter: namely, that the RMPC of the feedback linearized system (7.7) is feasible at all time steps if it starts out feasible, and that it stabilizes the nonlinear system, for all possible values of the state estimation error and feedback linearization error.

**Theorem 7.1 (Robust Feasibility).** If at some time step \( k_0 \geq 0 \), the RMPC optimization \( P_{k_0}(\hat{z}_{k_0}) \) is feasible, then all subsequent optimizations \( P_k(\hat{z}_k)k > k_0 \) are also feasible. Moreover, the nonlinear system (7.1) controlled by algorithm 3 and subject to the disturbances \( (E, W) \) satisfies its state and input constraints at all times \( k \geq k_0 \).

The proof is in the online report Pant et al. [2016].

**Theorem 7.2 (Stability).** Given an equilibrium point \( x_e \in X_0 \subset T^{-1}(Z) \) of the nonlinear dynamics (7.1), Algorithm 3 stabilizes the nonlinear system to an invariant set around \( x_e \).
Algorithm 2 RMPC via feedback linearization

**Require:** System model, $X$, $U$, $E$, $W$

- Offline, compute:
  - Initial safe sets $X_0$ and $Z$
  - $E_{\max}$, $W_{\max}$
  - $C_p$, $P_f$  \(\text{(Sec. 7.4.3)}\)

- Online:
  - if $P_f = \emptyset$ then
    - Quit
  - else for $k = 1, 2, \ldots$ do
    - Get estimate $\hat{x}_k$, compute $\hat{z}_k = T(\hat{x}_k)$
    - Compute $V_{k+j|k}, \hat{E}_{k+j|k}, \hat{W}_{k+j|k}$  \(\text{(Sec. 7.4.2)}\)
    - Compute $Z_{k+j|k}, V_{k+j|k}$  \(\text{(Sec. 7.3.1)}\)
    - $(v_{k|k}, \ldots, v_{k+N|k}) = \text{Solution of } P_k(\hat{z}_k)$
    - $v_k = v_{k|k}^*$
    - Apply $u_k = R(\hat{x}_k)^{-1}[b(\hat{x}_k) + v_k]$ to plant

**Proof.** Let $T$ be the diffeomorphism mapping $x$ to $z$ from feedback linearization. By a change of variables $z' = z - T(x_e)$, stabilizing the linear dynamics (with state $z'$) to 0 implies stabilizing the nonlinear dynamics to $x_e$. Recall that $Q$ and $Q_f$ of (7.7) are positive definite and that $R$ is positive semi-definite, so that the optimal cost $J^*(\hat{z}_k)$ is a positive definite function of $\hat{z}_k$, and that the terminal weight in (7.7) is equivalent to the infinite horizon cost (by our choice of $Q_f$). Finally Thm. 7.1 guarantees that the tail of the input sequence computed at $k$ is admissible at time $k + 1$. Therefore it is a standard result that the optimal cost $J^*(\hat{z}_k)$ is non-increasing in $k$ and that 0 is a stable equilibrium for the closed-loop linear system (e.g., see Kouvaritakis and Cannon 2015). Moreover the nominal feedback-linearized system $(\hat{z})$ converges to 0 from anywhere in $Z$. Therefore, the nominal $\hat{x}_k$ converges to $x_e$ from anywhere in $X_0 \subset T^{-1}(Z)$. The true state $(x_k)$ then converges to the invariant set around $x_e$.

### 7.4 Set definitions for the RMPC

Algorithm 2 and the problem $P_k(\hat{z}_k)$ (7.7) use a number of constraint sets to ensure recursive feasibility of the successive RMPC optimizations, namely: inner approximations of the admissible input sets $V_{k+j|k}$, bounding sets for the $(T$-mapped) estimation error $\hat{E}_{k+j|k}$, bounding sets for the process noise $\hat{W}_{k+j|k}$, and the largest error and noise sets $E_{\max}$ and $W_{\max}$. In this section we show how these sets are defined and computed.
7.4.1 Approximating the reach set of the nonlinear system

First we show how to compute an outer-approximation of the \( j \)-step reach set of the nonlinear system, starting at time \( k \), \( X_{k+j|k} \). This is needed for computing \( V_{k+j|k} \) and \( \tilde{E}_{k+j|k} \).

In all but the simplest systems, forward reachable sets cannot be computed exactly. To approximate them we may use a reachability tool for nonlinear systems like RTreach \textsuperscript{(Johnson et al.) \[2016\]. A reachability tool computes an outer-approximation exactly. To approximate them we may use a reachability tool for nonlinear systems like RTreach \textsuperscript{(Johnson et al.) \[2016\]. A reachability tool computes an outer-approximation of the reachable set of a system starting from some set \( \mathcal{X} \subset X \), subject to inputs from a set \( U \), for a duration \( T \geq 0 \). Denote this approximation by \( RT=\mathcal{T}(\mathcal{X},U) \), so \( x(T) \in RT=\mathcal{T}(\mathcal{X},U) \) for all \( T \), \( x(0) \in \mathcal{X} \) and \( u : [0,T] \to U \).

At time \( k \), the state estimate \( \hat{x}_k \) is known. Therefore \( x_k = \hat{x}_k - e_k \in \hat{x}_k \oplus (-E) := X_{k|k} \). Propagating \( X_{k|k} \) forward one step through the continuous-time nonlinear dynamics yields \( X_{k+1|k} \), which is outer-approximated by \( RT=\mathcal{T}(X_{k|k},U) \). The state estimate that the system will receive at time \( k+1 \) is therefore bound to be in the set \( RT=\mathcal{T}(X_{k|k},U) \oplus E \). Since \( 0 \in E \), we maintain \( X_{k+1|k} \subset RT=\mathcal{T}(X_{k|k},U) \oplus E \). We define the outer-approximate reach set at \( k+1 \), computed at time \( k \), to be

\[
X_{k+1|k} := RT=\mathcal{T}(X_{k|k},U) \oplus E \oplus (-E)
\]

(The reason for adding the extra \(-E\) term will be apparent in the proof to Thm. 7.1).

More generally, for \( 1 \leq j \leq N \), we define the \( j \)-step approximation computed at time \( k \) to be

\[
\begin{align*}
X_{k|k} &:= \hat{x}_k \oplus (-E) \\
X_{k+j|k} &:= RT=\mathcal{T}(X_{k+j-1|k},U) \oplus E \oplus (-E)
\end{align*}
\]

Fig. 7.1 shows a visualization of this approach. The following holds by construction:

**Lemma 4.** For any time \( k \) and step \( j \geq 1 \), \( X_{k+j|k} \subset X_{k+j|k} \).

This construction of \( X_{k+j|k} \) permits us to prove recursive feasibility of the RMPC controller, because it causes the constraints of the RMPC problem setup at time \( k+1 \) to be consistent with the constraints of the RMPC problem setup at time \( k \).

7.4.2 Approximating the bounding sets for the input

Given \( x \in X \), define the set \( V(x) := \{v \in \mathbb{R}^{n_v} \mid u(x) = R^{-1}(x)[b(x) + v] \in U\} \). We assume that there exist functions \( g_i, v_i : X \to \mathbb{R} \) s.t. for any \( x \), \( V(x) = \{[v_1, \ldots, v_{n_v}]^T \mid v_i(x;U) \leq v_i \leq v_i(x;U)\} \). Because in general \( V(x) \) is not a rectangle, we work with inner and outer rectangular approximations of \( V(x) \). Specifically, let \( \mathcal{X} \) be a subset of \( X \). Define the inner and outer bounding rectangles, respectively

\[
\begin{align*}
\overline{V}(\mathcal{X}) := \{[v_1, \ldots, v_{n_v}]^T \mid \max_{x \in \mathcal{X}} v_i(x;U) \leq v_i \leq \min_{x \in \mathcal{X}} v_i(x;U)\}
\end{align*}
\]
\[ X_{k|k} = (\hat{x}_k) \oplus (-E) \]
\[ X_{k+1|k} = RT_z(X_{k|k}, U) \oplus (-E) + E \]
\[ X_{k+2|k} = RT_z(X_{k+1|k}, U) \oplus (-E) \oplus E \]

Figure 7.1: The outer-approximated reach sets for \( x_{k+j} \), computed at time steps \( k, k+1 \), used to compute \( \tilde{E}_{k+j|k}, V_{k+j|k} \).

\[ V(\mathcal{X}) := \{ [v_1, \ldots, v_n]^T | \min_{x \in \mathcal{X}} u_i(x; U) \leq v_i \leq \max_{x \in \mathcal{X}} v_i(x; U) \} \]

By construction, we have for any subset \( \mathcal{X} \subset X \)
\[ \overline{V}(\mathcal{X}) \subseteq \cap_{x \in \mathcal{X}} V(x) \subset V(\mathcal{X}) \quad (7.13) \]

If two subsets of \( X \) satisfy \( \mathcal{X}_1 \subset \mathcal{X}_2 \), then it holds that
\[ \overline{V}(\mathcal{X}_2) \subset \overline{V}(\mathcal{X}_1), \ V(\mathcal{X}_1) \subset V(\mathcal{X}_2) \quad (7.14) \]

We can compute:
\[ V_{k+j|k} = V(X_{k+j|k}), \ V_{\text{inner-global}} = V(X) \quad (7.15) \]

In practice we use interval arithmetic to compute these sets since \( X_{k+j|k} \) and \( U \) are hyper-intervals. Fig. 7.2 shows these sets for the running example.

### 7.4.3 Approximating the bounding sets for the disturbances

We will also need to define containing sets for the state estimation error in \( z \) space: recall that \( \hat{z}_k = T(\hat{x}_k) = T(x_k + e_k) \). We use a Taylor expansion
\[
\hat{z}_k = T(x_k) + \frac{dT}{dx}(x_k) e_k + \frac{1}{2} e_k^T \frac{d^2T}{dx^2}(c) e_k, c \in x_k + E
\]
\[
= T(x_k) + M(x_k) e_k + r_k(c), c \in x_k + E
\]
\[
= T(x_k) + h_k + r_k(c), c \in x_k + E
\]

The remainder term \( r_k(c) \) is bounded in the set \( \cup_{c \in (x_k) + E} \frac{1}{2} e_k^T \frac{d^2T}{dx^2}(c) e \). Thus when setting up \( \mathbb{P}_k(\hat{z}_k) \), at the \( j \)th step, \( r_{k+j|k} \in D_{k+j|k} := \cup_{c \in X_{k+j|k} + E} \frac{1}{2} e_k^T \frac{d^2T}{dx^2}(c) e \), where \( X_{k+j|k} \) is the reach set computed in \( 7.12 \).
Figure 7.2: Local and global inner approximations of input constraints for running example, with $X_{k+j.k} = [-\pi/4, 0] \times [-0.9666, -0.6283]$ for some $k, j$ and $U = [-2.75, 2.75]$. Color in online version.
The error $h_k$ lives in $\cup_{x \in X_k,e \in E} M(x) e$. Thus when setting up $P_k(\hat{z}_k)$, the error $h_{k+jk}$ lives in $\cup_{x \in X_{k+jk}} M(x) E$. Finally the rectangular over-approximation of this set is

$$H_{k+j|k} = \{ h | \sum_{\ell=1}^{n_x} \min_{x \in X_{k+j|k}, e \in E} M_{i\ell}(x) e(\ell) \leq h(i) \leq \sum_{\ell=1}^{n_x} \max_{x \in X_{k+j|k}, e \in E} M_{i\ell}(x) e(\ell) \}$$  \hspace{1cm} (7.16)

where $M_{i\ell}$ is the $(i, \ell)$th element of matrix $M$ and $h(\ell)$ is the $\ell$th element of $h$.

Therefore the state estimation error $h_{k+j|k} + r_{k+j|k}$ is bounded in the set $H_{k+j|k} \oplus D_{k+j|k}$. In the experiments we ignore the remainder term $D_{k+j|k}$ based on the observation that $e_k$ is small relative to the state $x_k$. Thus we use:

$$\tilde{E}_{k+j|k} = H_{j+k|j}$$  \hspace{1cm} (7.17)

**Example 9.** For the running example (7.6), we have $M = [1, 0; 0, \cos(x_2)]$. If the estimation error $e$ (in radians) is bounded in $E = \{e || e||_\infty \leq 0.0227 \}$, then the relative linearization error, averaged over several realizations of the error, is less than $2 \times 10^{-3}$. \hspace{1cm} \$\Box$

We also need to calculate containing sets for the process noise $\hat{w}$. Recall that for all $k, j$, $\hat{z}_{k+j+1} = A \hat{z}_{k+j} + B v_k + \hat{w}_{k+j+1}$. Therefore

$$\hat{w}_{k+j+1} \in \hat{W}_{k+j+1|k} := W \oplus \tilde{E}_{k+j+1|k} \oplus (-A \tilde{E}_{k+j|k})$$  \hspace{1cm} (7.18)

We also define the set $\tilde{E}_{\text{max}}$, which is necessary for the terminal constraints of Eq. (7.10). $\tilde{E}_{\text{max}}$ represents the worst case bound on the estimation error $\tilde{e}_k$, and is computed similar to Eq. (7.17), but over the entire set $X$.

$$\sum_{\ell=1}^{n_x} \min_{x \in X, e \in E} M_{i\ell}(x) e(\ell) \leq \tilde{e}_k(i) \leq \sum_{\ell=1}^{n_x} \max_{x \in X, e \in E} M_{i\ell}(x) e(\ell)$$  \hspace{1cm} (7.19)

$\hat{W}_{\text{max}}$ is then defined as:

$$\hat{W}_{\text{max}} = W \oplus \tilde{E}_{\text{max}} \oplus (-A \tilde{E}_{\text{max}})$$  \hspace{1cm} (7.20)

For the running example, Fig. 7.3 shows the set $\tilde{E}_{\text{max}}$ and $\tilde{E}_{k+j|k}$ computed by Eqs. (7.17) and (7.19), for an arbitrary $X_{k+j|k} = [-\pi/4, 0] \times [-0.9666, -0.6283]$. It also shows 1000 randomly generated values for $T(\hat{x}) - x$ (for randomly generated $e \in E$ and $x \in X_{k+j|k}$), and all fall inside $\tilde{E}_{k+j|k}$.
Figure 7.3: The error sets $\tilde{E}_{max}$ and $\tilde{E}$ computed over an arbitrary $X_{k+j|k}$. Also shown are realizations of $\tilde{e} := T(\hat{x}) - T(x)$ for randomly chosen $x \in \mathcal{X}$. Color in online version.
7.4.4 Transforming between x-space and z-space

Since we control the system in z-space, we need to compute a set $Z \subset \mathbb{R}^{n_z}$ s.t. $z \in Z \implies x = T^{-1}(z) \in X$, i.e. $Z \subset T(X)$. Thus keeping the state $z$ of the linearized dynamics in $Z$ implies the nonlinear system’s state $x$ remains in $X$. Moreover, to check feasibility at time 0 of the MPC optimization, and for stability of the nonlinear dynamics, we need a subset $X_0 \subset X$ s.t. $x \in X_0 \implies z = T(x) \in Z$, i.e. $X_0 \subset T^{-1}(Z)$. Because $T$ can be an arbitrary diffeomorphism $Z$ and $X_0$ have to be computed numerically.

1. Let $Z_1 \subset \mathbb{R}^{n_z}$ be the rectangle with bounds in the $i^{th}$ dimension $[\min_{x \in X} T_i(x), \max_{x \in X} T_i(x)]$, $i = 1, \ldots, n_x$. This over-approximates $T(X)$. Next we need to prune it so it under-approximates $T(X)$.

2. Define $z_{in} := \min\{\|z\|_0 \mid z \in Z_1, T^{-1}(z) \notin X\}$. $z_{in}$ is the smallest-norm inadmissible $z$ in $Z_1$. Thus all points in the $\ell_0$-ball of radius $\|z_{in}\|, B_z(0, \|z_{in}\|)$, are admissible, i.e. their pre-images via $T^{-1}$ are in $X$.

3. Let $R_z$ be the largest inscribed rectangle in $B_z(0, \|z_{in}\|)$. Now we need to get the $x$-set that maps to $R_z$ (or a subset of it).

4. Let $X_1 \subset X$ be the rectangle with bounds in the $i^{th}$ dimension $[\min_{x \in R_z} T^{-1}_i(z), \max_{x \in R_z} T^{-1}_i(z)]$. Again, this is an over-approximation of $T^{-1}(R_z)$, so it needs to be pruned.

5. Define $x_{in} = \inf\{\|x\|_0 \mid x \in X_1, T(x) \notin R_z\}$. Then every point in the $\ell_0$-ball $B_x(0, \|x_{in}\|) \subset X$ maps via $T$ to $R_z$.

Therefore we choose $Z = R_z$ and $X_0$ to be the largest inscribed rectangle in $B_x(0, \|x_{in}\|)$.

7.5 Experiments

We evaluate our approach on the running example, and on a 4D flexible joint manipulator. We implemented the RMPC controller of Alg. 2 in MATLAB. The set computations were done using the MPT Toolbox [Herceg et al., 2013], and the invariant set computations using the Matlab Invariant Set Toolbox [Kerrigan, 2016]. The reachability computations for $X_{k+j|k}$ were performed on the linear dynamics and mapped back to x-space as described in point 4) of Sec. 7.4.4. The RMPC optimizations were formulated in CVX [Grant and Boyd, 2013] and solved by Gurobi Optimization [2015].

7.5.1 Running example

For the running example of Eq. 7.6, we discretize the feedback linearized system at 10Hz and formulate the controller with a horizon of $N = 15$ steps. The cost function has parameters $Q = I$ and $R = 10^{-2}$, and $W = [-10^{-2}, 10^{-2}]^2$. The state trajectories
Figure 7.4: The states and their estimates of the feedback linearized and non-linear running example. Recall that $z_1 = x_1$ therefore to reduce clutter, we only plot the first state only for the feedback linearized system. Color in online version.
(and estimates) for the nonlinear and linearized systems are shown in Fig. 7.4. Note that the states converge to the equilibrium 0. The input $u$ is shown in Fig. 7.5, and it can be noted that $u_k \in U$ for all $k$.

### 7.5.2 Single link flexible joint manipulator

We consider the single link flexible manipulator system $S$, also used in [Son et al. 2001](#) and [Seidi et al. 2012](#), shown in Fig. 7.6 whose dynamics are given by:

$$
S : \begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 \\
\dot{x}_3 \\
\dot{x}_4
\end{bmatrix} = \begin{bmatrix}
x_2 \\
-\frac{mg l}{I} \sin(x_1) - \frac{\sigma}{J} (x_1 - x_3) \\
x_4 \\
\frac{\sigma}{J} (x_1 - x_3)
\end{bmatrix} + \begin{bmatrix}
0 \\
0 \\
0 \\
\frac{1}{J}
\end{bmatrix} u
$$

This models a system where a motor, with an angular moment of inertia $J = 1$, is coupled to a uniform thin bar of mass $m = 1/g$, length $l = 1m$ and moment of inertia $I = 1$, through a flexible torsional string with stiffness $\sigma = 1$ and $g = 9.8ms^{-2}$. 

---

119
States $x_1$ and $x_2$ are the angles of the bar and motor shaft in radians, respectively, and $x_3, x_4$ are their respective rotational speeds in radians/sec. The safe set is the box $X = [-\pi/4, \pi/4] \times [-\pi/4, \pi/4] \times [-\pi, \pi] \times [-\pi, \pi]$. The input torque $u$ is bounded in $U = [\underline{u}, \overline{u}] = [-10, 10]N \cdot m$. The estimation error $e = \dot{x} - x$ is bounded in $E = [-\pi/180, \pi/180] \times [-\pi/180, \pi/180] \times [-10^{-3}, 10^{-3}]^4$.

The diffeomorphism $T$ is given by:

$$z = T(x) = \begin{bmatrix} x_1 \\ x_2 \\ -\frac{mgI}{I} \sin(x_1) - \frac{\sigma I}{I} (x_1 - x_3) \\ \frac{mgI}{I} x_2 \cos(x_1) - \frac{\sigma I}{I} (x_2 - x_4) \end{bmatrix}$$

The input to the feedback linearized system is given by $v = \beta u + \alpha(x)$ where $\beta = \frac{\sigma}{I}$ and

$$\alpha(x) = \frac{mgl}{I} x_2 \sin(x_1) + \frac{\sigma^2}{IJ} (x_1 - x_3) - (\frac{mgl}{I} \cos(x_1) - \frac{\sigma}{I})(\frac{mgl}{I} \sin(x_1) + \frac{\sigma}{I} (x_1 - x_3))$$

The feedback linearized system $S_{fl}$ has the dynamics: $\dot{z}_1 = z_2$, $\dot{z}_2 = z_3$, $\dot{z}_3 = z_4$, $\dot{z}_4 = v$.

A global inner approximation of the $v$ input set is computed, via interval arithmetic, as $V_{inner-global} = [\max_{x \in X} \alpha(x) + \beta \underline{u}, \min_{x \in X} \alpha(x) + \beta \overline{u}]$. Similarly, the inner approximations $V_{k+jk}$ are computed online by interval arithmetic as $V_{k+jk} = [\max_{x \in X_{k+jk}} \alpha(x) + \beta \underline{u}, \min_{x \in X_{k+jk}} \alpha(x) + \beta \overline{u}]$. Using the procedure of Sec. 7.4.4 the set of safe states for $S_{fl}$ is given by $Z = [-0.5121, 0.5121]^2 \times [-2.5347, 2.5347] \times$
Figure 7.7: The states and their estimates of the feedback linearized and non-linear manipulator. Recall that \( z_1 = x_1 \) and \( z_2 = x_2 \), therefore to reduce clutter, we only plot first two states only for the feedback linearized system. Color in online version.

\([-2.5603, 2.5603]\). Also \( X_0 = [-0.4655, 0.4655]^2 \times [-2.7598, 2.7598] \times [-2.793, 2.793] \). Comparing it to the set \( X \), it shows that we can stabilize the system starting from initial states in a significantly large region in \( X \).

We applied our controller to the above system with a discretization rate of 10Hz and MPC horizon \( N = 10 \). Fig. 7.7 show the states of the feedback linearized system \( S_{fl} \). They converge to the origin in the presence of estimation error, while respecting all constraints. Fig. 7.7 also shows \( x_3 \) and \( x_4 \): they also converge to zero (and remember \( x_1 = z_1 \) and \( x_2 = z_2 \)).

Fig. 7.8 shows the input \( v \) to \( S_{fl} \) along with the global inner approximation \( V_{inner-global} \) and the \( x \)-dependent inner approximations at the instant when the control is applied, \( V_{k|k} \) computed online. Note that the bounds computed online allow for significantly more control action compared to the conservative global inner approximation. Finally, Fig. 7.8 also shows the input \( u \) applied to the non-linear system (and its bounds), which robustly respects its constraints \( u \in U \).
Figure 7.8: Inputs $v$ and $u$ and their bounds for the manipulator example. Color in online version.
7.6 Discussion

In this chapter we present the first algorithm for robust control of a non-linear system with estimation errors and state and input constraints via feedback linearization and Robust MPC. We develop a method to compute linear constraints on the state and inputs of the feedback linearized system such that the non-linear system respects its state and input constraints under bounded state estimation errors. We demonstrate the applicability of our approach on a planar system and a flexible link manipulator example. Results show that the control algorithm stabilizes the systems while ensuring robust constraint satisfaction. While we only evaluated our approach for single input systems, the formulation and set computations involved hold as is for multi-input systems as well.

Limitations of the approach mostly have to do with the numerical limitations involved in computing the constraint sets. E.g., in the manipulator example, the set of states $X_0$ from which we can control the system is a strict subset of the set of safe states $X$. Similarly in the computation of the input and error sets, over-conservatism is potentially a problem.

Ongoing work focuses on implementing this approach for evaluation on a $1/10^{th}$ scale autonomous car, running a low power embedded platform. Real-time online reachability [Bak et al. 2014], interval arithmetic, and support function based computations for $Z_{k+j|k}$ should allow for fast enough computation of the linear constraints for the RMPC optimization. In Pant et al. [2015a], we have already shown that CVXGEN [Mattingley and Boyd 2012] is fast enough to solve Quadratic programs with linear constraints on low-powered embedded platforms at high enough sampling rates to allow for satisfactory control of a real system.

7.7 Proof of main result

7.7.1 Constraints of successive MPC problems

We are now ready to state and prove a key lemma regarding the evolution of the state, error and input sets between MPC optimization problems. This lemma will be key to proving recursive feasibility of the MPC controller, since it allows us to show that the constraint sets of one problem, at time $k$, are appropriate supersets of the constraint sets of the next problem, at time $k + 1$.

Lemma 5. Let $X_{k+j|k}$ be the $j$-step outer-approximate reach set computed at time $k$ by a reachability tool as described in Sec. 7.4.1.
Let $\tilde{W}_{k+j|k}$ be the set defined in (7.18).
Let $\tilde{E}_{k+j|k}$ be the error set computed using (7.16), (7.17) by substituting $E \leftarrow \tilde{E}_{k|k}$.
Let $V_{k+j|k} = V(X_{k+j|k})$ and $\tilde{V}_{k+j|k} = V(X_{k+j|k})$
Then the following hold for all $k \geq 0, j \geq 1$: 123
(7.21a) \[ \tilde{z}_{k+j+1|k} = z^*_{k+j+1|k} + L_j \hat{w}_{k+1}, \forall j \in [0:N] \]

(7.21b) \[ \tilde{z}_{k+N+2|k+1} = A \tilde{z}_{k+N+1|k+1} + B \hat{v}_{k+N+1|k+1} \]

(7.21c) \[ \hat{v}_{k+j+1|k+1} = v^*_{k+j+1|k} + KL_j \hat{w}_{k+1}, \forall j \in [0:N-1] \]

(7.21d) \[ \hat{v}_{k+N+1|k+1} = K \tilde{z}_{k+N+1|k+1} \]
First we will show that the input and state constraints are satisfied by $v_k$ and $\bar{z}_{k+1}$, then prove feasibility of the above candidate solution for $P_{k+1}(\hat{z}_{k+1})$.

**Validity of input and next state:** The next state is:

$$z_{k+1} = A z_k + B v_k + w_k$$

$$= A(\hat{z}_k - \tilde{e}_k) + B v^*_{k|k} + w_k$$

$$= A\hat{z}_k + B v^*_{k|k} - \tilde{e}_{k+1} + (w_k + \tilde{e}_{k+1} - A\tilde{e}_k)$$

$$= A z^*_{k|k} + B v^*_{k|k} - \tilde{e}_{k+1} + \hat{w}_{k+1}$$

($P_k(\hat{z}_k)$ initialization)

$$= z^*_{k+1|k} - \tilde{e}_{k+1} + \hat{w}_{k+1}$$

(7.22)

By feasibility of the solution at time $k$,

$$z^*_{k+1|k} \in \mathbb{Z}_{k+1|k} = Z \ominus (-\tilde{E}_{k+1|k}) \ominus L_0 \hat{W}_{k+1|k}$$

Therefore, $z_{k+1} \in Z$ and so $x_{k+1} \in X$.

Moreover, by the feasibility of $v^*_{k|k}$ for $P_k(\hat{z}_k)$ and by the definition of $V_{k|k}$, $v_k = v^*_{k|k} \in V_{k|k}$, which implies that $u_k \in U$.

Hence, if $P_k(\hat{z}_k)$ is feasible, then the applied input at time step $k$ and the resulting next state $z_{k+1}$ (and hence $x_{k+1}$) are admissible under all possible disturbances. The next part of the proof will focus on showing that the candidate solution of Eq. (7.21a) is indeed feasible for $P_{k+1}(\hat{z}_{k+1})$ by proving that it meets all the constraints.

**Initial Condition:** Recall from (7.5) that $\hat{z}_{k+1} = A\hat{z}_k + B v_k + \hat{w}_{k+1}$. Also by the construction of the candidate solution,

$$\hat{z}_{k+1|k+1} = z^*_{k+1|k} + L_0 \hat{w}_{k+1}$$

$$= A z^*_{k|k} + B v^*_{k|k} + \hat{w}_{k+1}$$

(7.23a)

Since $z^*_{k|k} = \hat{z}_k$ and $v^*_{k|k} = v_k$, by the two equations above, we have

$$\hat{z}_{k+1|k+1} = \hat{z}_{k+1}$$

(7.24)

Hence, the candidate solution does indeed satisfy the initial condition for $P_{k+1}(\hat{z}_{k+1})$. Next we show that the candidate solution satisfies the nominal dynamics:
**Nominal Dynamics:** For \(0 \leq j < N\), we have:

\[
\begin{align*}
\tilde{z}_{k+j+2|k+1} & = z_{k+j+2|k}^* + L_{j+1}\hat{w}_{k+1} \\
& = A\tilde{z}_{k+j+1|k+1} + B\tilde{v}_{k+j+1|k} + L_{j+1}\hat{w}_{k+1}
\end{align*}
\]

By the construction of the candidate solution

\[
\begin{align*}
A(\tilde{z}_{k+j+1|k+1} - L_j\hat{w}_{k+1}) + B(\tilde{v}_{k+j+1|k+1} - KL_j\hat{w}_{k+1}) \\
& = A\tilde{z}_{k+j+1|k+1} + B\tilde{v}_{k+j+1|k+1} + KL_j\hat{w}_{k+1} \\
& = A\tilde{z}_{k+j+1|k+1} + B\tilde{v}_{k+j+1|k+1} + L_{j+1}\hat{w}_{k+1}
\end{align*}
\]

For \(j = N\), by construction \(\tilde{z}_{k+N+2|k+1} = A\tilde{z}_{k+N+1|k+1} + B\tilde{v}_{k+N+1|k+1}\). Hence, the candidate solution does indeed satisfy the nominal dynamics.

**State Constraints:** To show feasibility of the candidate solution w.r.t. the state constraints, we need to show that \(\bar{z},...,N - \hat{j} + 1\), i.e. for \(w\)

\[
\text{State Constraints: To show feasibility of the candidate solution w.r.t. the state constraints, we need to show that } \bar{z},...,N - \hat{j} + 1. \text{ Hence, the candidate solution does indeed satisfy the nominal dynamics.}
\]

Also, let us write the state constraints for all \(j = 0, \ldots, N\) for the problem at time \(k + 1\), i.e. for \(P_k(\hat{z}_k)\) for \(j = 0, \ldots, N - 1\), we have:

\[
Z_{k+j+1|k} = Z \ominus_{i=0}^{j-1} L_i \hat{W}_{k+j-i|k} \ominus (\tilde{E}_{k+j+1|k})
\]

Using points 2) and 3) from Lemma

\[
Z \ominus_{i=0}^{j-1} L_i \hat{W}_{k+j-i|k} \ominus (\tilde{E}_{k+j+1|k}) \subseteq Z \ominus_{i=0}^{j-1} L_i \hat{W}_{k+j-i|k+1} \ominus (\tilde{E}_{k+j+1|k+1})
\]
And using Eq. (7.7.2), this implies for all $j = 0, \ldots, N - 1$

$$\bar{z}_{(k+1)+j|k+1} \in \mathbb{Z}_{k+1+j|k+1}$$

Now for $j = N$, $\bar{z}_{k+N+1|k+1} = z_{k+N+1|k}^* + L_N \hat{w}_{k+1}$. From the terminal constraint we have $[z_{k+N+1|k}^* v_{k+N|k}] \in P_f = C_p \ominus \hat{L}_N \hat{F}_{\max}$. Since $w_{k+1} \in \hat{W}_{\max}$, and by the construction of the candidate solution

$$[\bar{z}_{k+N+1|k+1} \bar{v}_{k+N|k+1}] \in C_p$$ (7.27)

Remember, by definition of the invariant set, $C_p \in P_N(\hat{E}_{\max}, \hat{E}_{\max})$, and since by definition of $\hat{E}_{\max}$ and Eq. (7.8) we have $P_N(\hat{E}_{\max}, \hat{E}_{\max}) \subseteq \mathbb{Z}_{k+1+N|k+1} \times V_{k+1+N|k+1}$, or $C_p \in \mathbb{Z}_{k+1+N|k+1} \times V_{k+1+N|k+1}$. This implies that $\bar{z}_{k+N+1|k+1} \in \mathbb{Z}_{k+1+N|k+1}$ and additionally, $v_{k+N|k+1} \in V_{k+1+N|k+1}$. Therefore, the set constraints are met by candidate solution $\forall j = 0, \ldots, N$.

**Input Constraints:** For the inputs, we show that the candidate solution, $\bar{v}_{k+j+1|k+1}, j = 0, \ldots, N - 2$, satisfies the input constraints for $P_{k+1}(\hat{z}_{k+1})$ by using a similar argument as that used for the state constraints. Let us re-write the input constraints for $P_k(\hat{z}_k)$ for $j = 0, \ldots, N - 2$,

$$V_{k+j+1|k} = V_{k+j+1|k} \ominus_i^j K L_i \hat{W}_{k+j+1-i|k}$$ (7.28a)

$$= V_{k+j+1|k} \ominus K L_j W_{k+1|k} \ominus_i^j K L_i \hat{W}_{k+j+1-i|k}$$ (7.28b)

$$= V_{k+j+1|k} \ominus K L_j W_{k+1|k} \ominus_i^{j-1} K L_i \hat{W}_{k+j+1-i|k}$$ (7.28c)

Let us also re-write the input constraints for $P_{k+1}(\hat{z}_{k+1})$ for $j = 0, \ldots, N - 1$,

$$V_{k+1+j|k+1} = V_{k+j+1|k+1} \ominus_i^{j-1} K L_i \hat{W}_{k+j-i|k+1}$$ (7.29)

By construction of the candidate, we have $\bar{v}_{k+j+1|k+1} = v_{k+j+1|k}^* + K L_j \hat{w}_{k+1}$. Also by feasibility of the algorithm at time $k$, we have $v_{k+j+1|k}^* \in V_{k+j+1|k}$, and by definition, $L_j \hat{w}_{k+1} \in L_j \hat{W}_{k+1|k}$. Therefore, by definition of the Pontryagin difference and Eq. (7.28) we have $\forall j = 1, \ldots, N - 1$,

$$\bar{v}_{(k+1)+j|k+1} \in V_{k+j+1|k} \ominus_i^{j-1} L_i \hat{W}_{k+j-i|k}$$ (7.30a)

Using points 3) and 4) from Lemma 5

$$V_{k+j+1|k} \ominus_i^{j-1} L_i \hat{W}_{k+j-i|k} \subseteq V_{k+j+1|k+1} \ominus_i^{j-1} L_i \hat{W}_{k+j-i|k+1}$$ (7.30b)

And using Eq. (7.29) this implies

$$\bar{v}_{(k+1)+j|k+1} \in V_{k+1+j|k+1}$$ (7.30c)

Note, for $j = N - 1$, we have already shown in the proof for the state constraints that by definition of the invariant set $C$, $v_{k+N|k+1} \in V_{k+1+N-1|k+1}$ by respecting
an even tighter constraint. For the last input for $j = N$, we have \( \ddot{v}_{k+1+N|k+1} = K \ddot{z}_{k+1+N|k} \); we show that it is inside the (joint) terminal constraint \( P_f \), and hence is feasible.

**Terminal Constraints:** Finally, we need to show that \([\ddot{z}_{k+1+N+2} \ddot{v}_{k+1+N+1}]' \in P_f\). This can be shown using the construction of the terminal set and the candidate solution. From Equation 7.21a, we have:

\[
\begin{align*}
\ddot{z}_{k+1+N+2|k+1} &= A \ddot{z}_{k+1+N+1|k+1} + B \ddot{v}_{k+1+N+1|k} \quad \text{(7.31a)} \\
\ddot{v}_{k+1+N+1|k+1} &= K \ddot{z}_{k+1+N+1|k+1} \quad \text{(7.31b)}
\end{align*}
\]

Concatenate these two into \( p_{k+1+N+2|k+1} = [\ddot{z}_{k+1+N+2|k+1} \ddot{v}_{k+1+N+1|k+1}]' \). Also \( p_{k+1+N+1|k+1} = [\ddot{z}_{k+1+N+1} \ddot{v}_{k+N}]' \) was in \( C_p \) as shown previously (Eq. 7.27). Therefore, by definition of the invariant set \( C_p \) (Equation 7.11), we have that \( p_{k+1+N+2|k+1} + \hat{L}_N \hat{F} w_{k+1|k} \in C_p \) for all \( w_{k+1|k} \in \hat{W}_{k+1|k} \subseteq \hat{W}_{\max} \). Therefore \( p_{k+1+N+2|k+1} \in C_p \ominus \hat{L}_N \hat{F} \hat{W}_{\max} = P_f \). Therefore the terminal constraint is also met.

With this, we have the proof for Theorem 1 as we have shown that feasible solution at time step \( k \) for \( P_k(\ddot{z}_k) \) implies that the applied input \( v_k \) is feasible, the next state \( \ddot{z}_{k+1} \in Z \) and the problem \( P_{k+1}(\ddot{z}_{k+1}) \) is feasible at time \( k + 1 \), and hence \( P_{k+2}(\ddot{z}_{k+2}) \) is feasible for time step \( k + 2 \) and so on. ■
Chapter 8

Related work

The following sections outline some of the relevant research for the topics presented in this document.

8.1 Related work for chapter 3

Once the requirements are expressed as an Metric Temporal Logic (or Signal Temporal Logic) formula, there are broadly two ways of designing a controller that satisfies the formula (fulfills the requirements). The first method automatically creates a Mixed Integer Program (MIP) from the semantics of the STL formula and solves the MIP to yield a satisfying control sequence [Raman et al. 2014], [Saha and Julius 2016], [Karaman and Frazzoli 2011]. These works extend the encoding of Bemporad and Morari [1999b] for Mixed Logical Dynamical systems to the problem of making dynamical systems satisfy STL specifications. Similar encoding has also been used for hybrid systems with piece-wise affine dynamics [Frick et al. 2019].

The second method, upon which methods presented in this document build upon, uses the robustness of MTL specifications [Fainekos et al. 2006], [Donzé and Maler 2010a]. Robustness is a rigorous notion that has been used successfully for the testing and verification of automotive systems [Fainekos et al. 2012], [Dreossi et al. 2015], medical devices [Sankaranarayanan and Fainekos 2012a], and general CPS.

Current approaches to optimizing the robustness fall into four categories: the use of heuristics like Simulated Annealing [Nghiem et al. 2010], cross-entropy [Sankaranarayanan and Fainekos 2012b] and RRTs [Dreossi et al. 2015]; non-smooth optimization [Abbas and Fainekos 2013]; Mixed Integer Linear Programming (MILP) [Raman et al. 2014], [Saha and Julius 2016]; and iterative approximations [Abbas and Fainekos 2011], [Abbas et al. 2014], [Deshmukh et al. 2015]. Black-box heuristics are the most commonly used approach. The clear advantage of these methods is that they do not require any special form of the objective function: they simply need to evaluate it at various points of the search space, and use its value as feedback to decide on the next point to try. A significant shortcoming is their lack of guarantees as explained.
earlier. Because the robustness is non-smooth, the work in Abbas and Fainekos [2013] developed an algorithm that decreases the objective function along its sub-gradient. This involved a series of conservative approximations and was restricted to the case of safety formulae. In Raman et al. [2014], the authors encoded the MTL formula as a set of linear and boolean constraints (when the dynamical system is linear), and used Gurobi to solve them. MILPs are NP-complete, non-convex, and do not scale well with the number of variables. The sophisticated heuristics used to mitigate this make it hard to characterize their runtimes, which is important in control - see examples in Raman et al. [2014] and chapter 3. In general, MILP solvers can only guarantee achieving local optima. Note also that Raman et al. [2014] requires all constraints to be linear, so all atomic propositions must involve half-spaces \( p : a'x \leq b \), which is not a restriction in the current work. Another MILP based approach is presented in Saha and Julius [2016] where constraints are added when necessary, in order to reduce MILP complexity. The work closest to the approach presented in chapter 3 is Abbas and Fainekos [2011], Abbas et al. [2014]. There, the authors considered safety formulas, for which the robustness reduces to the minimum distance between \( x \) and the unsafe set \( U \). By sub-optimally focusing on one point on the trajectory \( x \), they replaced the objective by a differentiable indicator function for \( U \) and solved the resulting problem with gradient descent. The use of fast smooth approximations of robustness circumvents most of the above issues and gets closer to real-time control by robustness maximization.

Linear Temporal Logic (LTL) can also be used to encode the requirements considered in this document, e.g. the bounded time eventually can be encoded as disjunctions of next (and nested next) operators in LTL. However state-of-the-art approaches for synthesis with LTL specifications cannot generally scale to the dimensions of our problems, or work with fragments of LTL that do not include the next operator, e.g. Kloetzer and Belta [2008] deals with a fraction of LTL, LTL\( X \) that does not contain the next operator. Similarly the LTL-based motion planning approach of Fainekos et al. [2005] also restricts specifications to LTL\( X \). The swarm-planning approach of Kloetzer and Belta [2006] has been shown to scale to up to 30 agents, although in a planar workspace, although all agents have the same specification, which is again restricted to LTL\( X \). State-of-the-art methods for bounded time LTL (LTL\( f \)) synthesis, e.g. Camacho et al. [2018], De Giacomo and Vardi [2015], Saha et al. [2014] show promise but cannot deal with the problems covered in this document (also see section 8.2).

The methods presented in this document (chapters 3 and 4) differ significantly from these approaches as they are not restricted to swarm applications, explicitly deal with bounded-time operators, e.g. the eventually and until operator over time intervals, do not rely on discretization of the workspace and also offer continuous time guarantees.
8.2 Related work for chapter 4

The mission planning problem for multiple agents has been extensively studied. Most solutions work in an abstract grid-based representation of the environment [Saha et al. 2014, DeCastro et al. 2017], and abstract the dynamics of the agents [Desai et al. 2017, Aksaray et al. 2016]. As a consequence they have correctness guarantees for the discrete behavior but not necessarily for the underlying continuous system. Multi-agent planning with kinematic constraints in a discretized environment has been studied in [Honig et al. 2016] with application to ground robots. Planning in a discrete road map with priorities assigned to agents has been studied in [van den Berg and Overmars 2015] and is applicable to a wide variety of systems. Another priority-based scheme for drones using a hierarchical discrete planner and trajectory generator has been studied in [Ma et al. 2016]. Most of these use Linear Temporal Logic (LTL) as the mission specification language, which doesn’t allow explicit time bounds on the mission objectives. The work in [Aksaray et al. 2016] uses STL. Other than [van den Berg and Overmars 2015] and [Ma et al. 2016], none of the above methods can run in real-time because of their computational requirements. While [van den Berg and Overmars 2015] and [Ma et al. 2016] are real-time, they can only handle the Reach-Avoid mission, in which agents have to reach a desired goal state while avoiding obstacles and each other. [Saha et al. 2014] uses a subset of LTL, safe-LTL, that allows them to express reach-avoid specifications with explicit timing constraints. However their approach requires a discretization of the workspace. The robot behavior is also discretized through a fixed set of motion primitives. They evaluate their work experimentally on a quad-rotor robot but restrict the allowable motion to 2 dimensions to keep the problem tractable. Their motion primitives, although chosen elegantly, also restrict the quadrotor robots to stop-and-go trajectories, while our approach (in addition to being able to handle stop-and-go motion) generates much more free-form trajectories which allow for exploring the full range of motion of the robot.

In a more control-theoretic line of work, control of systems with STL or Metric Temporal Logic (MTL) specifications without discretizing the environment or dynamics has been studied in [Raman et al. 2014, Sadraddini and Belta 2015, Pant et al. 2017b]. These methods are potentially computationally more tractable than the purely planning-based approaches discussed earlier, but are still not applicable to real-time control of complex dynamical systems like quadrotors (e.g. see [Pant et al. 2017b]), and those that rely on Mixed Integer Programming based approaches [Raman et al. 2014] do not scale well. Stochastic heuristics like [Abbas et al. 2013] have also been used for STL missions, but offer very few or no guarantees. Non-smooth optimization has also been explored, but have been applied only to safety properties [Abbas and Fainekos 2013].

In the method presented in chap. 4, we focus on multi-rotor systems, and work with a continuous representation of the environment, i.e. do not have to discretize the workspace and rely on simplified motion primitives, and take into account the
behavior of the trajectories of the quadrotor. With the mission specified as a STL formula, we maximize a smooth version of the robustness (Aksaray et al. 2016, Pant et al. 2017b). This, unlike a majority of the work outlined above, allows us to take into account explicit timing requirements. Our method also allows us to use the full expressiveness of STL, so our approach is not limited to a particular mission type. Finally, unlike most of the work discussed above, we offer guarantees on the continuous-time behavior of the system to satisfy the spatio-temporal requirements. Through simulations and experiments on actual platforms, we show real-time applicability of our method for simple cases, as well as the scalability in planning for multiple quadrotors in a constrained environment for a variety of mission specifications.
8.3 Related work for chapter 5

Existing mission planner software for autonomous drone operations like ArduPilot mission planner [apm] and QGroundControl [qgr] offer UAS enthusiasts the ability to quickly plan out autonomous UAS flights by sequencing multiple simple operations (like take-off, hover, go to a way-point, land) together. However these planners either cannot handle missions involving multiple UAS and complicated requirements like co-ordination between UAS or completing tasks within given time intervals, or require hand-crafted sequences of maneuvers to meet the requirements in a safe manner. We propose a tool that can inherently deal with multi-agent missions as well as timing constraints on completion of tasks while guaranteeing that planned flight paths are safe. As opposed to existing mission planning software, our tool does not require the user to explicitly plan out maneuvers for the drones to execute to follow out a mission, e.g. in the case where two UAS have to enter the same region during the same time interval, our method generates trajectories that ensure the two UAS do so without crashing into each other without any user based scheduling of which drones enters first.

The tool presented here relies on interpreting a mission as a STL specification and generating trajectories that satisfy it. While there are multiple methods and tools that aim to solve such a problem, e.g. Mixed Integer Programming-based [Raman et al. 2014] and based on stochastic heuristics [Annapureddy et al. 2011], we use an underlying method [Pant et al. 2018] that is tailored for generating trajectories for multi-rotor UAS, including those that allow hovering, to satisfy STL specifications in continuous-time. A detailed comparison can be found in [Pant et al. 2018, 2017c].
8.4 Related work for chapter 6

Algorithms, that can be interrupted at any point at run-time and still return an acceptable solution, are called Anytime Algorithms [Boddy and Dean, 1989]. Such algorithms generally return solutions with improving quality of output the longer they run for. A subset of these are Contract Algorithms [Zilberstein, 1996] which can be interrupted only at a finite number of pre-agreed upon times. Chapter 6 presents the design of a Contract-driven perception-based state estimator, but significantly expands the notion of a contract to now include the quality of the solution (estimation error in our case) as well as the computation time.

Anytime algorithms have found particular importance in the field of graph search [Likhachev et al., 2008], evaluation of belief networks [Wellman and Liu, 1994] and GPU architectures [Mangharam and Saba, 2011], [Pant et al., 2015b]. With autonomous systems gaining popularity, computationally overloaded systems with real-time requirements are becoming the norm. This has generated interest in the development of anytime algorithms in the field of control theory, with Quevedo and Gupta [Quevedo and Gupta, 2013], Bhattacharya and Balas [Bhattacharya and Balas, 2004], and Fontanelli et al. [Fontanelli et al., 2008] exploring this line of research. Anytime algorithms have also found widespread use in the field of motion planning [Narayanan et al., 2012], [Jha et al., 2016], [Choudhury, 2017], [Karaman et al., 2011].

The work presented in chapter 6 contrasts considerably with these efforts as the assumption of anytime computation is not on the controller or planning side but on the perception-based state estimation component of the feedback control loop. The loop is closed by the control algorithm presented here that decides the contract for the anytime state estimator at run-time. Also differing from the works discussed above, which require instantaneous and perfect full state access for the controller, our control algorithm takes into account the computation time and the estimation error of the perception-based estimators that are common in autonomous systems. The recent work of Falanga et al. [Falanga et al., 2018] also tackles the problem of co-designing the perception and the control, but does so as a joint optimization that takes into account both the perception and the control. Our work differs from this significantly as we introduce the notion of contracts to decouple the perception-based estimator’s performance and the control optimization. Our method also explicitly incorporates the timing and the estimation performance of the perception-based estimator in the control design, and can be used for the off-the-shelf perception-based estimators (for an example, see section 6.6.3).

The methods developed in chapter 6 rely on a Robust Model Predictive Control formulation, a comprehensive survey of which is found in Bemporad and Morari [1999a]. Unlike the traditional methods that handle uncertainty in state estimation [Richards and How, 2006] as well delays in actuation [Richards and How, 2005b], our approach (that builds upon [Richards and How, 2005b]) works with possibly time-varying uncertainty in the state estimate, as well as delays that are time-varying without sacrificing guarantees on stability and recursive feasibility.
In the domain of real-time systems, Worst Case Analysis, along with Logical Execution Time semantics are used in Frehse et al. [2014] to imbue a controller with information of the timing characteristics of the closed loop implementation. On the other hand, our approach involves profiling the estimation algorithm in a direct manner to get timing and estimation error characteristics. While Frehse et al. [2014] involves formally verifying a given controller, we design a control algorithm that is correct by construction and takes advantage of delay/accuracy trade-offs in real-time.

In the context of autonomous multi-rotor UAVs, the effect of increasing the computation time of task on the overall performance of the system has been analyzed in de Niz et al. [2012] by using a resource allocation algorithm similar to QRAM Rajkumar et al. [1997]. Our approach contrasts with this as we focus on the execution time of a particular task, the perception-based state estimator, which directly impacts the closed loop control performance. In addition to this, we also formulate a controller that provides mathematical guarantees on the system’s performance.

Finally, in the area of computer architecture, approximate computing approaches Sidiroglou-Douskos et al. [2011], Carbin et al. [2013], St. Amant et al. [2014] have been explored to get savings in time or energy through performing a computation in an approximate manner, rather than precisely. While anytime algorithms and approximate computing have a common high-level goal, approximate computing methods are run-to-completion and lack a feedback mechanism to permit computation and resources to be balanced dynamically. It is also worth noting that time and energy scale that our approach deals with are much greater than those which concern approximate computing.
8.5 Related work for chapter 7

Previous work on nonlinear MPC with feedback linearization assumed the state \( x(t) \) is perfectly known to the controller at any moment in time \cite{Simon2013}. In practice, only a state estimate \( \hat{x}(t) \) is available, and \( \hat{x}(t) \neq x(t) \). Thus a controller designed to work optimally when operating on the true state \( x \) is in general sub-optimal when operating on \( \hat{x} \) (and may even lead to instability). Robust MPC (RMPC) \cite{Bemporad1999} has been investigated as a way of handling state estimation errors for linear systems \cite{Richards2005,Kouvaritakis2015} and nonlinear systems \cite{Mayne2007,Streif2014,Oliveira1994}, but not via feedback linearization. In particular, for non-linear systems, \cite{Mayne2007} develops a non-linear MPC with tube like constraints for robust feasibility, but involves solving two (non-convex) optimal control problems. In \cite{Streif2014}, the authors solve a non-linear Robust MPC through a bi-level optimization that involves solving a non-linear, non-smooth optimization which is challenging. \cite{Streif2014} also guarantees a weaker form of recursive feasibility than \cite{Richards2005} and what we guarantee in this work. In \cite{Zhao2014} the authors approximate the non-linear dynamics of a quadrotor by linearizing it around hover and apply the RMPC of \cite{Richards2005} to the linearized dynamics. This differs significantly from our approach, where we formulate the RMPC on the feedback linearized dynamics directly, and not on the dynamics obtained via Jacobian linearization of the non-linear system. Existing work on MPC via feedback linearization and input/state constraints has also assumed that either \( T \) is the identity, which simplifies the subsequent stability and performance analysis \cite{Simon2013}, or, in the case of uncertainties in the parameters, that there are no state constraints \cite{Son2001}. A non-identity \( T \) is problematic when the state is not perfectly known, since the state estimation error \( e = \hat{x} - x \) maps to the linearized dynamics via \( T \) in non-trivial ways greatly complicating the analysis. In particular, the error bounds for the state estimate in \( z \)-space now depend on the current nonlinear state \( x \). One of the complications introduced by feedback linearization is that the bounds on the input \( (u \in U) \) may become a non-convex state-dependent constraint on the input \( v \) to \( S_f \): \( V = \{ v(x, U) \leq v \leq v(x, U) \} \). In \cite{Simon2013} forward reachability is used to provide inner convex approximations to the input set \( V \). A non-identity \( T \) increases the computational burden since the non-linear reach set must be computed (with an identity \( T \), the feedback linearized reach set is sufficient).


Michael Carbin, Sasa Misailovic, and Martin C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *Proceedings of the*


Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3d path planning algorithms. Journal of Control Science and Engineering, 2016:5, 2016.
